UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Gray Box Performance Modeling of In-Memory Distributed Transactional Platforms

Diego Didona

Supervisor: **Doctor Paolo Romano**

Thesis approved in public session to obtain the PhD degree in Information Systems and Computer Engineering

Jury final classification: Pass with distinction

Jury

Chairperson: Chairman of the IST Scientific Board
Members of the Committee:
Doctor Guillaume Pierre
Doctor Paolo Romano
Doctor João Manuel dos Santos Lourenço
Doctor Francisco António Chaves Saraiva de Melo

2015

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Gray Box Performance Modeling of In-Memory Distributed Transactional Platforms

Diego Didona

Supervisor: **Doctor Paolo Romano**

Thesis approved in public session to obtain the PhD degree in Information Systems and Computer Engineering

Jury final classification: Pass with distinction

Jury

Chairperson: Chairman of the IST Scientific Board
Members of the Committee:
Doctor Guillaume Pierre, Full Professor, Université de Rennes 1, France
Doctor Paolo Romano, Professor Associado, Instituto Superior Técnico, Universidade de Lisboa
Doctor João Manuel dos Santos Lourenço, Professor Auxiliar, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Doctor Francisco António Chaves Saraiva de Melo, Professor Auxiliar, Instituto Superior Técnico, Universidade de Lisboa

Funding Institutions
Fundação para a Ciência e a Tecnologia
European Commission
COST (European Cooperation in Science and Technology)

2015

Acknowledgements

I cannot but begin this document by expressing my deepest gratitude to my advisor, Professor Paolo Romano. Yet, I cannot thank him for his guidance and his teaching, as I know he would reply just like he has humbly been doing since the very first day: "it is my job and duty". Hence, I am going to thank him for his true friendship outside the office and for his constant presence and support in the hardest moments: for every all-nighter pulled off up to the dawn and beyond, he has always been at the other side of the desk or of the Skype window, never giving up first. This kind of commitment hardly falls within the job specifications or among the duties of an ordinary advisor. Finally, I want to sincerely thank him for all the opportunities he gave me throughout the years, and all the doors that he has opened (or at least knocked on) to me: they have determined my path up to now, and they are shaping my future. I am happy and, undeservedly, a little bit proud of being his first *full* PhD student to graduate.

I also want to warmly thank my grand-advisor, Professor Francesco Quaglia, who guided my first steps in the slippery field of research. His teachings and inspirational words have taken root in my mind, and I hope to have honored them so far, and to continue to do so in the future.

To conclude with my academic ancestors, I want to thank my grand-grand-advisor, Professor Bruno Ciciani, for his support in my last (sad) days in Rome and for showing me that a good spirit is sometimes the key to approach research (and life in general).

I also want to dedicate a few words to people who have been instrumental, in several ways, to the completion of my PhD. First and foremost, I want to express my gratitude to Sebastiano Peluso, with whom I shared the first, and probably most exciting, part of this adventure. With him, I faced my first deadline and wrote my first paper, celebrated the first *accept* and dealt with the first *reject*; I shared with him my first lab (in Rome) and my first flat (in Lisbon). And, most importantly, the creaky and ice-cold room of Residencial Marisela: this is a bond that lasts

forever. Much like the echoes of the whistle of Trinity's song rising in the morning: they still give me goosebumps, reminding me of those glorious days. I want especially to thank him for his friendship, and for the enthusiasm and rigor he approaches research with, which have been truly an inspiration for me.

I would also like to thank all the people I have met during my stay at INESC-ID —from the custodians to the professors— and occasionally abroad, my several co-authors, grad and undergrad students with whom I have worked; but they are so many that I cannot possibly mention all of them. However, I want to especially thank Professor Luís Rodrigues and Professor João Barreto for providing me support and means to start and develop my research, and Professor Pascal Felber and Professor Rachid Guerraoui for having hosted me in their labs during my two brief Swiss missions. As for the peons, ehm, junior researchers, I want to especially thank Maria Couceiro, Nuno Machado, João Paiva, Liliana Rosa and Xavier Villaça for their companionship in and out the walls of INESC-ID. I cannot help but dedicate a special thought to Pedro "Pedrito" Ruivo, with whom I saw Benfica (lose), spent many of my coffee-breaks and faced some of the toughest all-nighters; and to Nuno "the pruner" Diegues, whom I had the pleasure to work with during my last Portuguese project, for sharing with me his uncommon knowledge and research wisdom.

To conclude this section, I would like to express my deepest gratitude to people, or entities, who had nothing to do with or to gain from my PhD journey, and yet stood by my side providing me with invaluable support.

To my whole family, for their unconditional love and care. In particular, to my parents, for their teaching and education and for being, through their very lives, the model I always turn to; for letting me do my mistakes, yet helping me to recover in the aftermath; for minimizing and forgetting delusions and disappointments, while pretending every small accomplishment to be a major success to be celebrated. Specifically, to my mother, who gave up everything for her family: I wish I had her strength. And to my father, who is the most dedicated researcher I know, yet he cannot compute his h-index. I wish I were as genuine as he is.

To my best friends, Matteo, Giacomo and Matteo, whose long-distance presence and hilar-

ious messages have kept my mental sanity during these years... Or have they?

To Annamaria, for her constant moral support, friendship and true kindness; and for the countless dinners and cakes she prepared me, always pleasantly taking into account my dietary habits and quirks.

To `matish`, for having originally generated in me the desire of pursuing my PhD, even if he does not know that.

To Portugal, and in particular the city of Lisbon: I have not even taken care of learning its language, yet it hosted me like one of its very children. *Obrigado*.

To my little okapi, whose smiles, sweetness and beauty just *kept me hanging on* in some imperfect days.

And to music, *that mysterious form of time*.

The work presented in the thesis was supported by the following funding agencies:

- INESC-ID through PEst-OE/EEI/LA0021/2011, PEst-OE/EEI/LA0021/2013 and Incentivo/EEI/LA0021/2014
- European Commission through project CloudTM (FP7/257784) and the EuroTM COST Action (IC1001)
- Fundação para a Ciência e Tecnologia (FCT) through projects Aristos (PTDC/EIA-EIA/102496/2008), SpecSTM (PTDC/EIA- EIA/122785/2010) and GreenTM (EXPL/EEI-ESS/0361/2013)

Thesis Publications List

Most of the material presented in this work can also be found in the following publications:

Journals

1. D. Didona, P. Felber, D. Harmanci, P. Romano, J. Schenker. Identifying the Optimal Level of Parallelism in Transactional Memory Applications. Springer Computing Journal , December 2013.
2. D. Didona, P. Romano, S. Peluso, F. Quaglia. Transactional Auto Scaler: Elastic Scaling of Replicated In-Memory Transactional Data Grids. ACM Transactions on Autonomous and Adaptive Systems (TAAS), Vol. 9, Issue 2, Article 11, July 2014.
3. P. Di Sanzo, F. Quaglia, B. Ciciani, A. Pellegrini, D. Didona, P. Romano, R. Palmieri, S. Peluso. A Flexible Framework for Accurate Simulation of Cloud In-Memory Data Stores. Elsevier Simulation Modelling Practice and Theory Journal (SIMPAT), to appear.

Conferences and Workshops with proceedings

1. R. Palmieri, P. Di Sanzo, F. Quaglia, P. Romano, S. Peluso, D. Didona. Integrated Monitoring of Infrastructures and Applications in Cloud Environments. Workshop on Cloud Computing Projects and Initiatives (CCPI, collocated with Euro-Par), 2011.
2. D. Didona, P. Romano, S. Peluso, F. Quaglia. Transactional Auto Scaler: Elastic Scaling of In-memory Transactional Data Grids. In Proceedings of the ACM 9th International Conference on Autonomic Computing (ICAC), 2012.
3. B. Ciciani, D. Didona, P. Di Sanzo, R. Palmieri, S. Peluso, F. Quaglia, P. Romano. Automated Workload Characterization in Cloud-based Transactional Data Grids. In Proceedings of the IEEE 17th Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS, collocated with IPDPS), 2012.

4. D. Didona, D. Harmanci, P. Felber, P. Romano, J. Schenker. Identifying the Optimal Level of Parallelism in Transactional Memory Applications. In Proceedings of the 1st International Conference on Networked Systems (NETYS), 2013 (**Best paper award**).

5. D. Didona, P. Romano. Self-Tuning Transactional Data Grids: the Cloud-TM Approach. In Proceedings of the IEEE 3rd Symposium on Network Cloud Computing and Applications (NCCA), 2014.

6. D. Didona, P. Romano. Performance Modelling of Partially Replicated In-Memory Transactional Stores. In Proceedings of the IEEE 22nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2014.

7. D. Didona, F. Quaglia, P. Romano, E. Torre. Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning. In Proceedings of the ACM/SPEC 6th International Conference on Performance Engineering (ICPE) , 2015.

8. D. Didona, P. Romano. Hybrid Machine Learning/Analytical Models for Performance Prediction: a Tutorial. In Proceedings of the ACM/SPEC 6th International Conference on Performance Engineering (ICPE) , 2015.

9. D. Didona, P. Romano. Using Analytical Models to Bootstrap Machine Learning Performance Predictors. In Proceedings of the IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2015.

Book chapters

1. M. Couceiro, D. Didona, L. Rodrigues, P. Romano. Self-tuning in Distributed Transactional Memory. In Transactional Memory: Foundations, Algorithms, Tools and Applications. Springer, Lecture Notes in Computing Sciences, 2015

To my parents.

Abstract

Cloud Computing has drastically changed the way modern applications are provisioned: in the Cloud, resources are acquired elastically, i.e., on-the-fly, depending on actual applications' demands, thus allowing for significantly reducing both upfront capital investments and operational costs.

Distributed Transactional Platforms (DTPs) have emerged as an important class of computing platforms for the Cloud: by building upon the familiar construct of *transaction*, programmers can delegate to a distributed middleware the burden of regulating concurrent accesses to critical sections, and be spared from the complexity associated with enforcing data consistency in large scale distributed platforms.

Modeling the performance of DTPs is a fundamental requirement to implement automatic resource provisioning and self-tuning schemes aimed to take advantage of the elastic provisioning capabilities of the Cloud. Performance models, in fact, represent valuable means to estimate the resources demanded by applications to meet their service level objectives, as well as to predict the impact on performance of several DTP design choices and configuration parameters.

Classical approaches to performance prediction of computer systems rely on two techniques: white box and black box modeling. White box modeling, e.g., Analytical Modeling, exploits *a priori* knowledge of the internal dynamics of the target application/system in order to predict performance. Once built, white box models can typically be promptly instantiated, i.e., they require minimum (or null) training times. On the downside, they usually rely on assumptions or introduce simplifications to ensure tractability. As a result, their accuracy can be impaired in scenarios in which the employed assumptions do not hold or the introduced approximations prove to be too coarse.

Black box modeling, on the other hand, assumes no knowledge on the internal dynamics of the target system; instead, it leverages Machine Learning algorithms to infer, by means of a dedicated training phase, statistical relationships between a set of input variables, corresponding to the system's configuration and its workload's characteristics, and some key performance indicator(s). Black box models typically deliver good accuracy when working in *interpolation*, i.e., in regions of the input space that have been sufficiently explored; conversely, their accuracy is typically poor when working in *extrapolation*, i.e., in regions of the input space not explored enough during the training phase.

To cope with the limitations of the two aforementioned performance modeling paradigms, recent years have witnessed an increasing interest in gray box modeling approaches, which aim at combining the white box and

black box performance modeling methodologies.

This dissertation advances the state of the art in performance modeling of complex systems by introducing several innovative gray box performance modeling methodologies and by applying them to accurately predict the performance of different types of in-memory DTPs, employing diverse concurrency control and replication mechanisms. The resulting hybrid models take the best of the white box and black box performance modeling approaches: reduced training time, increased robustness in extrapolation, and ability to correct initial inaccuracies by incorporating new factual knowledge over time.

In particular, this dissertation make the following contributions:

- it proposes a taxonomy of existing gray box modeling methodologies to predict performance of computer systems, identifying two main branches: *Parameter Estimation*, in which black box techniques are used to infer input parameters for a white box model, and *Hybrid Ensemble*, in which the output of white box and black box models are combined to minimize predictive error;
- it introduces a novel modeling methodology, named *Divide et impera*, which allows for the joint usage of white and black modeling techniques targeting the performance prediction of distinct, but possibly inter-related, modules of the system being modeled;
- it advances the state of the art in the field of the *Hybrid Ensemble* methodology, by introducing three novel techniques, namely *Hybrid KNN*, *Hybrid Boosting* and *Probing*; in addition, it proposes new algorithmic implementations of the *Bootstrapping* technique, a *Hybrid Ensemble* approach recently proposed in literature;
- it proposes innovative analytical DTPs performance models, which target combinations of concurrency control schemes and replication policies that have never been analyzed in the literature.

The DTP employed to implement and validate the effectiveness of the proposed models and performance modeling techniques is Infinispan by JBoss/RedHat, an industrial-quality, open source distributed transactional key value store: the experimental validation is performed using standard benchmarks and using large scale Infinispan deployments (up to 140 instances), over both private and public Cloud infrastructures.

By means of extensive experimental evaluations, this dissertation shows that performance models based on the proposed hybrid techniques are able to outperform the single black and white box models they are based on, resulting in both higher accuracy and reduced training time.

Keywords: In-Memory Distributed Transactional Platforms, Performance Modeling, White Box Performance Modeling, Black Box Performance Modeling, Gray Box Performance Modeling, Analytical Modeling, Machine Learning

Resumo

A Computação na Nuvem tem vindo a mudar drasticamente a forma como as aplicações modernas são provisionadas: na Nuvem, os recursos são adquiridos elasticamente, ou seja, dependem da carga da aplicação em cada momento, permitindo portanto uma redução significativa do investimento financeiro inicial e dos custos de operação.

As Plataformas Distribuídas Transacionais (PDTs) congregam um conjunto importante de plataformas para a Nuvem: ao usarem o conceito familiar de transacção, os programadores podem delegar, para a plataforma distribuída, a responsabilidade de gerir os acessos concorrentes a secções críticas do código, e são portanto evitadas as complexidades associadas à garantia de consistência dos dados na presença de mudanças da escala da infraestrutura distribuída de forma elástica assim como falhas e fontes de assincronismo.

A modelação do desempenho de PDTs é um requisito fundamental para implementar o provisionamento automático de recursos e esquemas de ajuste automático que tiram partido das vantagens da elasticidade da Nuvem. Os modelos de desempenho, na verdade, representam um meio valioso para estimar os recursos necessários por uma aplicação para que esta cumpra os níveis objectivos de serviço, assim como para prever o impacto no desempenho de diversas escolhas de desenho em PDTs e os seus parâmetros de configuração.

As estratégias clássicas de previsão de desempenho de sistemas computacionais dependem em duas técnicas: modelação por caixa branca e por caixa preta. A modelação por caixa branca, como por exemplo Modelação Analítica, explora conhecimento adquirido a priori sobre as dinâmicas internas da aplicação/sistema cujo desempenho está a ser estimado. Uma vez construídos, os modelos de caixa branca podem ser imediatamente instanciados, ou seja, requerem tempo de treino mínimo (ou mesmo nulo). Por contraste, as estratégias por caixa branca dependem usualmente de pressupostos ou introduzem simplificações para garantir a tratabilidade do problema. Como resultado, a sua exactidão pode ser diminuída em cenários cujos pressupostos não se verifiquem ou nos quais as simplificações introduzidas sejam pouco realistas.

A modelação por caixa preta, por outro lado, não assume qualquer conhecimento sobre as dinâmicas internas do sistema; a sua estratégia passa por explorar algoritmos de Aprendizagem Automática para inferir, por via de uma fase de aprendizagem dedicada, relações estatísticas entre um conjunto de variáveis de entrada, correspondendo às configurações do sistema e às características da carga, e possíveis elementos indicadores de desempenho. Modelos de caixa preta têm tipicamente boa exactidão quando funcionam por interpolação, ou seja, em regiões do espaço de valores de entrada que tenham sido suficientemente explorados; em contraste, a sua exactidão é tipicamente fraca quando funcionam em extrapolação, ou seja, em regiões do espaço de valores de entrada que tenham sido pouco

exploradas durante a fase de treino.

Para lidar com as limitações de ambas as abordagens de previsão de desempenho acima mencionadas, temos vindo a observar nos anos mais recentes um aumento de interesse nas abordagens de caixa cinzenta, que procuram combinar as metodologias de modelação de desempenho de caixa branca e caixa preta.

Esta dissertação avança o estado da arte em modelação de desempenho de sistemas complexos ao introduzir várias metodologias de modelação de desempenho por caixa cinzenta, e ao aplicá-las para prever com exactidão o desempenho de diferentes tipos de PDTs com dados em memória, usando diversos mecanismos de controlo de concorrência e de replicação de dados. Os modelos resultantes, de natureza híbrida, exploram o melhor das abordagens de modelação de desempenho por caixa branca e caixa preta: tempo de aprendizagem reduzido, robustez adicional em extrapolação, e a habilidade de corrigir inexactidões iniciais ao incorporar novos factos de conhecimento ao longo do tempo.

Em particular, esta dissertação tem as seguintes contribuições:

- propõe uma taxonomia para as metodologias de modelação de desempenho de sistemas de computação por caixa cinzenta, identificando dois ramos principais: Estimação por Parâmetros, nos quais as técnicas de caixa preta são usadas para inferir os parâmetros de entrada para o modelo de caixa branca, e Conjunto Híbrido, no qual as saídas da caixa branca e caixa preta são combinadas para minimizar o erro de estimação;
- propõe uma metodologia de modelação nova, de nome Divide et impera, que permite o uso em conjunto de técnicas de modelação branca e preta para a estimação de desempenho de módulos, possivelmente inter-relacionados, de um sistema.
- avança o estado da arte na área da metodologia Conjunto Híbrido, ao introduzir três novas técnicas, chamadas Hybrid KNN, Hybrid Boosting e Probing; para mais, propõe novas implementações algorítmicas da técnica de Bootstrapping, que é uma abordagem de Conjunto Híbrido recentemente proposta na literatura;
- propõe modelos analíticos de desempenho inovadores para PDTs, que visam a combinação de esquemas de controlo de concorrência e políticas de replicação que nunca haviam sido analisados na literatura.

A PDT usada para implementar e validar a eficácia dos modelos propostos e o desempenho das técnicas de modelação é o Infinispan da JBoss/RedHat, um alojador distribuído e transacional de dados por chave valor, de código aberto e qualidade industrial: a validação experimental foi elaborada com aplicações típicas e usando conjuntos de máquinas com Infinispan de larga escala (até 140 instâncias), tanto em infra-estruturas de Nuvem privadas como públicas.

Por via de avaliações exaustivas experimentais, esta dissertação mostra que o desempenho de modelos, baseados nas técnicas híbridas propostas, consegue ultrapassar o desempenho individual dos modelos de caixa branca e preta em que se baseiam, resultando tanto em exactidão mais alta como em tempo de aprendizagem mais baixo.

Palavras chave: Plataformas Transacionais em Memória Distribuídas, Modelação de Desempenho, Modelação por Caixa Branca, Modelação por Caixa Preta, Modelação por Caixa Cinzenta, Modelação Analítica, Aprendizagem Automátiqua

Index

List of Figures

1 Introduction

1.1 Context

The advent of Cloud computing has drastically impacted the resource provisioning scheme at the basis of current computing platforms: in typical Cloud Infrastructure as a Service (IaaS) platforms, in fact, resources are dispensed elastically, with a seemingly unbounded amount computational power and storage available on demand, in a pay-only-for-what-you-use pricing model. This *elastic scaling* capability allows any user to provision a cluster of virtually any size within minutes and comes with the promise of enormous money saving and resource efficiency (Buyya et al., 2009).

Getting additional computational resources, however, is not as simple as upgrading to a bigger, more powerful machine with commensurate increases in CPUs, memory, and local storage; conversely, in the cloud, additional resources are typically obtained by allocating additional server instances to a task. Therefore, programmers are faced with the challenging task of developing distributed applications tailored for very dynamic, elastic, and fault-prone environments.

Distributed Transactional Platforms (DTPs) are specifically built to facilitate this task: by building upon the familiar construct of *transaction*, programmers can delegate to a distributed middleware the burden of regulating concurrent accesses to critical sections, enforcing coherence and consistency of data, while transparently tolerating failures and masking latencies.

On the other hand, unfortunately, determining the right amount of resources to allocate to a DTP in such a way to match a desired Quality of Service (QoS) is far from being a trivial task. Solutions to automate the elastic scaling of data platforms that are provided by commercial Cloud providers are typically based on heuristics and thresholds. As an example, a widely employed rule-of-thumb policy to orchestrate the resource provisioning process is based on CPU utilization: if it is higher than a user-supplied threshold, then new resources are allocated,

proportionally to the incoming workload's intensity; if it is lower than another user-defined threshold, then some resources are deallocated (Google, 2015b; Amazon, 2015a; Microsoft, 2015a).

This provisioning scheme leaves on the user's shoulders the burden of determining proper values for thresholds and amount of resources to de/allocate at need. Unfortunately, scalability trends and performance of DTP applications are influenced by several, intertwined, factors, e.g., CPU processing times, network latencies and dynamics of transactional protocols.The problem of right-sizing a DTP is further exacerbated by the huge design space that characterizes this kind of platforms. For example, transactional protocols can be implemented according to different principles, e.g., single vs multi-master, and resilience to faults can be achieved by different means, e.g., logging to disk or in-memory replication. The space of possible DTP configurations grows even further when considering the set of different parameterizations for tuning knobs that are typically exposed to programmers/system administrators. For example, for in-memory replicated platforms, the number of per-datum copies that are maintained represents a fundamental trade-off between performance and fault resiliency.

Manually identifying, in such a large search space, the platform configuration that minimizes resource usage, and hence operational cost, while meeting a target QoS is, therefore, a daunting task. Performance modeling represents the means by which the problem of provisioning and self-tuning of DTPs can be effectively tackled. Once a performance model of the target DTP application has been obtained, in fact, the end-user can simply exploit it in order to jointly determine the optimal application's parameterization and platform sizing for a given workload, or the process of tuning the application can be automatized (IBM Corp., 2004).

1.2 The evolution of Performance Modeling

Analytical Modeling (AM) and Simulation have been for decades the reference technique for performance modeling and prediction (Kleinrock, 1976; Fujimoto, 1990; Menasce & Almeida, 2001; Menasce et al., 2004; Harchol-Balter, 2013; Tay, 2013). These two techniques rely on some *a priori* knowledge of the internal dynamics of the target application/system and of the hosting platform in order to express their mapping to performance, which they express by

means of some kind of mathematical formalism, e.g., a set of equations in the case of Queueing Theory (Kleinrock, 1975) or a graph in the case of Petri Nets (Petri, 1966) or via simulation model (e.g., a discrete-event based simulator (Fujimoto, 1990)). For this reason, these two techniques are referred to as *white box*.

The most appealing feature of white box modeling solutions is that, once instantiated, they are capable of accurately predicting several Key Performance Indicators (KPI), e.g., throughput and response time, for a wide set of application's workloads, tuning parameters and characteristics of the underlying hosting infrastructure. Yet, white box modeling comes often with the downside of relying on assumptions and approximations, which are necessary to ensure the analytical tractability of the model, in the case, e.g., of Queueing Theory-based model, or to avoid an overly complex code in the case of a simulation-based model. The prediction accuracy of a white box model is, thus, hindered in scenarios that do not match such assumptions (Harchol-Balter, 2013).

Over the last years, however, the successful application of white box modeling has been progressively challenged by a multitude of factors, which jointly contributed to exacerbate the complexity of applications and systems.

A first source of increase in complexity lies in the rise of new computing paradigms, which led to a revolution in the way applications are designed, encoded and deployed, challenging existing and long-standing white box modeling methodologies. On one hand, the advent of multi-core architectures has brought concurrent programming to the forefront of software development, leading applications to be parallel by design. Not only does this reflect into the need of a proper modeling of resource sharing in architectures characterized, for example, by a massive amount of computing units and non-uniform memory accesses (Hong & Kim, 2009); it also forces to explicitly take into account, when devising a white box model, the effects of concurrent accesses of processes/threads to shared data, capturing dynamics that are specific to the abstractions exploited to regulate concurrency, e.g., locking in databases (Yu et al., 1993) or Transactional Memory (Di Sanzo et al., 2010).

Analogously, the establishment of Cloud Computing as a reference computing model has caused a revolution in the distributed computing paradigm. The elastic nature of Cloud infras-

tructures challenges the ability of white box models to predict accurately the performance of an application when deployed over a platform whose scale can range from a handful to hundreds or thousands of — possibly heterogeneous — machines. Moreover, the need for elasticity has led to the proliferation of new data models (e.g., NoSQL (Han et al., 2011)) and abstractions (e.g., Distributed Transactional Memory (Kotselidis et al., 2008; Couceiro et al., 2009)), aimed at enhancing scalability or at simplifying the development of distributed applications. Just like in the case of parallel programming, white box models architects must bear the burden of formalizing the impact that adopting a specific data model and relying on a specific abstraction has on performance.

In addition, the Cloud Computing paradigm brought about the challenge of explicitly coping with virtualized platforms. In Cloud infrastructures, in fact, applications do not interact directly with the hardware of the physical hosting machine; instead, they are stacked on top of a virtualization layer that intentionally hides low-level details on the actual underlying platform (Barham et al., 2003). While being the fundamental enabling technology at the basis of the Cloud revolution, virtualization represents at the same time a major threat to the derivation of white box applications performance models. In fact, virtualization impairs the possibility of observing the dynamics according to which the application's runtime interacts with the hardware, e.g., the network topology used by the cloud provider. This clearly impairs the ability to develop detailed white box models, hindering the adoption of white box modeling techniques

Further complexity in the design of modern applications stems from the tight dependability requirements that current services must abide by. A typical solution to achieve high availability and fault tolerance consists in replicating a service across multiple instances (Guerraoui & Schiper, 1997). Unfortunately, from a performance modeling perspective, this implies dealing with other complex dynamics, stemming from the actual implementation of the replication (e.g., full or partial replication of data and services (Charron-Bost et al., 2010)) and from the protocol according to which modifications to the state of a service at one site are consistently managed and propagated to other replicas (Charron-Bost et al., 2010).

On the light of these considerations, it appears evident that DTPs represent the archetype of the modern, complex, distributed application, as they incarnate all the functional and business

requirements discussed so far. DTPs are, in fact, intrinsically concurrent, parallel and distributed and adopt complex consistency and replication protocols. As already mentioned, moreover, they have established as reference data platforms for the Cloud, as proven by the numerous academic projects in the field (e.g., CloudTM (Romano et al., 2010) and CumuloNimbo (Jiménez-Peris et al., 2012)) and by the proliferation of commercial relational and NoSQL Cloud platforms (e.g., Google Cloud Datastore (Google, 2015a), Amazon SimpleDB (Amazon, 2015b) or Microsoft Azure SQL Database (Microsoft, 2015b)). As such, the natural deployment environment for DTPs is mainly constituted by virtualized infrastructures. All these characteristics jointly contribute to make the definition of an accurate white box performance model for DTPs an extremely challenging task.

Fortunately, the last years have also witnessed the maturing of research in the area of Machine Learning (ML) (Bishop, 2006), which resulted into the development of a wide range of freely-available high quality toolkits (e.g., Weka (Hall et al., 2009) and Apache Mahout (Owen et al., 2011)). This has led a growing number of researchers to explore the possibility of using ML techniques to build *black box* predictors of the performance of complex computer systems (Ganapathi et al., 2009; Delimitrou & Kozyrakis, 2013; Tesauro et al., 2006).

The black box approach lies on the other side of the performance modeling methodologies spectrum with respect to its white box counterpart: it relies on inferring the input/output relationships that map application's and system's characteristics to the target KPI, and on encoding such relationships via statistical models. Such models are built on the basis of a so called *training* phase, during which the target application is tested while generating different workloads and being parametrized with different configurations, with the purpose of observing the corresponding achieved performance. The most appealing property of this approach is that it is sufficient for the model architect to identify which are the inputs — a.k.a. features — of the target performance function, and the ML algorithm will take care of inferring how they map to the target KPI, without exploiting any additional knowledge about the application.

ML, however, does not represent the definitive solution to the performance modeling problem, as the lack of *a priori* information about the target application/system does come with a price. The accuracy of ML-based performance predictors, in fact, ultimately depends on the

representativeness of the input/output samples collected during the training phase. In order to exhaustively cover the whole space of possible inputs, the training phase should ideally sweep all combinations of possible workloads and system configurations. Unfortunately, the cardinality of the resulting set grows exponentially with the number of input features, making it cumbersome, or even practically impossible, to carry out an exhaustive training phase for complex systems — the so called *curse of dimensionality*. As a result, black box models typically deliver a very good accuracy when working in *interpolation*, i.e., in regions of the features' space that they have been sufficiently explored; conversely, their accuracy is typically poor when working in *extrapolation*, i.e., with inputs belonging to a portion of the features' space not sampled sufficiently during the training phase (Bishop, 2006).

As already happened for the white box modeling case, unfortunately, DTPs fall also into the category of systems for which building an accurate black box performance model is cumbersome. In fact, as previously described, DTPs are characterized by a huge design and configuration space, and transactional workloads are characterized by high heterogeneity (Q. Zhang et al., 2007; Elnikety et al., 2009). Therefore, the training phase of a black box algorithm aimed at learning the behavior of a DTP application is severely affected by the curse of dimensionality, with a commensurate effect on the predictive power of the resulting performance model.

To cope with the limitations of white box and black box modeling in predicting performance of complex applications and computer systems, in the last years researchers have started investigating possible synergies between these two paradigms. The solutions resulting from these research efforts aim at leveraging on both white box and black box techniques to create a new, superior breed of performance predictors, referred to as *gray box* models (Q. Zhang et al., 2007; Thereska & Ganger, 2008; Herodotou et al., 2011).

Unfortunately, research on gray box performance modeling is still in its infancy and none among the limited number of hybrid techniques that it has produced can be straightforwardly applied to accurately capture the performance dynamics of DTPs. To the best of our knowledge, in fact, gray box performance models for transactional platforms have either been proposed in the context of single-node environments (Mozafari et al., 2013; Di Sanzo et al., 2013) or, when targeting distributed deployments, do not capture dynamics stemming from data distribution or

concurrent, potentially conflicting accesses to shared data (Q. Zhang et al., 2007).

1.3 Thesis statement and Outline of the Contributions

The proposed retrospective has described how research in the field of performance modeling has evolved into two main branches, namely white box and black box modeling, highlighting their advantages and limitations. White box models, on the one hand, require very limited (or null) training time in order to be instantiated; on the other hand they often introduce approximations and rely on assumptions to reduce complexity and ensure tractability. Black box modeling, conversely, is oblivious about low-level details of the target application and hosting platform, but normally achieve poor accuracy in areas of the model's input space that have not been sufficiently explored.

For these reasons, the two methodologies have been regarded as antithetic, with one being preferred over the other depending on specific characteristics of the target application and of its use cases; particular, the previous section has discussed why neither of these two approaches tackles adequately the challenges related to performance modeling of modern DTPs.

To overcome their limitations, researchers have started to propose computer systems' performance prediction methodologies that draw solutions and techniques from both the white box and black box modeling paradigms.

This dissertation is framed in the context of this emerging performance modeling field and defends the thesis that it is possible to construct accurate performance prediction models of DTPs by using the white box and black box performance modeling methodologies in synergy, i.e., by employing hybrid performance predictors that aim to take the best of the two worlds: reduced training time, increased robustness in extrapolation, and ability to correct initial inaccuracies by incorporating new factual knowledge over time. These appealing properties can be achieved by exploiting the complementarity of white and black box approaches' strengths: on one hand, by integrating a white box model, a gray box performance predictor can be instantiated requiring a shorter training phase than a pure black box one; on the other hand, by incorporating some ML component, the accuracy of a gray box model can be increased as new

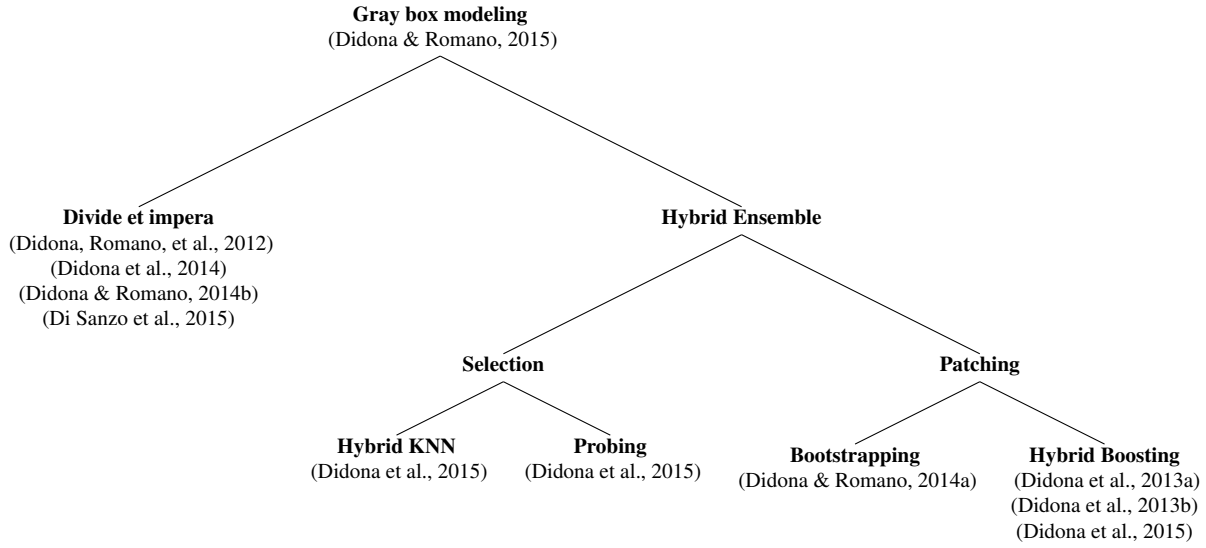


Figure 1.1: Gray box performance modeling methodologies proposed in this dissertation.

data from the operational system are collected, allowing for the correction of possible inaccuracies of the complementary white box model.

In order to support this thesis, this dissertation defines several novel gray box performance modeling methodologies, which are summarized in Figure 1.1 and briefly described below.

1. Divide et impera. This approach allows for the joint usage of white and black box modeling techniques, each capturing the performance of distinct, yet possibly inter-related, modules of the target system. The key advantage of this technique is that it allows for decomposing the problem of predicting the performance of a complex system into a set of simpler sub-models, which can exploit the modeling methodology (i.e., white box or black box) that better fits the characteristics of the considered system's module.

   In this dissertation the *Divide et impera* approach is employed in the context of DTP modeling as follows: analytical models are employed to capture the performance of transactional consistency protocols, whose algorithmic dynamics are fully specified and are thus amenable to be described via white box approaches; black box modeling is used to predict response time of operations that require distributed synchronization, which would be hard to predict accurately without detailed knowledge on the underlying physical (e.g., networking) infrastructure.

2. Hybrid Ensemble. This approach is based on the idea of combining the output of a set of white and black box models with the purpose of generating a single model with a higher predictive accuracy than any of its constituent parts. In particular, the role of the black box components is to complement the predictions provided by one or more base white box models by correcting their possible inaccuracies. Such a correction can be implemented using two key principles:

   (a) Selection, i.e., employing the performance model (either white or black) that is expected to maximize accuracy depending on the configuration/workload whose performance is being predicted.

   (b) Patching, i.e., progressively learning how to "patch" the predictions generated by the white box models. Such a goal is pursued by accumulating knowledge on the errors of such models in various regions of their parameter space.

   These two principles can be instantiated using different techniques, which correspond to different ways of combining the outputs of an ensemble of white and black box models. This dissertation investigates four such techniques; the first two are based on the *Selection* principle, whereas the last two are inspired by the *Patching* principle.:

   - Hybrid KNN, which consists in building several performance models independently and to use, depending on the incoming query, the one with the lowest expected predictive error. In particular, in order to estimate the error in predicting the performance for a new sample x, this technique evaluates the accuracy of the available models on a set of samples D_k that are as similar as possible to x.
   - Probing, which trains a black box learner only in regions of the parameter space in which a base white box model (or a set thereof) is found to achieve unsatisfactory accuracy. In order to decide in which scenarios the black box or the white box models should be used, an additional black box classifier is trained to predict which of the available models is expected to maximize accuracy in different regions of the parameter space.
   - Bootstrapping, which consists in relying on a white box model to generate a synthetic training set over which a complementary machine learner is initially trained.

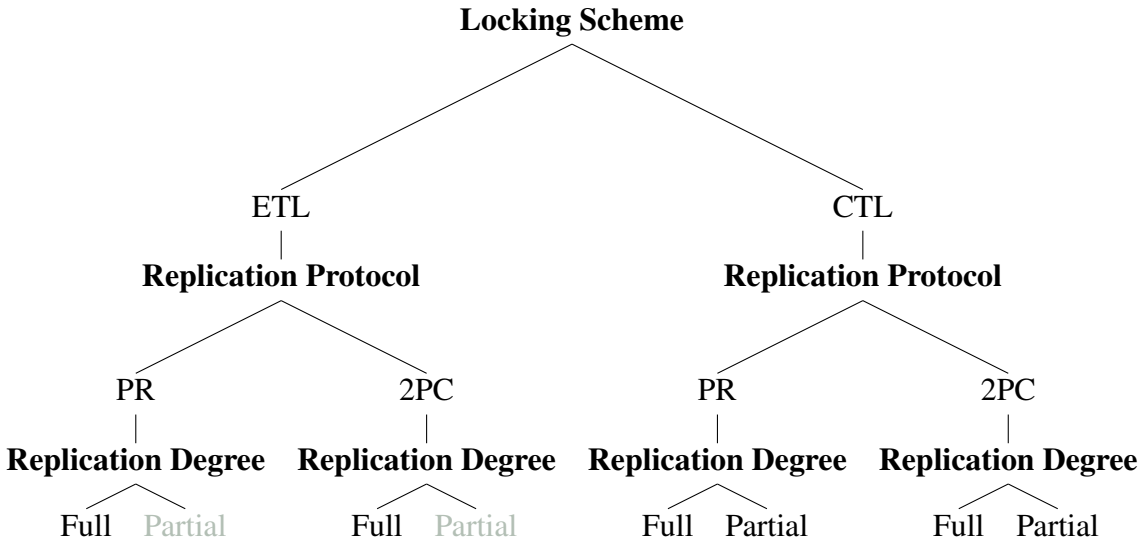


Figure 1.2: DTPs design space dimensions considered in this dissertation: replication degree, replication protocol and concurrency control scheme. Leaves of the tree marked in black correspond to design choices for which this dissertation presents a performance model; lighter gray, instead, denotes combinations that are not addressed. The motivations behind the exclusion of specific combinations are discussed in Chapter 4.

The synthetic training set is then updated over time to incorporate new samples collected from the operational system. By re-training the machine learner over the updated knowledge base, the Bootstrapping technique aims to progressively correct the performance function of the white box model.

- Hybrid Boosting, which is based on the idea of exploiting a base white box model and a chain of black box learners. The latter ones, instead of targeting the prediction of the desired performance function, are trained to learn the error function of the white box model, with the ultimate goal of compensating for it.

The proposed gray box methodologies are employed to derive performance models that address a wide region of the design space of DTP systems. In particular, the proposed gray box models capture the performance of DTP systems that use different approaches regarding three key dimensions, as also illustrated in Figure 1.2:

- Full vs partial replication: In fully replicated DTPs, the entire data set is replicated on every machine of the DTP, whereas in partially replicated systems, data items are replicated

over a (typically small) number of machines. The two approaches have dual pros and cons. Larger degrees of replication lead to higher synchronization overheads and memory consumption since data need to be updated and stored on every machine, but provide stronger failure resilience guarantees. Smaller replication degrees, conversely, require costly remote accesses to retrieve data that is not replicated locally, but have higher scalability potential thanks to their lower memory and synchronization overheads.

- Locking strategy: In order to ensure isolation, several locking strategies have been proposed in the literature. In this dissertation two variants are considered: *Encounter-time* vs *Commit-time* locking, which, as the name suggests, differ by the time in which locks are acquired by a transaction (i.e., upon the first access to a data item or at commit time). The former has the advantage of allowing a more timely detection of conflicts, but can unnecessarily constrain parallelism by increasing the lock duration. Commit time locking avoids this drawback, but, due to its more optimistic nature, can lead to worse performance in high contention scenarios.

- Single vs Multi master: The cost and complexity of inter-node synchronization in a DTP is strongly affected by the choice of the number of nodes that can execute update transactions. *Multi master* schemes allow any node in a DTP to process update transactions, and, as such, require a (costly) distributed consensus phase to manage conflicts arising from transactions executing at different nodes. In *single master* approaches, only a single node (typically called primary) is entitled to process update transactions. This hampers the scalability of the system in presence of write-intensive workloads, but streamlines the processing of update transactions by detecting and resolving conflicts locally at the primary node.

In this dissertation the gray box methodologies illustrated in Figure 1.1 are employed to derive performance models for DTPs as follows.

1. First, the *Divide et impera* methodology is employed to derive gray box models capable of predicting the performance of a DTP depending on its scale, parametrization and workload characteristics. Achieving this result has required developing a set of novel

white box models which capture the performance dynamics of the different transactional consistency protocols depicted in Figure 1.2.

The devised white box models represent another relevant contribution of this dissertation. In fact, not only they fill relevant gaps in the literature on performance modeling of DTPs by targeting transactional protocols whose models, to the best of our knowledge, have never proposed in literature. They also introduce a novel abstraction, namely the *Application Contention Factor* (ACF), which is crucial to enable the modeling of realistic applications that generate complex, non-uniform data access patterns. More in detail, the ACF abstraction allows for modeling arbitrary data access patterns (e.g., skewed data popularity) by means of an equivalent one in which each datum is accessed with equal probability. It is noteworthy to mention that the applicability of the ACF abstraction is not constrained to the case of the DTP algorithms considered in these dissertation, but, conversely, it can be employed in the broader context of (possibly non distributed) transactional platforms.

The *Divide et impera*-based DTP performance models have been validated via an extensive experimental evaluation on an industrial quality open-source DTP, namely Infinispan by Red Hat (Marchioni & Surtani, 2012). Infinispan is, at the moment of writing, the reference NoSQL data platform and clustering technology for JBoss AS, one of the most popular open-source J2EE application servers. Infinispan has also been the reference transactional cloud data store for the recent European project Cloud-TM (Romano et al., 2010). The models presented in this dissertation have been integrated in the final prototype of the Cloud-TM's platform (Didona & Romano, 2014c) and employed to automate the resource provisioning process of the Cloud-TM platform.

2. The *Hybrid ensemble* techniques were instantiated and validated considering two base predictors:

   (a) a pure white box model, already available in literature (Romano & Leonetti, 2012), of Total Order Broadcast (TOB) implementation (Cachin et al., 2011), which is a fundamental building block for a number of replicated DTPs, e.g., Distributed Transactional Memories (Couceiro et al., 2009) and Distributed Databases (Pedone

et al., 2003).

(b) the gray box DTP performance models resulting from the application of the *Divide et impera* methodology. In fact, despite not having a purely white box nature, these models do embed *a priori knowledge* about the internals of the target application and can hence be regarded as base predictors, whose accuracy and robustness can be enhanced by means of *Hybrid ensemble* techniques.

The use of two base predictors, a purely white box one and a gray box one, allows for assessing not only the effectiveness of the *Hybrid ensemble* techniques to enhance the accuracy of a white box base predictor. It also aims at showing how different gray box modeling techniques can be combined to yield a single, superior performance predictor.

Finally, this dissertation also proposes what it is, to the best of our knowledge, the first taxonomy on gray box performance modeling techniques applied to the context of computer systems and applications. In fact, proposed solutions in this field represent isolated attempts to tackle specific problems, and the literature lacks a thorough and organic discussion that identifies and relates the different base methodologies that have been proposed to combine the white box and the black box modeling paradigms.

This taxonomy is not only intended to describe better how the techniques proposed in this dissertation relate to existing solutions, and how they are framed in the literature of gray box performance modeling. It especially aims at providing a first systematic presentation of the works that embody this emerging performance modeling paradigm: in particular, it classifies existing solutions on the basis of the methodology employed to reconcile the two conventional performance modeling paradigms and also highlights their relations with similar approaches proposed in other engineering fields, namely System Identification (Ljung, 1999).

The remainder of the thesis is structured as follows: Chapter 2 provides background on DTPs and presents Infinispan; Chapter 3 introduces basic concepts, terminology and methodologies in the fields of white and black box modeling, discusses existing solutions in the field of gray box modeling and provides an overview of state-of-the-art solutions for performance modeling of DPTs; Chapter 4 presents the *Divide et impera* gray box modeling methodology

and applies it to derive different performance models for DTPs; Chapter 5 presents and evaluates the *Hybrid Ensemble* modeling methodologies, employing both the DTP and the TOB case study; finally, Chapter 6 concludes the thesis and discusses future research avenues.

2 Background on Distributed Transactional Platforms

This chapter provides an overview of Distributed Transactional Platforms (DTP), and introduces Infinispan, the distributed transactional key-value store that is going to be employed in this dissertation as main case study to validate the effectiveness of the proposed gray box performance modeling techniques.

2.1 Overview on Distributed Transactional Platforms

DTPs are data platforms deployed over a set of distributed nodes that allow data manipulation by means of the *transaction* abstraction. A transaction is a portion of code, enclosed by a *begin* and a *commit* instruction, which abides by the so called ACID properties (Bernstein, 1986).

- (A)tomicity: transactions are executed according to an all-or-nothing fashion. This means that either a transaction successfully completes, i.e., it commits, or it aborts, i.e., it is canceled and all the performed tentative modifications to the system state are discarded, just like if they have never been issued. Transaction aborts can be triggered in two situations: *i*) upon an explicit application request, e.g., in case the system is in a state that does not allow processing correctly the transaction logic; *ii*) as a result of a *conflict* between two or more transactions, i.e., the attempt to modify or access a portion of the system state in a way that is not compliant with the ACID properties.

- (C)onsistency: transactions access consistent states of the system, and any committed transaction induces a transition towards a consistent state of the system. Consistency is defined as a set of domain-specific rules and invariants that define the state of the system, and that must be always respected and satisfied. A typical rule enforcing consistency comes from the banking world: if data items represent bank accounts, then, in order to withdraw money from

one of them, the deposited amount of money has to be greater or equal than the one to be withdrawn.

- (I)solation: this property defines when and how modifications to the state performed by a transaction become visible to other ones. For example, Serializability (Berenson et al., 1995) enforces that the execution of transactions, despite being concurrent, results in a system state that could have been obtained by executing the transactions in a serial order, i.e., one after the other.

- (D)urability: once a transaction has been committed, its effects on the system state are permanent. This entails that modifications to the state are made persistent and resilient to crashes and failures.

2.1.1 Consistency, Atomicity and isolation in DTPs

Consistency, atomicity and isolation are enforced in DTP by means of a *concurrency control scheme* and a *replication protocol*.

The concurrency control scheme is responsible for ensuring that, despite their parallel activation, transactions appear as if they were executed in isolation and atomically, thereby allowing only safe and consistent transitions between system states. The isolation level enforced by the concurrency control scheme directly maps to the ease of programming parallel/distributed applications. In fact, the stronger is the degree of isolation that is guaranteed to transactions, the more the behavior of the system is similar to a single-threaded, serial execution, which is the most natural way of reasoning about the flow of programs' execution.

On the other hand, in general, the stronger are the isolation guarantees, the higher is the overhead incurred by the running application. Two are the root causes of such overhead: *i*) the stronger is the isolation level, the lower is the number of transactional *histories*, i.e., interleavings of transactional operations, that are acceptable by the system. This, depending on the implementation, yields a higher number of aborts or a reduced degree of concurrency, which ultimately results in lower throughput; *ii*) the amount of meta-data to be collected and processed

on a per-transaction basis grows as a function of the strength of the isolation level. For example, enforcing Serializability requires to keep track not only of data items that are written by transactions, but also that are just read; this is not required, for instance, to enforce Snapshot Isolation (Berenson et al., 1995).

Much research effort has been spent to mitigate these overhead, especially motivated by the fact that their impact grows with the scale a the DTP. The solutions proposed over the years have mainly pursued two, orthogonal, research lines, namely *i*) increasing the available concurrency while keeping a strong isolation level, e.g., Serializability (Faleiro et al., 2014; Bailis et al., 2014; Diegues & Romano, 2015) and *ii*) identifying weaker isolation levels that strike a good balance between imposed overhead and ease of programming (Sovran et al., 2011; Peluso et al., 2012; Li et al., 2012).

The replication protocol determines how updates to the system state attempted on one node are propagated to other nodes in the platform. Replication protocols can coarsely be grouped according to two schemes, namely the multi-master and the single-master, also referred to as Passive Replication (PR). According to the multi-master scheme, each node in the system can serve both read-only and update transactions; this requires taking into account the need to deal with conflicts among transactions originated at different nodes.

The most common multi-master replication protocol is Two-Phase Commit (2PC) (Bernstein, 1986). As the name suggests, 2PC is a protocol that evolves in two rounds. In the first, namely the prepare phase, the node which wants to commit a transaction, namely the coordinator, sends the set of the data accessed by the transaction to a cohort of nodes for validation. The amount of information sent to the cohort depends on the enforced isolation level and by the adopted concurrency control scheme. The cohort of nodes send back a vote, depending on whether they have been able to successfully validate the transaction, i.e., on whether the execution of the transaction is compliant to the ACID properties on their side. In the second round, namely the commit phase, if the coordinator receives all positive votes, than it sends a commit message, and the transaction's outcome is persisted on each node; otherwise it sends a rollback message, and the transaction is discarded. The cohort coincides

with all the nodes involved in the transaction, i.e., all the nodes which replicate at least one datum accessed by the transaction.

A drawback of the 2PC protocol is that it is prone to distributed deadlocks, i.e., situations in which two (or more) transactions require conflicting sets of locks on multiple nodes in different orders, mutually blocking their progress. This happen because the prepare messages corresponding to conflicting transactions can be delivered in different orders on different replicas. Distributed deadlocks are a major threaten for the scalability of DTPs (Gray et al., 1996), and implementing mechanisms for their detection and resolution introduces additional overheads (Singhal, 1989). A possible solution to the issue of distributed deadlocks in DTPs is to exploit a Total Order primitive to disseminate the prepare messages, which ensures them to be delivered in the same order on all the recipients (Ruivo et al., 2011). By leveraging such a primitive, nodes in the platform may commit transactions according to a common order, thus ruling out the possibility of incurring deadlocks. On the downside, Total Order primitives are typically expensive, as they require multiple synchronization rounds among the nodes in the system to reach consensus on the transactions delivery order.

Also the single-master scheme represents a solution to the distributed deadlocks problem. In such a scheme, in fact, only a node, namely the *primary*, is entitled to serve update transactions: all other nodes, the backups, can only process read-only transactions (Budhiraja et al., 1993). This implies that, once the validation of a transaction has been performed on the primary, then it is safe to replicate its outcome also on the backups. As a result, the commit/abort decision about a transaction can be performed locally on the master, without the need of expensive distributed validation protocols. The price to pay for distributed deadlock freedom, in this case, is that the primary node may become the bottleneck for the whole system, in presence of write-intensive workloads.

The scope of the performance modeling techniques proposed in this dissertation is general, i.e., they are intended to be applicable to any variant of DTPs, and to other kind of applications and systems as well. As introduced in Section 2.2 and detailed in Chapter 4, however, the novel analytical models provided in this thesis are specifically oriented towards DTPs that pursue

lower overheads, and hence higher scalability, by means of lower isolation level; on the other hand, the proposed models target both a single-master and a multi-master replication protocol, so as to showcase their wide applicability.

2.1.2 Durability in DTPs

The conventional approach to achieve durability in DTPs is based on persisting the system state on disk, by registering on a log modifications to the system state performed by committed transactions (Bernstein, 1986). The rationale behind this choice is that stable storage was orders of magnitude cheaper than main memory; synchronous writes to persistent storage, however, poses a significant overhead to transactions execution, as the operation of writing to disk is on the critical path of transactions processing (Harizopoulos et al., 2008).

The landscape has drastically changed over the last years, as the price of main memory has exponentially decreased, while RAM banks' capacity has increased (Plattner & Zeier, 2011). In addition, the use of high performance networking infrastructures like Infiniband (Pfister, 2001) or the new breed of Ethernet (IEEE, 2014) allows for drastically reducing the communication latency between remote hosts. These factors have jointly contributed to the shift towards the in-memory data replication paradigm, according to which all data is maintained in main memory (Marchioni & Surtani, 2012; Kallman et al., 2008). In this case, the *replication degree*, noted r, determines on how many different nodes every item in the platform is replicated and, consequently, the degree of resilience to failures: a replicated DTP can tolerate up to $r-1$ concurrent failures without suffering any data loss.

This design choice allows for removing logging to stable storage from the critical path of execution of transactions, as well as for adopting optimized strategies (e.g., asynchronous writes (Perez-Sorrosal et al., 2011), batching (Baker et al., 2011), data de-duplication (Zhu et al., 2008)) aimed at minimizing the costs associated with the usage of cloud storage systems (typically offered as additional pay-per-use service (Amazon, 2013) by IaaS infrastructures).

For these reasons, in-memory DTPs are particularly attractive and have gained, in the last decade, a lot of momentum (Plattner & Zeier, 2011): the performance models developed in this

dissertation specifically target this emerging class of DTPs, although they could be extended to cope also with persistent storage logging dynamics.

2.1.3 Data models in DTPs

The importance and widespread adoption of DTPs also reflect into the diversity of proposed designs and implementations stemming from the supported data models, which varies depending on the target use case.

- Distributed Database Management Systems (DDBMS). They represent the evolution of classic centralized databases. This kind of platform has been designed to improve availability, reliability and performance of their single-node counter-part. DDBMS embrace the classic relational model, and provides support for complex SQL queries. In order to support high-level operators, like the join, DDBMS typically rely on data sharding, such that a single query can be served by a single node in the platform (Curino et al., 2010; Taft et al., 2014).

- NoSQL data stores. This new breed of data platforms has been built from the ground up specifically to meet the elasticity and availability requirements of the Cloud. For this reason, they typically embrace lower isolation levels (Wada et al., 2011) and expose schema-less, non-relational data models, e.g., key-value (Marchioni & Surtani, 2012; Das et al., 2010; Project Voldemort, 2015; Oracle, 2011), graph (Martínez-Bazan et al., 2007; Neo4j, 2015; Objectivity, 2015), or document-oriented ones (mongoDB inc., 2015; Apache Software Foundation, 2015). Because of the exposed data model, unlike the relational case, these platforms typically do not natively provide support for complex queries. As such, although optimizations can be implemented (Paiva et al., 2014), NoSQL data platforms typically do not rely on data sharding and manage data placement by means of consistent hash functions (Karger et al., 1997). Additional details on hash functions will be provided when introducing the Infinispan data platform.

- Distributed Transactional Memories (DTM). In the Transactional Memory (TM) paradigm, the abstraction of transaction becomes an integrating part of the programming language and acts as a replacement of the typical lock-based scheme for accessing critical sections (Herlihy

& Moss, 1993). DTMs represent the distributed evolution of shared-memory TMs, and allow every object or memory location of the system state to be transactionally accessed and modified (Kotselidis et al., 2008; Couceiro et al., 2009).

2.2 The Infinispan case study

Infinispan is a distributed, in-memory, transactional key-value store, written in Java and developed by JBoss/Red Hat. At the moment of writing, Infinispan represents the reference NoSQL data platform and clustering technology for JBoss AS, one of the most popular open-source J2EE application servers. It has also been the reference cloud data store in the context of the European project Cloud-TM (Romano et al., 2010), which aimed at developing a transactional data platform specifically conceived to meet the requirements of Cloud applications.

An application deployed over Infinispan interacts with the data items by means of *get* and *put* operations. A get is a read operation, i.e., it retrieves the value corresponding to a given key; a put is a write operation, i.e., it modifies the value corresponding to a given key, optionally returning the previous value. Note that a put operation can be any operation that modifies the state of the application, i.e., an insertion of a new data item, as well as the modification or the deletion of an existing one. Transactions are enclosed between a *begin* and a *commit* statements, and can include an arbitrary number of put and get operations.

Like many other NoSQL platforms (Orientechnologies, 2014; Aerospike, 2014; eXistdb, 2014; Kobrix Software, 2014; Redis, 2014), Infinispan opts for partially sacrificing consistency in order to enhance performance. In particular, it does not ensure serializability (Bernstein, 1986), but only guarantees variants of the Repeatable Read ANSI/ISO isolation level (Berenson et al., 1995). Specifically, upon reading a datum, a transaction caches the obtained corresponding value, and returns it for any subsequent read of the same datum; if the datum is updated by the transaction itself, its value is cached as well, and the most up-to-date cached value is returned upon read.

Isolation is enforced by means of a non-serializable variant of the multi-version concurrency control algorithm, which never blocks or aborts any transaction upon a read operation.

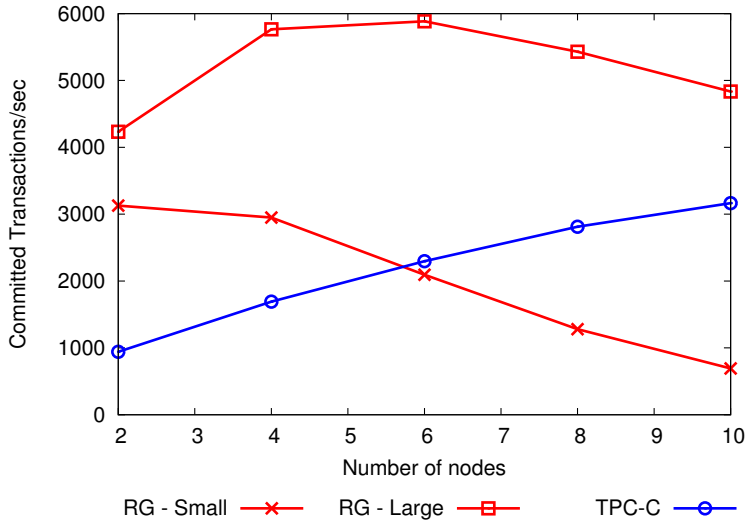


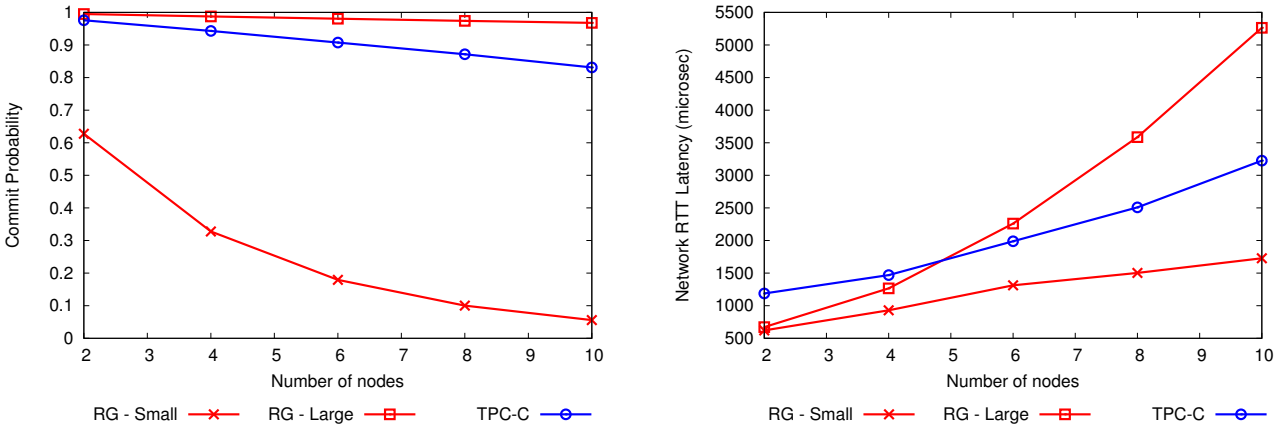
Figure 2.1: Performance of different workloads in Infinispan.

This allows read-only transactions to commit locally, without requiring any inter-node synchronization. Only write-write conflicts are detected by means of locking: this prevents concurrent updates of existing data items or insertions of new ones by different transactions that would result into breaking isolation. Deadlocks are detected using a simple, user-tunable, time-out based approach.

2.2.1 Complexity of the Infinispan Case Study

Due to its complex internal dynamics, predicting the performance of applications deployed over Infinispan as a function of the workload, of the various replication and concurrency control protocols that it supports, as well as of its internal parameters (e.g, the replication degree), is a very challenging task. This complexity is clearly highlighted by Figure 2.1, which reports on the y axis throughput values (expressed as committed transactions per second) obtained by running two transactional benchmarks, namely Radargun[1] and TPC-C[2], over a platform interconnected by a Gigabit Ethernet network, while varying the size of the platform from 2 to 10 nodes (on the x axis). These results have been obtained using a 2PC replication protocol, and an encounter time locking scheme in full replication, which will be described in Section 4.2.1.1.

[1] https://github.com/radargun/radargun/wiki
[2] http://www.tpc.org/tpcc



(a) Transcription commit probability

(b) Distributed commit network latency

Figure 2.2: Scalability analysis of different workloads in Infinispan (2PC-ETL).

As shown in Figure 2.1, the scalability trends of the three considered workloads are quite heterogeneous. The considered TPC-C workload scales almost linearly, whereas the two Radargun workloads clearly demonstrate how the effects of high contention on logical and physical resources can lead to non-linear scalability trends. The workload RG-large represents a case in which transactions insist on data items selected uniformly at random from a large dataset (100K objects): as shown in Figure 2.2a, transactions of this workload incur very little abort probability; on the other hand, network latencies sharply increase as the delivered throughput and the size of the underlying platform increase, as depicted in Figure 2.2b, up to the point that adding more nodes to the platform is actually detrimental for performance. Finally, the workload RG-small depicts a case in which transactions insist on a small data set (1K objects) and it is representative of a completely non-scalable workload. Unlike the RG-Large workload, however, looking at Figures 2.2a and Figure 2.2b, it is evident that in this case scalability is not hindered by congestion on the network; rather, the lack of scalability is caused by the excessive level of data contention, which impairs transactions progress.

These dynamics are caused by the intertwined effect of workload characteristics, contention on data and on physical resources: accurately capturing such complex dynamics via white box models appears to be a daunting task. On the other hand, the vast design space of the platform, e.g., scale and replication degree, and the several parameters characterizing a workload, e.g., percentage of update vs read-only transactions and conflict likelihood, make the application of

pure black box solutions easy prey of the curse of dimensionality. Chapters 4 and 5 will describe in detail how the proposed gray box modeling techniques can be applied to build an accurate yet fast-to-instantiate performance model for such complex and high-dimensional performance functions characterizing DTP, and more specifically Infinispan, applications.

3 Related Work

This chapter serves a twofold purpose. In Section 3.1 it provides an overview of the main performance modeling methodologies that have been proposed in literature, and discusses the innovative contributions in the field of gray box performance modeling that are proposed in this dissertation. Section 3.2, instead, presents the state of the art on performance modeling and self-tuning solutions for DTPs, and highlights how the proposed analytical models overcome the main limitations that affect existing solutions.

3.1 Performance Modeling Methodologies

As described in Chapter 1, performance modeling has evolved into two main branches, namely white box and black box modeling: background on these two modeling methodologies is provided, respectively, in Section 3.1.1 and Section 3.1.2. Section 3.1.3, instead, discusses existing techniques and solutions that combine the two aforementioned performance modeling paradigms into gray box models; specifically, the section serves the twofold purpose of *i*) organizing existing and proposed gray box performance modeling methodologies into a taxonomy and *ii*) describing how the proposed techniques relates to and advance the state of the art in the field of gray box performance modeling.

3.1.1 White Box Performance Modeling

White box modeling refers to the modeling methodology that leverages specific and detailed knowledge about the internals of a system or application to model its behavior and, ultimately, predict its performance. Many different modeling techniques comply with such definition, including Queueing Theory (Kleinrock, 1975), Petri Nets (Petri, 1966) and Simulation (Fujimoto,

1990). This section, however, will only cover introductory aspects about Queueing Theory, as it is at the basis of the analytical performance models proposed in Chapter 4.

In Queueing Theory, a system is modeled as a set of waiting lines, or queues. To each queue corresponds a server, which is in charge of processing requests coming from clients. Requests join the queue at a rate λ and are enqueued till they can be processed. A queue is described by a set of parameters, which are typically expressed via the Kendall's notation $A/B/C/K/N/D$[1].

- A is the distribution of the requests' arrival rate.
- B is the distribution characterizing the service demand of a request, i.e., the time it takes to be served without accounting for the waiting time in the queue. Values for A and B are typically D (deterministic), G (generic), M (markovian, i.e., following an exponential distribution).
- C is the number of servers that process elements in a queue.
- K is the capacity of the queue, i.e., the total number of requests, both in service and in the waiting line, that the queue can accommodate.
- N is the number of clients that generates requests towards the queue. If N is a finite quantity, then the queueing system is called *closed*; a closed system is generally characterized also by a *think time*, i.e., the time that a client waits after the completion of a request before issuing a new one. If N is infinity, instead, then the queue serves an unbounded numbers of clients and the system is called *open*. The requests' arrival rate in an open system does not depends on the number of request currently in the system. An open system is said to be *stable* if the requests' arrival rate does not exceed the system capacity: this prevents an unbounded number of jobs to queue up. According to this definition, closed systems are stable by nature, since the population of users is fixed and a user cannot issue a new request towards the system if the previous has not been served yet.
- D is the policy adopted to serve request. Values for this parameter include First/Last Come First Serve, in which requests are processed depending on the order they join the queue, and

[1] If omitted, C and K are equal to ∞; similarly, the default value for D is First Come First Served

Processor Sharing, in which the processing power of the servers is concurrently shared by the requests.

Once this set of parameters is specified, a model based on Queueing Theory can be solved to obtain several performance indicators, like the average number of requests in the system, maximum achievable throughput (i.e., number of jobs completed per unit of time) and response time of a request (i.e., the time it spends in the queue plus the time needed to be actually processed by the server).

The model resolution process does not always consist in applying a closed-form equation: in many case, Queueing Theory models are solved iteratively or recursively, providing as input parameters for iteration k the output obtained at iteration $k-1$. The Mean Value Analysis (MVA) algorithm (Reiser & Lavenberg, 1980) to solve model for closed systems is probably the most famous example of an iterative resolution technique. It exploits the arrival theorem, which states that when one user in an N-users closed system arrives at a queue, she observes the rest of the system to be in the equilibrium state for a system with $N-1$ users: therefore, MVA obtains solves the performance model for N users by iteratively solving the same model for $n = 1 \ldots N-1$ users first.

The literature on DTPs performance modeling is also rich in solutions that rely on iterative/recursive schemes to obtain the target performance indicator, typically to break the interdependency that arises between the transactions' response time function and the lock contention probability (Nicola & Jarke, 2000). Also the analytical models presented in the next chapter represent instances of models solved by means of iterative scheme, precisely to cope with the aforementioned issue.

Finally, one of the most important results relevant to Queueing Theory, which will also be exploited in the proposed analytical models, is Little's law (Little, 1961). It states that the long-term average number of customers in a stable system is equal to the long-term average effective arrival rate multiplied by the average time a customer spends in the system. This is one of the most exploited results in Queueing Theory-based analytical models, including the models introduced in Section 4, because of its universal applicability (Little, 2011). Little's law, in fact, applies independently from the arrival process distribution, the service distribution

and the serving policy.

3.1.2 Black Box Performance Modeling

Unlike its white box counterpart, which entails different ramifications, in the area of performance modeling of computer systems black box approaches have largely borrowed results from the Machine Learning (ML) domain. ML is the field of artificial intelligence that deals with construction and study of systems that can learn from data (Bishop, 2006): this section introduces the fields of ML that are more closely related to the gray box models that are object of this dissertation, and that are among the most widely applied in related works on performance modeling and self-tuning of computer systems, namely, Supervised, Unsupervised and Reinforcement Learning.

3.1.2.1 Supervised Learning

Supervised Learning is the task aimed at learning the relation between a set of input parameters, called input features, and a set of outputs, called target features. More formally, a supervised machine learner tries to infer a function (also called model) $\phi : X \rightarrow Y$ basing on the observed output corresponding to a set $\tilde{X} \subset X$, called *training set*. Such a function can, then, be exploited to predict the output value y corresponding to values of the input parameters that are not present in the training set. If the codomain of the ϕ function is continuous, then the machine learner is defined *regressor*. If it is discrete, then the learner is defined *classifier*; in this case, the values that the output can assume are called *classes*.

Many supervised ML tools have been proposed in literature; the remainder of this Section briefly overviews the ones that are exploited the most for performance modeling purposes.

- **Decision trees (DT)**(Quinlan, 1986) build their inner model according to a tree-structured graph. Each interior node of the tree corresponds to one of the input variables; edges from a parent to a child are labeled with a predicate about the value of the input feature relevant to the parent node. In the case of DT classifiers, each leaf represents a value of the target feature; in the case of regressors, it is a function of the input values.

- **Support vector machines (SVM)**(Cortes & Vapnik, 1995) are classifiers which map the points of the training set to a multidimensional space W such that elements in the same class occupy a specific portion of that space and are as far away as possible from elements of other classes. The dimensionality of W, in general, is higher than the one, X, of the samples in the training set; the function responsible for the mapping of elements from X to W is called *kernel*. Although initially introduced in the context of classification, also SVM has been extended to cope with regression problems (Bishop, 2006).

- **Artificial Neural Networks (ANN)** (Haykin, 1998) are machine learners whose inner design resembles the neural structure of the brain, which learns through experience thanks to the interconnection of billions of simple neurons. Similarly, an ANN is based on artificial neurons. In its simplest form, a neuron is a classifier that performs a weighted sum of the input parameters and maps it onto a binary set by means of a sigmoid function. The weights used by the neurons are adjusted during an initial learning phase over the training set. An ANN is structured in layers. The input layer, which does not execute any computation and just replicates the input in such a way that neurons in the next layer are provided with the whole set of features; the hidden layers, which are composed by neurons and perform weighted sums and classification; the output layer, which combines the output of the neurons into the final result (which, again, can either be a class or a real number).

A particular sub-field of Supervised Learning is Instance-based Learning. This technique is based on computing the output relevant to a previously unseen input x by analyzing the output of the training samples that, given a *similarity* function, are the most similar to x. In this way, the machine learner is spared from the burden of performing explicit generalization of the input/output relations by means of inferring a mapping function. A typical instance of this kind of learning is the **K-Nearest-Neighbor** algorithm (Cover & Hart, 1967), in which the output corresponding to x is computed, in the regression case, as weighted sum of the K samples in the training set which are most similar to x and, in the classification case, by means of voting.

3.1.2.2 Unsupervised Learning

Unsupervised Learning is the task aimed at finding patterns in the data above and beyond what would be considered pure unstructured noise (Ghahramani, 2004). An Unsupervised Learning algorithm, in fact, is not provided with an output corresponding to the value of a function evaluated on the samples in the training set: all it can do, therefore, is extracting hidden structures according to which samples are organized. The most common application of unsupervised learning in performance modeling works is **Cluster analysis** (Sander et al., 1998). The aim of Cluster analysis is to group objects into disjoint sets, such that, given a *similarity* function, the most similar elements belong to the same set.

3.1.2.3 Reinforcement Learning

Reinforcement Learning is the branch of ML that investigates the issue of how an *agent* should perform actions in an environment in order to maximize a cumulative *reward* (Bishop, 2006). One of the most important issues in Reinforcement Learning techniques is the trade-off between exploration and exploitation. In absence of an explicit model capable of determining *a priori* the optimality of an action over another one, the agent must explore the environment in order to gather feedback on the rewards of the action. A reinforced machine learner tries to identify the minimum number of exploratory steps that maximizes the long-term reward. On the other hand, in order to reduce the number of sub-optimal choices when performing actions, a Reinforcement Learning algorithm also exploits knowledge about actions that are known to be good, favoring them over exploration of actions whose outcome is more uncertain (but potentially better than the optimal known so far).

In Reinforcement Learning algorithms applied to the optimization of an application, the agent is represented by a controller in charge of determining the optimal configuration for the application, the environment is the set of tunable parameters and external factors (e.g., the workload) and the reward generally identifies with performance. Two of the most widely applied Reinforcement Learning algorithm in performance modeling and optimization tasks are probably Q-Learning and UCB.

- **UCB (Auer et al., 2002)** is an algorithm which solves the so called multi-armed bandit problem (Robbins, 1985). In this problem, a gambling agent is faced with a slot machine (the bandit) with k arms, each associated with an unknown reward distribution. The gambler iteratively plays the arms and observes the associated reward, adapting his strategy in order to maximize the average reward. An optimization problem consisting in finding the optimal value out of k possible alternatives for a parameter can be cast to the multi-armed bandit problem and solved with UCB.

- **Q-learning (Watkins & Dayan, 1992)** is a model-free Reinforcement Learning technique aimed at learning a function which maps ⟨action,state⟩ pairs into values, such that the value corresponding to an ⟨a,s⟩ pair is the expected utility of performing action a in the state s and following a fixed policy thereafter. Two important parameters for a Q-learning based agent are the *learning rate*, i.e., the weight given to most recent explorations, and the *discount factor*, which discriminates between a more greedy or a long term reward striving exploration of actions. Despite the transient behavior of the agent is highly affected by these parameters and the policy used to drive exploration, Q-learning is characterized by a strong convergence property towards a near-optimal learnt function.

3.1.2.4 Ensemble Learning

Ensemble learning is a specific branch of ML that investigates how to combine multiple learning algorithms to build a predictor with better accuracy than any of its components alone. The literature on ensemble learning is vast, and several solutions have been proposed, corresponding to different specializations of the general recipe of combining models (Dietterich, 2000; Caruana et al., 2004). Among these solutions, the most important and most widely employed ones are based on three following principles.

- **Bagging** (Breiman, 1996). Given a training set D, Bagging generates K sub-training set obtained by sampling D uniformly at random with replacement; then, K black box models are trained over the K sub-training sets (typically, the K models are different instances of the same learning algorithm, e.g., DT). The outputs of the K models obtained in this way are combined according to a majority voting, in case of classification, or by means of averaging, in case of

regression problems. By obtaining a unique predictive model by means of sampling on the original data distribution in D, Bagging is especially effective into reducing over-fitting in the prediction process.

- **Boosting** (Schapire, 1999). Given a training set D, Boosting serially trains a chain of ML algorithms in such a way that the $i-$th such algorithm tries to compensate for the prediction errors of the $i-1-$th one. In the case of classification, this is done by training the $i-$th algorithm on a sampling of D which gives more priority to elements in D that had been mis-classified in previous iterations (Schapire, 1999). In the case of regression, instead, this is accomplished by training the $i-$ to learn not the target feature itself, but the error distribution of the previous learner on the target feature, so that such error can be properly compensated (J. Friedman et al., 2000).

- **Stacking** (Wolpert, 1992). Given a training set D, Stacking independently trains K ML algorithm over it and, then, combines their output by means of a *gating* function. The definition of the Stacking technique is rather general, and particular implementations of it can coincide with other established ensemble learning techniques. For example, it is clear that Bagging can be seen as a Stacking technique, with the several learners being trained over sub-sets of the original training set and the gating function being an arithmetic average or a voting scheme. The training phase and gating function, however, can be arbitrarily more complex than in the Bagging case. In the proposed *Hybrid Ensemble* techniques that embrace the *Selection* principle (introduced in Section 1.3), for example, the gating function is implemented via a classifier that acts as a multiplexer, providing as output of the stacked model only the prediction of the inner model that is supposed to be the best for a particular query.

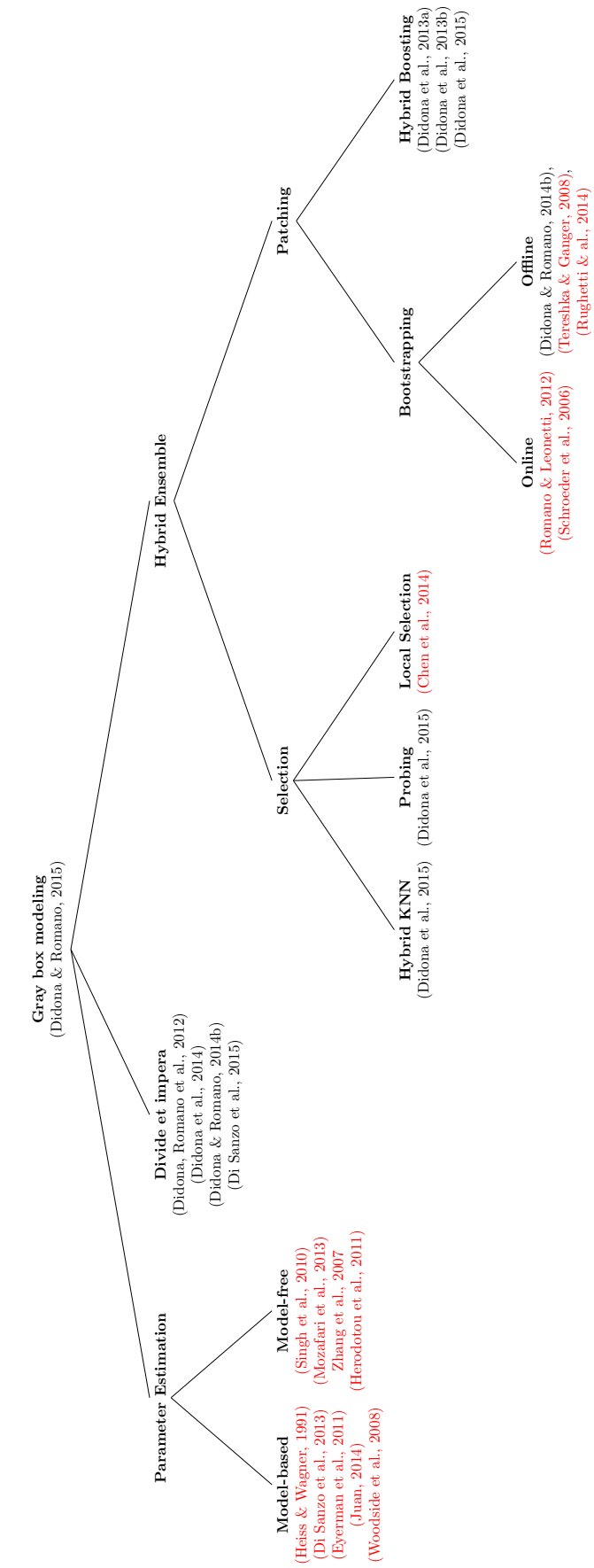


Figure 3.1: Taxonomy of gray box performance modeling methodologies. The solutions that have been proposed in the context of this dissertation are in black; the ones that can be already found in the literature are in red.

3.1.3 Gray Box Performance Modeling

As already discussed in Section 1, the white box and black box performance modeling approaches exhibit complementary strengths and limitations. This has lead to the definition of a research line aimed at devising effective ways to exploit the two techniques in synergy to model the performance of complex computer systems. This research field is, however, still in its infancy: the number of proposed gray box performance modeling techniques is indeed very small, with respect to the abundant literature on white and black box modeling.

This section serves two purposes. On one hand it reviews state-of-the-art gray box performance modeling techniques and organizes them according to a taxonomy, which classifies existing solutions depending on the methodology employed to combine white box and black box modeling. On the other hand, this section also compares the gray box proposed in this dissertation with existing ones.

The proposed taxonomy, depicted in Figure 3.1, extends the gray box performance modeling techniques classification already introduced in Section 1.3: existing solutions are marked in red; the techniques proposed in this dissertation, instead, are in black. To the best of our knowledge, this taxonomy represents the first attempt to classify existing solutions on the basis of the methodology they employ to reconcile the two performance modeling paradigms: it aims at providing an initial, systematic study of the gray box performance modeling approaches that have been proposed in literature. The taxonomy identifies three main classes of gray box modeling techniques applied to the problem of computer systems' performance prediction: *Divide et impera*, *Parameter Estimation* and *Hybrid Ensemble*.

The remainder of this section provides, for each of these techniques, a discussion that develops on three levels: *i*) description of the technique, with the aim of identifying the key principles that are at the basis of different gray box methodologies; *ii*) overview of the main solutions that adopt the technique, so as to show how a specific methodology has specialized in different implementations; and *iii*) comparison of state-of-the-art solutions with the new techniques proposed in this dissertation, with the aim of discussing relations and highlighting how this thesis advances the state of the art in the field of gray box performance modeling.

3.1.3.1 Divide et Impera

The *Divide et impera* approach consists in the joint usage of white and black modeling techniques that target the performance prediction of distinct, but possibly inter-related, modules of the system being modeled. In particular, the *Divide et impera* approach decomposes the performance prediction problem into sub-tasks, and tackles each of them by means of the modeling methodology (either white or black) that is more appropriate.

The different models, although built separately, may exhibit input dependencies: for example, a model may need, as input, a parameter that is not measurable in the current configuration/workload, and that, therefore, needs to be estimated by another model. For this reason, the *Divide et impera* also requires the definition of a resolution scheme that reconciles the output of the various sub-models in such a way to produce the performance prediction for the system as a whole.

In this thesis, the *Divide et impera* approach is applied to the problem of predicting performance of DTPs: white box modeling is applied to capture the effect of replication and concurrent accesses to data; black box modeling, instead, is used to predict the response time of network-bound operations. This modeling choice is aimed at circumventing the previously discussed limitations posed by virtualization in Cloud environments. Specifically, the lack of details about the networking layer of the hosting infrastructure impairs the adoption of the pure white box modeling paradigm to predict the cost of transactional operations that require inter-node synchronization.

To the best of our knowledge, the *Divide et impera* methodology has never been proposed before in literature: therefore, additional details about this technique are going to be provided in Chapter 4, which will describe extensively this gray box performance modeling methodology and how it has been implemented to predict the performance of DTPs. The discussion of the relations with other gray box performance modeling techniques, instead, is deferred to the following two sections and is provided after a review of the other existing methodologies and of the solutions that implement them.

3.1.3.2 Parameter Estimation

In the proposed taxonomy, the *Parameter Estimation* technique refers to solutions that rely on black box techniques to infer or perform pre-processing of a set of parameters or inputs of a white box predictor. Techniques belonging to this category can be further classified into two sub-classes, namely *Model-based* and *Model-free*, which are going to be described in the following.

Model-based. Solutions belonging to this class rely on a white box model to provide the functional form of the target KPI function (e.g., polynomial) and aim at estimating the model's coefficient and tunable parameters by fitting (e.g., via regression) the model's output to experimental data. To the best of our knowledge, model-based *Parameter Estimation* techniques have originally been proposed in the field of System Identification (SI) (Ljung, 1999). SI techniques aim at inferring, from available measurements, the *transfer function* of a system, i.e., the function that describes the input-output relationship of such system (Ogata, 2001).

SI solutions are usually black box, as they typically obtain the whole transfer function (and not only its parameters) entirely from data, by means of regression or the use of ANN (Ljung, 1999). It is possible, however, to build a parametric model of the target system and to apply *Parameter Estimation* techniques to infer only values for the free parameters of the model, rather than the whole transfer function (Astrom & Eykhoff, 1971; Kristensen et al., 2004).

Gray box SI techniques have been mainly applied to build mathematical model of physical systems, e.g., aircrafts or engines, which are typically based on sets of differential equations, rather than on the aforementioned solutions for performance modeling of computer applications (e.g., Queueing Theory). This model-based *Parameter Estimation* branch of the SI field is mentioned in this dissertation to show how synergies between white box and black box modeling have been object of research also out of the context of performance prediction of computer systems: the focus of this section, however, is on reviewing existing (and proposed) classes of gray box modeling techniques applied to performance prediction of computer systems and applications.

The first application of this technique in this context is — to the best of our knowledge — the "parabola approximation", proposed to capture the relation between the level of parallelism in a database and its performance (Heiss & Wagner, 1991). In this model, the throughput achieved by a database is expressed as a parabolic function of the number of active threads, i.e., it increases up to a maximum and then decreases because of thrashing phenomenons due to contention on hardware and data items. The functional form of the performance model is, therefore, a second level polynomial, and its coefficients are estimated by sampling the target performance function at some selected configurations.

Always in the context of transactional systems, Di Sanzo et al. developed an analytical model to predict performance of Transactional Memory applications (Di Sanzo et al., 2013); such model captures the data conflict likelihood by means of an exponential function, whose parameters are fitted via regression starting from runtime measurements.

The model-based *Parameter Estimation* technique has also been applied to model the performance of processors (Eyerman et al., 2011), to tune operational voltage and frequency in multi-core processors (Juan, 2014) and to represent birth-death stochastic processes (Kleinrock, 1975), which are the foundations of many analytical models for computer systems (Kleinrock, 1976).

Woodside et al. propose the use of Kalman Filters (Welch & Bishop, 1995) instead of regression to estimate the parameters of performance models from runtime measurements, and show how to obtain input parameters for open and closed queueing models (Woodside et al., 2008).

Model-free. Unlike the previous case, techniques belonging to this class do not exploit the target white box model to perform parameter estimation. On the other hand, they rely on black box techniques to pre-process experimental data and are aimed at obtaining the value of some input parameters of the white box model, rather than coefficients of the functional form embedded by it. Therefore, in these approaches the black box component is not actually employed to build the model, but rather to identify, in a robust manner, the input parameters, e.g., workload characterization, needed for its effective invocation.

Singh et al. propose the use of Clustering to determine the workload characteristics for a multi-tier system, i.e., average service demand and requests mix (Singh et al., 2010). A similar approach is undertaken by Mozafari et al. to identify different transactional classes, their computational requirements and data access pattern in a database (Mozafari et al., 2013).

Zhang et al., instead, starting from Little's Law, use regression to determine the service demand of different job types in a multi-tier architecture (Q. Zhang et al., 2007). Such demands are, then, supplied as input to a MVA-based performance model of the system.

Finally, a different approach is undertaken by the Elastizer framework, which is aimed at supporting automated provisioning for Map-Reduce (Dean & Ghemawat, 2008) jobs processing infrastructures in the Cloud (Herodotou et al., 2011). In Elastizer Decision Trees are used to estimate how some processing costs and other relevant parameters of Map-Reduce jobs change when moving from the development cluster (used for profiling and training the black box learner) and the production one (where the jobs submitted by users are actually executed). Such costs are provided as input to an analytical model, which produces jobs running time estimates.

Among the proposed gray box modeling techniques, *Parameter Estimation* solutions may be seen as related to the *Divide et impera* methodology: also this technique, in fact, encompasses the possibility of some models to provide input to other ones.

The *Divide et impera* approach, however, is fundamentally different from existing *Parameters Estimation* techniques. In these solutions, in fact, the performance function is ultimately expressed only by means of white box modeling: black box approaches are solely undertaken to provide inputs to the model, and not to predict the performance of a whole sub-module of the target applications. On the other hand, the *Divide et impera* approach encompasses the definition of multiple, distinct performance models, which can be either white box or black box. In addition, black box modeling is not used to solely initialize white box predictors as in existing *Parameter Estimation*-based solutions. Rather, as explained in more detail in the next chapter, the models can exhibit arbitrary input dependencies and, once solved, their outputs are reconciled to produce the performance prediction for the target application as a whole.

3.1.3.3 Hybrid Ensemble.

This modeling methodology is descended from the Ensemble Learning techniques originally introduced in the ML community (see Section 3.1.2.4); it comprises different solutions that combine multiple instances of black box and white box predictors with the aim of building a gray box model whose predictive accuracy is higher than any of its components alone.

In the proposed taxonomy, this rather general principle can be implemented according to two paradigms, namely the *Patching* and the *Selection* one.

Patching. In Patching-based *Hybrid Ensemble* techniques, white box modeling is used to instantiate a base performance predictor, whose accuracy is refined via black box techniques. In particular, the black box model construction phase takes as input both the output of the base white box predictor and performance measurements collected from the operational system, with the aim of correcting inaccuracies in the function initially encoded by the base white box model.

The proposed taxonomy identifies two hybrid techniques that implement such principle: Bootstrapping and Hybrid Boosting.

Bootstrapping. This *Patching* technique consists into exploiting a white box model to initialize the knowledge base, i.e., the training set, of a black box learning algorithm. This allows a black box model to be initially instantiated without the need for a training phase on the live system; the training set can be updated over time to include samples collected from the running system, and a new model can be trained, so as to include this new available factual knowledge about the system behavior. The peculiar feature of this Hybrid Ensemble technique is that, after being used in the initialization phase, the white box model is put aside: once the initial training set has been populated, in fact, both the learning and the prediction phases are performed by a black box algorithm.

The Bootstrapping technique has been applied in the context of both on-line and off-line learning. In the on-line domain, Romano and Leonetti (Romano & Leonetti, 2012) propose the use of UCB to minimize the messages delivery latency in a Total Order Broadcast implementa-

tion depending on the messages arrival rate. This solution relies on discretizing the domain of the encompassed arrival rates: one UCB instance is associated with each of the obtained messages arrival rate intervals, and is responsible for the optimization of the target application in the corresponding workload conditions. An analytical model, based on Queueing Theory, is used to initialize the state of the UCB instances, so as to reduce their learning phase. Schroeder et al. (Schroeder et al., 2006), instead, exploit hill-climbing (Russell & Norvig, 2003) to tune the number of active threads in a database; an analytical model is responsible to identify a suitable starting point for the hill-climbing, in such a way to achieve faster convergence to the optimal solution.

An on-line solution that is related to Bootstrapping has also been proposed by Tesauro et al. (Tesauro et al., 2007). In their solution, an analytical model is used to build an approximate performance model to initially guide the automatic provisioning process in a data center. A Reinforcement Learning algorithm is trained on data collected while the analytical model-based policy controls the system; at steady state, the RL-based policy completely replaces the AM-based ones and implement a continuous process of refinement of the learned provisioning scheme.

In the off-line domain, IRONMODEL (Thereska & Ganger, 2008) represents, to the best of our knowledge, the first attempt to initialize the knowledge base of a regressor (specifically, a DT) by means of analytical modeling combine analytical, and has been proposed in the context of anomaly detection in data centers. IRONMODEL relies on human supervision to detect the deviation of an application/component's performance from the expected one and to trigger, as a consequence of such a deviation, a corrective phase. This phase consists in the execution of a set of synthetic tests specifically crafted to replicate the conditions (i.e., workload) that lead to the deviation between observed and predicted performance and that are aimed at providing enough data to induce the black box learner to correct the original discrepancy between observed behavior and model's prediction. In a similar fashion, Rughetti et al. (Rughetti et al., 2014) propose to train an ANN with a mix of samples generated by an analytical model and obtained by available execution traces of the target application. The obtained model is used to tune the number of active threads in a single-node Transactional Memory system.

This dissertation makes a twofold contribution in the field of off-line *Bootstrapping*-based performance modeling, *i*) by presenting the most in-depth study, to the best of our knowledge, of the Bootstrapping technique to date and *ii*) by proposing novel algorithms to implement the *Bootstrapping* technique. The work presented in this thesis, in fact, explores extensively the design space of this methodology, proposes new algorithmic variants and provides novel insights on its effectiveness, thanks to a comprehensive experimental evaluation on two different case studies.

With respect to IRONMODEL (Thereska & Ganger, 2008), the proposed Bootstrapping implementation relies on a totally automated work-flow, which is capable of correcting the inaccuracies of the base white box model without requiring any human intervention. In addition, IRONMODEL does not address two crucial issues, which, as shown in Section 5.4.2, strongly impact the accuracy of a bootstrapped model, namely how to initialize and update the knowledge base of the black box model. Conversely, the proposed Bootstrapping technique encompasses different implementations and parameterizations for these operations, and their impact on the accuracy of the resulting gray box model is thoroughly investigated.

The solution proposed by Rughetti et al. (Rughetti et al., 2014) is, to the best of our knowledge, the most similar to the proposed Bootstrapping technique[2]. With respect to it, however, the investigation on Bootstrapping provided in this thesis presents several differences, both at the design and experimental level.

On the design side, this dissertation proposes and evaluates *i*) an automated mechanism to determine how many sample to adopt for the initial training set obtained by querying the base white box model; *ii*) several variants to update the (initially "white") training set with samples collected on the running applications; and *iii*) the use of weighting, to allow the black box learning algorithm to give more relevance to real samples over synthetic ones. Conversely, the proposal by Rughetti et al. does not specify how to properly size the synthetic training set, does not investigate the use of weighting and only encompasses one training set update technique (which the performed experimental evaluation, reported in Section 5.4.2.2, has shown to be

[2]It is important to point out that the investigation on Bootstrapping presented in this dissertation has been performed concurrently with and independently from Rughetti et al. (Rughetti et al., 2014)

sub-optimal for the considered case studies).

On the experimental side, this work differs from the one by Rughetti et al. for several reasons: *i*) the Bootstrapping technique is evaluated by using two, very diverse, case studies. This allows for assessing the impact that the characteristics of the performance function and of the corresponding performance model have on the delivered accuracy of the bootstrapped predictor; *ii*) a thorough sensitivity analysis is performed, with the aim of understanding how the setting of the several encompassed parameters affects accuracy; and *iii*) an experimental evaluation of the accuracy of Bootstrapping when working in extrapolation is performed, so as to assess the robustness of this technique when the "patching" of the base white box model can be performed only in a limited region of the input space.

Note that most of the aforementioned contributions could also be extended to on-line solutions, like the one developed by Romano and Leonetti (Romano & Leonetti, 2012). The main difference between off-line and on-line Bootstrapping, in fact, is that on-line approaches do not maintain a training set, as they already specify how the state/knowledge of the learner (e.g., a UCB instance) is updated whenever a new sample is received. Conversely, updating the training set to include factual knowledge coming from the real system is a key function in off-line bootstrapped predictors. Therefore, except from the novel training set updating algorithms, the contributions aimed at addressing the other limitations of off-line Bootstrapping equally apply to the on-line case.

Hybrid Boosting. Hybrid Boosting builds upon the Additive Logistic Regression algorithm (J. Friedman et al., 2000), namely a Boosting-oriented solution for regression: this technique organizes several instances of learning algorithms in a chain, such that each algorithm is trained to learn how to correct for the inaccuracies of the previous one (also referred to as *residuals*).

To the best of our knowledge, this *Correction*-based *Hybrid Ensemble* technique has never been proposed in the literature of performance modeling before. The Hybrid Boosting technique proposed in this dissertation inherits from the Additive Logistic Regression algorithm the sequential training of learning algorithms over residual training sets; it places, however, a white

box model as first in chain. This allows Hybrid Boosting to leverage some knowledge domain that is encoded in the base white box model and to employ black box learning algorithms to incrementally refine its predictions by correcting for its inaccuracies. To the best of our knowledge, the idea of specializing the boosting algorithm to learn correcting functions for a white box performance model is still unexplored in the literature.

Unlike Bootstrapping, Hybrid Boosting does not discard the white box model once it has been used: as detailed in Section 5.2.2, in fact, it is retained to be queried both upon re-training the chain of black box learners, as well as at prediction time.

Selection. According to this *Hybrid Ensemble* principle, white box and black box models are built independently, and a black box classifier is trained to determine, depending on the input configuration x, which is the model, among the available ones, that is expected to produce the most accurate performance prediction corresponding to x.

This principle is implemented by three solutions, namely Local Selection, Hybrid KNN and Probing. The first one has already been proposed in literature; the other two represents a novel contribution of this dissertation.

Local Selection. This approach is implemented by the Chorus framework (Chen et al., 2013), which relies on the availability of several white and black box models to predict the performance of general applications in a data center. For this reason, Chorus' white box models are typically simple, as they are designed to capture asymptotic behaviors of the target application in well specified operational conditions, for example in cases in which the workload is CPU-, disk- or memory-bound. Black box models include SVMs and simple regression models (linear, polynomial or exponential functions), whose parameters are determined by fitting the output of the various models to the data in the training set.

These models are all trained independently but queried selectively, depending on the input sample. In particular, the inputs space of the target performance function is initially discretized and partitioned into regions, and the target application is profiled over a number of configurations. The resulting data set is employed both to train the black box learners and to perform

a validation phase aimed at assessing which predictor performs best for each of the regions corresponding to the aforementioned partitioning of the input space.

Hybrid KNN. This approach leverages the availability of potentially more than one white box model and several black box learning algorithms. According to this technique, the available data-set is partitioned intro a training and a validation set (noted TS and VS respectively): TS is used to train the black box models, whereas VS is used at query time. In fact, to predict the query corresponding to a sample x, Hybrid KNN evaluates the accuracy of the available models in predicting the performance for the samples in VS that are more similar to x; the models that minimize predictive error is selected to predict the performance corresponding to x.

Probing. This technique relies on the availability of potentially multiple white box models and a single black box learning algorithm, which is trained only in regions of the feature space corresponding to unsatisfactory accuracy of any of the white box predictors. Probing, therefore, aims at building a black box model that is as specialized, and, hence, as accurate as possible, by narrowing the black box learning problem to circumscribed regions of the feature space. In order to determine which among the available models must be used to serve a specific performance prediction query, an arbitrary classifier can be used.

The two Selection-based techniques proposed in this dissertation are orthogonal to the approach undertaken by the aforementioned Chorus framework. One important distinction, however, is that Chorus must undergo a preliminary discretization phase, aimed at defining the regions over which the different models are specialized. This step has a crucial impact on the effectiveness of the approach: if regions are too wide, the corresponding models will be under-specialized; if they are too narrow, the training phase is going to be negatively affected as it must collect a huge amount of data to understand which model is the best for a huge number of regions.

Unfortunately, Chorus does not provide an automated way to perform such critical operation, leaving to the end user the burden of identifying the most effective discretization technique to learn the target performance function (Liu et al., 2002). On the other hand, both Hybrid KNN

and Probing avoid the need for discretization; moreover, the effect of internal parameters that affect their operational behavior is thoroughly investigated and solutions for their automated tuning are provided.

As a final note on *Hybrid Ensemble* techniques in general, it is important to stress that these solutions are orthogonal and complementary to the other two aforementioned gray box performance modeling methodologies. In fact, as detailed in the next chapters, *Divide et impera*-based and *Parameter Estimation*-based performance models can be used as base building blocks to build *Hybrid Ensemble* performance predictors. The study conducted in this dissertation, therefore, also shows how different gray box performance modeling techniques can be combined via the *Hybrid Ensemble* methodology, with the aim of building more accurate and robust performance predictors.

To conclude this section, Table 3.1 summarizes the discussion conducted so far about the gray box performance modeling paradigm, by reporting the techniques identified in the proposed taxonomy, a brief description of the solutions that implement them and the contributes made by this dissertation to advance the state of the art in each of such methodologies.

Technique	Description	Implementations	Thesis contributions
Divide et impera	Divide the performance prediction problem into sub-tasks and tackle each of them by means of the most appropriate modeling technique (i.e., black or white).	This technique has been applied to model the performance of DTPs applications: white box modeling is applied to capture contention/replication dynamics and CPU response time; black box modeling is applied to predict latency of network-bound transactional operations. White box techniques employed to this end are Analytical Modeling (Didona, Romano, et al., 2012; Didona et al., 2014; Didona & Romano, 2014b) and Simulation (Di Sanzo et al., 2015).	To the best of our knowledge, this technique is a novel contribution of this dissertation. Moreover, the proposed *Divide et impera* models are employed as building block to compose *Hybrid Ensemble* predictors.
Parameter Estimation	Model-Free: use black box techniques to obtain input parameters of a white box model	Use Clustering to obtain workload characterization (Singh et al., 2010; Mozafari et al., 2013) or DT to predict job service demands when changing deployment infrastructure (Herodotou et al., 2011).	The thesis shows how predictors based on this technique can be further improved upon by means of *Hybrid Ensemble*. In fact, this thesis evaluates the proposed *Hybrid Ensemble* models also using as base predictor an analytical model whose parameters values are obtained by fitting.
	Model-Based: use white box to obtain a parametric, functional form of the target KPI; use black box techniques to infer the parameters of the function via fitting over experimental data.	Approach borrowed from the System Identification field (Ljung, 1999). In the context of computer systems performance modeling, techniques employed to obtain the parameters for the white box functional form are regression (Heiss & Wagner, 1991; Q. Zhang et al., 2007; Di Sanzo et al., 2013) and Kalman Filters (Woodside et al., 2008).	
Hybrid Ensemble	Selection: build multiple white and black box models in parallel and use, depending on the incoming performance prediction query, only the one that is expected to maximize accuracy.	Local Selection: partition the feature space in region and find the per-region most accurate model.	Partitioning the input space is a hard discretization problem, for which an automated solution has not been proposed in the original paper. This thesis, instead, provides an analysis of the proposed approaches to the setting of internal parameters, and provides solutions for their automatic tuning.
		Hybrid KNN: find the training samples that are more similar to the input one; use the model which is more accurate in predicting their performance.	Proposed in this thesis.
		Probing: train a black box model only in region of the input space where white box modeling is not accurate enough; use a generic classifier to decide which model to use depending on the input.	Proposed in this thesis.
	Patching: rely on a base white (or gray) box predictor and use black box techniques to progressively correct its output by incorporating factual knowledge about operational behavior of the system.	Hybrid Boosting: train a chain of black box algorithms to learn the error distribution of the base predictor, so as to build a "patching" function to correct for these inaccuracies.	Proposed in this thesis.
		Bootstrapping: in the off-line case, it uses white box models to produce an initial training set for a black box learning algorithm, which can be updated over time with samples corresponding to real input-output performance measurements (Thereska & Ganger, 2008; Rughetti et al., 2014); in the on-line case, it uses white box models to initialize Reinforcement Learning (Romano & Leonetti, 2012) or on-line optimization (Schroeder et al., 2006) algorithms.	This thesis provides the most extensive study and evaluation of the off-line *Bootstrapping* technique to date; moreover, it proposes new algorithmic implementations of the functions responsible for initializing the white box model-based training set and for updating it with factual input-output performance samples collected from the running system.

Table 3.1: Main limitations affecting existing performance models for DTPs and corresponding solutions proposed in this dissertation.

3.2 Performance Modeling and Self-tuning of DTPs

The problem of modeling and self-tuning performance of DTPs has been extensively investigated in literature, with solutions tailored for distributed databases (Nicola & Jarke, 2000; Elnikety et al., 2009; Soundararajan et al., 2009), data-grids (Di Sanzo et al., 2012, 2014; Couceiro et al., 2013) and transactional memory systems (Couceiro et al., 2011; Kobus et al., 2013).

Solutions for modeling DTPs performance can be divided into two classes. Although it may seem counter-intuitive at first, this classification is based on whether the performance model explicitly takes into account specific dynamics stemming from the adoption of the transaction abstraction, like data conflicts and global synchronization phases caused by data replication. The first class of solutions is hereafter referred to as *transactions-unaware* and the second as *transaction-aware*.

As already hinted, the proposal of transaction-unaware performance models for DTPs may appear a paradox, but it is not. Solutions belonging to this class, in fact, are tailored to model either *i*) transactional systems that, although distributed and dependable, do not encompass a distributed commit phase (e.g., by relying on Passive Replication and logging to persistent storage at the primary node for fault tolerance (Singh et al., 2010)) or *ii*) read-intensive workloads, or workloads that are assumed to induce negligible contention on data (Q. Zhang et al., 2007). The benefit of employing this approach to performance modeling of DTPs is twofold: transaction-unaware models *i*) can be exploited to capture the performance dynamics of many kinds of applications, like multi-tier ones (Dejun et al., 2010, 2011) or weakly consistent data platforms (Trushkowsky et al., 2011; Stewart et al., 2013) and *ii*) avoid the complexity of dealing with the dynamics stemming from concurrency control (CC) schemes and replication protocols. The resulting model, therefore, may rely on fewer assumptions, and may be able to exploit more detailed workload characterizations (e.g., generic service demands instead of exponentially distributed) and finer-grained KPI predictions (e.g., percentiles instead of mean values) than many transactions-aware models (particularly, AM-based ones). Consequently, transactions-unaware performance models for data platforms are by far more common than the transactions-aware counterparts, especially looking at the literature of the last ten years devoted to automatic resource provisioning schemes for Cloud applications (Galante & Bona, 2012).

The DTP performance models proposed in this dissertation are transactions-aware; however, this section also surveys state-of-the-art on transactions-unaware performance models. Such analysis is performed because of the wide adoption of this kind of solutions, especially in the context of multi-tier architectures that encompass transactional storage.

Given the vastness of the literature, first Section 3.2.1 and Section 3.2.2 provide an overview of the works that addressed the performance modeling of DTPs using, respectively, transaction-aware and transaction-unaware approaches. The focus of these sections is on describing the target systems to which existing performance models have been applied, and on providing an overview of the methodologies they employed. Section 3.2.3, then, presents a critical analysis of the relations between these works, with the aim of discussing their shortcomings and how they have been addressed by the contributions of this dissertation.

3.2.1 Transactions-unaware Performance Modeling for DTPs

Unlike their white box counterpart, black box models do not typically need to make simplifying assumptions about the replication scheme of the platform or the conflict likelihood of the target application: as long as a characterization of these aspects is provided as input to the learning algorithm, they will be taken into account in the performance model. For this reason, transactions-unaware performance models for DTPs are typically white box.

This class of models typically relies on the implicit assumption that performance of the target application is inherently scalable, i.e., adding more computational nodes to the platform results into better performance. The most prominent models of this kind are based on Queuing Theory and typically find applications in automatic resource provisioning schemes for Cloud applications deployed according to a multi-tier architecture. In these solutions, the system is modeled as a network of queues, in which requests flow from queue to queue, i.e., from tier to tier, until completion.

Singh et al. (Singh et al., 2010) consider an open system and employ a clustering algorithm to characterize the workload mix of the target application. A multi-class analytical model is then built, in which each class models the different arrival rate and resource demand of a cluster.

Such a model is able to determine the number of nodes to allocate to the tiers of a multi-tier application in order to guarantee a given QoS in terms of a high percentile of the response time distribution. This task is accomplished by modeling each server as an open G/G/1 queue.

Other works, instead, model the system as closed, in order to better capture the dynamics of session-based workloads. Urgaonkar et al. (Urgaonkar et al., 2005) employ G/G/1 queues whose arrival rate is adjusted to take into consideration multiple visits to the same queue stemming from the same session; Zhang et al. (Q. Zhang et al., 2007), instead, assume exponentially distributed service demands, thus solving the model by means of the MVA algorithm (see Section 3.1.1).

The main limitation of these works is that they only consider homogeneous platforms, i.e., in which each node in the system is equal to each other. Cloud providers, on the other hand, support multiple type of Virtual Machines (VM), which are differentiated by computational power. On one hand, this gives the possibility of tailoring the resource provisioning to the current workload at a finer granularity, leading to higher efficiency; on the other hand, taking advantage of machines with differentiated characteristics requires the definition of performance models capable of specifically taking into account the heterogeneity in the computational power of VMs.

Sharma et al. (Sharma et al., 2012) propose a heterogeneity-aware model, in which each tier is modeled as a $M/G/1/PS$ queue where the generic distribution of the service time is obtained by fitting observed response time on several shifted exponential distributions. The parameters for these distributions are obtained by means of a clustering algorithm.

The solution proposed by Dejun et al. (Dejun et al., 2011) deals with heterogeneity in Cloud platforms at two levels. In fact, not only it encompasses the possibility of provisioning the target platform with machines characterized by different computational powers (both in terms of CPU and I/O); it is also able to detect possible mismatches between the computational power that is advertised by the Cloud provider and the one that is effectively delivered by the machine (Dejun et al., 2009). This is accomplished by means of specific micro-benchmarks that are executed on newly acquired machines and are able to accurately characterize their computational power. Thanks to this profiling phase, the proposed scheme is able to determine

whether a given machine would be better exploited to serve CPU-intensive jobs (e.g., web-server workloads) or I/O intensive ones (e.g., database workloads). Moreover, by modeling nodes as $M/G/N/PS$ queues, it is able to support a smart load balancing scheme that is able to determine how many requests can be dispatched towards a given machine so as to match the desired QoS.

A final class of white box transactions-unaware models is represented by solutions that are able to take into account specific characteristics of the target system. The model proposed by Urgaonkar et al. (Urgaonkar et al., 2005), for example, is able to model possible caps on the maximum concurrency degree, which is a typical case for web-servers (e.g., Apache) and DBMS (Schroeder et al., 2006): if a server is configured to serve at most K concurrent jobs, then it is modeled as a $M/M/1/K$ queue.

Dejun et al. (Dejun et al., 2010) propose a model that is, instead, able to capture the effect that provisioning one tier has on the other ones. In the model, tiers are organized according to a hierarchical tree-graph, with the front-end tier being the root. The effect of a provisioning decision (acquire/release a node or relocate it to a different tier) at one level of the tree is propagated on other levels in order to determine its impact on the whole system. Thanks to this approach, for example, the model is able to predict the effect that provisioning the cache tier in a multi-tier architecture, and thus changing the cache hit rate, has on back-end tiers. Performance of single nodes in the system are predicted by modeling them as $M/M/K/PS$ queues.

3.2.2 Transactions-aware Performance Modeling for DTPs

There is an abundant body of work on both white and black box transactions-aware performance models for DTPs. The former mostly come from the literature on capacity planning of distributed databases; the latter are more common in the literature on self-tuning and automated resource provisioning, prominently spurred by the advent of Cloud Computing.

3.2.2.1 Black box models

Solutions based on black box models relax the most critical assumptions of the aforementioned works, i.e., the inherent scalability assumption and the conflict/replication unawareness. Thanks to their black box nature, in fact, they just require a characterization of the conflict likelihood of the target workloads and of the replication settings of the platform as input, in order to model their impact on performance.

In the context of database automatic resource provisioning, Chen et al. (Chen et al., 2006) propose a KNN-based approach in which the machine learner is trained off-line, i.e., before actually deploying the system to serve client requests. In a subsequent work (Ghanbari et al., 2007), the off-line training phase is only used to determine the features that are correlated the most with the target performance metric and to initialize the knowledge base of a SVM learner. This is responsible, at runtime, to guide the database provisioning scheme, and it is updated in an on-line fashion with samples gathered from the operational system corresponding to scenarios whose prediction had been inaccurate.

Different solutions have also been proposed to self-tune and provision other kinds of distributed transactional platforms, e.g., data-grids like Infinispan or Distributed Transactional Memories (Couceiro et al., 2011).

Di Sanzo et al. (Di Sanzo et al., 2012) propose the exploitation of ANN to determine the replication degree that maximizes throughput in in-memory distributed transactional data grids. The solution is extended in a following work (Di Sanzo et al., 2014), in which the ANN-based model is aimed at providing QoS on a per-transactional class basis, by jointly tuning the replication degree and the amount of allocated resources.

Self-tuning of the replication protocol has been investigated especially in the context of Distributed Transactional Memories. The authors of PolyCert (Couceiro et al., 2011) investigate the exploitation of both off-line regression algorithms (specifically DTs) and Reinforcement Learning (specifically UCB) to determine, on a per-transaction basis, which replication protocol to use among three different TO-based approaches. MORPHR (Couceiro et al., 2013), instead, employs C5.0 (Quinlan, 1996), a DT classifier, to switch among a TO-based, a 2PC and a PR

replication protocol depending on the workload that the application is generating.

3.2.2.2 White box models

White box transaction-aware models are explicitly crafted to capture how the concurrency control scheme and replication protocol reflects not only on resources utilization (CPU, disk, network) but also on data contention. The literature on performance modeling for DTPs is heavily oriented towards distributed database environments, and is rich in both simulation-based (Carey & Livny, 1988; Di Sanzo et al., 2015) and analytical models (Nicola & Jarke, 2000; Elnikety et al., 2009). The following discussion is specifically focused on surveying different modeling choices and techniques in the context of analytical models, as they are more closely related to the model proposed in Section 4.

The differences among the plethora of analytical models for DTPs that have been proposed throughout the years lie in the kind of considered system, the set of aspects and parameters of the platform that are captured and the way they are modeled.

3.2.2.2.1 System model. Open queueing networks are at the basis of many works, e.g., (Raghuram et al., 1992; Ciciani et al., 1990, 1992); others exploit a closed system model and solve it by means of the MVA algorithm, e.g., (Elnikety et al., 2009; Q. Zhang et al., 2007). Open models are more common, and the arrival rate is typically assumed to follow an exponential distribution, because it has been found to be a good approximation for a large number of users submitting jobs independently (Gallersdörfer & Nicola, 1995). On the other hand, closed systems are more suitable to model session-based workloads (Q. Zhang et al., 2007; Urgaonkar et al., 2005) or to capture dynamics of solutions that limit the number of concurrently active transactions to reduce data contention and resource sharing (Schroeder et al., 2006; Urgaonkar et al., 2005). Yu et al. (Yu et al., 1993) propose an iterative solution that consists in modeling the system as open and then exploiting Little's law to approximate the behavior of a closed system; a more accurate description of this technique will be provided in Section 4.2.4.

3.2.2.2.2 Data Management Scheme. Analytical models for distributed databases typically focus on serializability (Ciciani et al., 1990, 1992; Raghuram et al., 1992). This isolation level is obtained by means of two-phase locking (Ciciani et al., 1990; Raghuram et al., 1992)

(locks are acquired during the execution of transaction), optimistic locking (Ciciani et al., 1992) (locks are acquired only at commit time) or multi version concurrency control (Ren et al., 1996; Son & Haghighi, 1990) (several versions of each datum are stored). Wojciechowski et al. (Wojciechowski et al., 2012) analyze and compare the performance of a Distributed Transactional Memory delivering serializable isolation level by means of two different TO-based replication protocols.

Weaker isolation levels are also investigated. Elnikety et al. (Elnikety et al., 2009) consider Snapshot Isolation (Berenson et al., 1995), enforced by a single node in the system which acts as certifier; Gallersdorfer and Nicola (Gallersdörfer & Nicola, 1995) study the benefits coming from relaxing consistency by means of a temporal parameter which defines how much time a replica may lag behind another replica of the same logical data object.

Another aspect that has been largely investigated is the impact of data replication on performance of DTPs. Many policies have been studied and evaluated via analytical modeling in literature, ranging from the no distribution case (i.e., data partitioning) (Son & Haghighi, 1990; Raghuram et al., 1992), to the full replication one (Garcia-Molina, 1979; Sheth et al., 1985). Intermediate policies entail different kinds of partial replication including *i*) every datum is stored by a fixed number of replicas and *ii*) each datum has a probability r of being fully replicated and $1-r$ of not being replicated at all (Gallersdörfer & Nicola, 1995). Other proposed policies allow for specifying a replication degree on a per-datum basis (Carey & Livny, 1988, 1991); nevertheless their instantiation has only been considered for a simple case, in which each datum has a uniformly distributed replication degree, thus narrowing down its applicability to the partial replication case. Ciciani et al. (Ciciani et al., 1992), instead, consider a hybrid distributed-replicated system in which a central node maintains the full dataset and other, geographically distributed, nodes only store data relevant to their zone.

3.2.2.2.3 Physical resources. Existing DTPs performance models are also very heterogeneous in the way in which they model physical resources, i.e., CPU, disk and network. Transactions are typically characterized by exponentially distributed service demands (Ciciani et al., 1990, 1992; Raghuram et al., 1992; McDermott & Mukkamala, 1994; Elnikety et al., 2009; Menascé & Nakanishi, 1982), and are served in a FCFS fashion. Some models are more com-

plex from this point of view, as they entail generic (Alonso et al., 1990), or 2-phase hyper-exponentially CPU service demands (Gallersdörfer & Nicola, 1995; Att & Leung, 1997) to capture the variance in the distribution, or more sophisticated jobs serving policy (Round Robin in (Att & Leung, 1997)). Disks are generally modeled as $M/M/1$ queues (Menascé & Nakanishi, 1982) or as $M/D/\infty$ (Ciciani et al., 1990), i.e., servers with infinite capacity which result into fixed delays.

As for inter-node communication modeling, most performance studies assume assume infinite network bandwidth, i.e., they neglect queueing effects at the network level. The response time is supposed either to be fixed ($M/D/\infty$ server) (Garcia-Molina, 1979; Ciciani et al., 1990, 1992), exponentially distributed ($M/M/\infty$) (Shyu & Li, 1990; Kuang & Mukkamala, 1991) or generic ($M/G/\infty$) (Ren et al., 1996). In some studies (Gallersdörfer & Nicola, 1995; Nicola & Jarke, 2000) messages containing read/write sets are modeled as bigger than distributed protocol messages (e.g., commits), whereas in others this distinction is not present (Menascé & Nakanishi, 1982). More realistic proposals model the network layer at each node as a queue with a finite capacity, thus encompassing queueing effect on the inter-node communication medium (Sheth et al., 1985; Raghuram et al., 1992; Nicola & Jarke, 2000; Knottenbelt et al., 2001).

3.2.2.2.4 Workload characterization. One of the key aspects of analytical models for transactional platforms is the workload that they are able to capture, i.e., the mix of transactional classes and the data access pattern that they exhibit.

The great majority of analytical models for transactional platforms assumes that the dataset is uniformly accessed (Raghuram et al., 1992; Garcia-Molina, 1979; Elnikety et al., 2009); this implies that, whenever a transaction issues a read/write operation on a dataset of cardinality D, any datum has a probability $\frac{1}{D}$ of being chosen. This assumption is clearly far from being matched in typical real world scenarios, thus yielding to the proposals of increasingly more complex data access pattern modeling techniques.

To the best of our knowledge, more sophisticated data access patterns, from the point of view of data conflicts, are considered in solutions tailored for centralized transactional systems. These solutions are surveyed in the following, as they are closely related to the workload char-

acterization technique proposed in Section 4.

Tay et al. (Tay et al., 1985) introduce the *b*-*c* data access pattern, in which the $b\%$ of transactions access the $c\%$ of the total number D of data item and the $1-b\%$ access the remainder. In the same work it is shown that, under a set of mild assumptions and the *a priori* knowledge of the $b-c$ parameters, it is possible to approximate this access pattern through an equivalent uniform one. This model allows for capturing the presence of *hot spots* in the dataset, i.e., elements that are accessed more frequently than others.

A generalization of the $b-c$ model is provided in following works (B. Zhang & Hsu, 1995; Thomasian, 1998). There, the stream of requests is partitioned in transactional classes and the dataset is divided in disjoint sub-datasets. Each sub-dataset corresponds to a class, and elements in each of them are accessed according to a uniform distribution.

The uniformity assumption is further relaxed in a following work (Di Sanzo et al., 2008), which provides a model of a multi-version concurrency control scheme. In this work, every data item x is given an access probability $P(x)$; this allows for capturing more complex non-uniform data access patterns. In particular, the model is validated, via simulation, against different parameterizations of the $zipf$ function such that $P(x) = zipf(x)$.

Finally, more sophisticated models allow for defining even more complex data access patterns, according to which each distinct operation can access a given portion of the whole dataset (Thomasian, 1994; Mozafari et al., 2013; Di Sanzo et al., 2010). The detailed knowledge about the data items accessed by each operation is carried out by means of off-line tracing.

In distributed environments, the data access pattern also includes the locality of accesses, i.e., how likely it is that a datum is collocated with the transaction that wants to access it. This depends both on characteristics of the workload and on the data replication scheme/data placement adopted by the hosting platform.

Regarding the characterization of a workload's locality, the $b-l$ locality model (Raghuram et al., 1992; Hwang et al., 1996) entails that the $b\%$ of the request can be served without any remote interaction; Alonso et al. (Alonso et al., 1990), instead, consider the case of a non-transactional system in which a percentage of data items is cached upon remote access, and is

f time more popular than other data items.

Regarding the modeling of the shift of the locality as a function of the nodes composing the distributed platform, existing works either consider all data required by a transaction to be stored on a node (whether it is the local or a remote one) (Hwang et al., 1996; Gallersdörfer & Nicola, 1995) or do not investigate the shift of data locality when scaling the platform (i.e., they assess the impact of locality for different scales, but the probability of a local access is an invariant of the application) (Ciciani et al., 1990, 1992; Raghuram et al., 1992).

Finally, some works focus on analytically deriving the probability distribution for the number of nodes involved in the execution of a distributed transactions, but do not couple the proposed stochastic model with a performance one (Mukkamala & Bruell, 1990; Simha & Majumdar, 1997; Thomasian, 1993).

Feature / Work	Modeling Methodology	Conc. Contr.	What-if support	Distributed	Data Contention		Replication			Locality		Networking model
					Uniform	Skewed	Full	Partial	None	Fixed	Variable	
(Elnikety et al., 2009)	White	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	Fixed delay
(Yu et al., 1993)	White	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	N/A
(Di Sanzo et al., 2008, 2010; Mozafari et al., 2013; Thomasian, 1994)	White	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	N/A
(Raghuram et al., 1992)	White	✓	✓	✓	✓	✗	✗	✗	✓	✓	✗	M/M/1 queue
(Tay et al., 1985)	White	✓	✓	✗	✗	✓	✗	✗	✗	✗	N/A	✗
(Gallersdörfer & Nicola, 1995; Nicola & Jarke, 2000)	White	✗	✓	✓	✓	✗	✓	✓	✓	✗	✓	$M/M/1$
(Di Sanzo et al., 2012, 2014; Chen et al., 2006; Ghanbari et al., 2007)	Black	✓	✓	✓	–	–	–	–	–	–	–	—-
(Urgaonkar et al., 2005; Dejun et al., 2010, 2011)	White	✗	✓	✓	–	–	–	–	–	–	–	—-
(Ciciani et al., 1990)	White	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	Fixed delay
(Ciciani et al., 1992)	White	✓	✓	✓	✓	✗	Hybrid full / partial			✓	✗	Fixed delay
(Hwang et al., 1996)	White	✓	✓	✓	✗	✗	✓	✓	✓	✗	✓	Fixed delay
(Alonso et al., 1990)	White	✗	✓	✓	✗	✗	✓	✓	✓	✓	✗	$M/M/1$
(Knottenbelt et al., 2001)	White	✓	✓	✓	✗	✗	Centralized / none			✗	✓	$M/G/1$
(Sheth et al., 1985)	White	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	$M/M/1$
(Menascé & Nakanishi, 1982)	White	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	Fixed delay
(Shyu & Li, 1990)	White	✓	✓	✓	✓	✗	✗	✗	✓	✓	✗	$M/M/\infty$
(McDermott & Mukkamala, 1994)	White	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	Fixed delay
(Garcia-Molina, 1979)	White	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	Fixed delay
(Couceiro et al., 2011)	Black	✓	✗	-	-	-	✓	✗	✗	✓	✗	-
(Couceiro et al., 2013)	Black	✓	✗	-	-	-	✓	✗	✗	✓	✗	Commit dur. = feature
(Q. Zhang et al., 2007)	Gray	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	–
Proposed Models	Gray	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	Black Box

Table 3.2: Comparison between some of the main existing DTPs performance models and the proposed ones.

3.2.3 Critique to state of the art DTP performance modeling and thesis contributions

This section is devoted at analyzing the limitations that affect the surveyed state-of-the-art solutions in performance modeling and optimization for DTPs and at describing how the proposed models and modeling techniques overcome them.

Table 3.2 compares the most relevant DTP performance models with the ones proposed in this dissertation. The comparison is performed on the basis of the main characteristics of the corresponding performance model or optimization scheme: the following discussion is aimed at highlighting in which aspects state-of-the-art solutions fall short, and how the proposed gray box models and modeling techniques overcome their limitations.

Modeling methodology. The first characteristic existing solutions can be grouped by is the modeling methodology that they embrace, namely white or black box. As it is possible to see, black box solutions typically encompass the majority of the most important features of a DTP performance model. This characteristic is descended by the fact itself that ML algorithms only need an input feature characterizing a given aspect of the system to include it in the model, taking the burden of statistically correlating it to the target KPI. For example, the work by Chen et al. (Chen et al., 2006) characterizes conflict likelihood by means of the number of locks requested by a transaction: this feature works equally well in case of uniform or skewed data access patterns. Likewise, MorphR (Couceiro et al., 2013) takes as input (among other features) the abort rate measured when running with a given replication protocol P and determines whether P is optimal for the current workload: as in the previous case, the abort rate feature is unaware of the data access pattern that has generated it.

Clearly, as already discussed, the key drawback of black box approaches is that they typically require long training phases. In order to derive models capable of accurately capturing the complex performance dynamics of DTPs, black box approaches need to be fed with a sufficiently large number of samples to be able to infer statistical relations between input and output variables.

White box approaches, on the other hand, typically focus on some aspects of the target platform and neglect others. The rationale behind this choice is, of course, to remove some complexity in the design of the model, and ease its analytical tractability. The most evident simplicity vs expressiveness trade-off lies in the modeling of the data access pattern: non-uniform popularity of data items, leading to hot-spots of data contention, are only encompassed by some works on single-node transactional system (Thomasian, 1994; Di Sanzo et al., 2010; Mozafari et al., 2013); conversely, models for DTPs consider non-uniformity in the locality of data accesses, but neglect skewed popularity of data items from the point of view of contention (Ciciani et al., 1990; Raghuram et al., 1992; Elnikety et al., 2009).

This simplicity vs expressiveness trade-off, as already discussed, reduces the applicability of analytical models to the scenarios in which simplifying assumptions hold.

On the other hand, the proposed DTPs performance models embrace the gray box modeling paradigm, and, specifically, represent an instantiation of the *Divide et impera technique*. The technique divides the performance prediction problem into sub-problems, each to be tackled by the most suitable modeling technique. In particular, as detailed later, the proposed performance models regard the networking layer as a black box, so as to overcome the availability of limited knowledge about the hosting infrastructure that is typical of virtualized environments. At the same time, by reducing the extent of the prediction task tackled by means of black box modeling, the *Divide et impera* approach also enables to instantiate a performance model requiring less samples than a pure ML-based approach.

In addition, these thesis also proposes the *Hybrid ensembling* gray box technique, which aims at enhancing the accuracy of a given white (or gray) box predictor by complementing its predictive capabilities with ML techniques applied to data collected from the operational system. This gray box technique aims at reconciling the strengths of the white and black box modeling techniques into a unified approach: by including a white box model component, which embeds domain knowledge, it allows for reducing the number of samples needed to build an accurate and reliable predictor in comparison to a pure black box approach; at the same time, the black box component allows for incorporating factual knowledge about the target system's performance, thus enhancing the accuracy of the original white box model.

Chapter 5 will show how *Hybrid Ensemble* methods can be applied to the proposed DTPs models to enhance their predictive capabilities and to make them more robust in performing predictions for workloads that do not meet some of their basic assumptions.

Transaction awareness. As discussed in the previous section, existing solutions can be coarsely classified based on whether they explicitly model concurrency and replication protocols or not. Section 2.2.1, however, has provided clear evidence of the critical effect that transactional dynamics have on performance and scalability of applications deployed on DPTs such as Infinispan.

Therefore, the proposed performance models embrace the transactional-aware design. In particular, they target different combinations encompassing two replication protocols (2PC and PR) and two concurrency control schemes (ETL and CTL), and consider both full and partial replication deployments.

What-if support. Typical solutions to support capacity planning and on-line optimization/resource provisioning rely on a performance model capable of supporting what-if analysis, i.e., to predict the *absolute* performance of a given workload in a target platform configuration.

On the other hand, the surveyed literature on DTPs performance modeling and optimization includes two solutions for DTPs optimization that do not provide what-if analysis support (Couceiro et al., 2011, 2013). The performance models proposed in those works are, in fact, not able to predict the performance under different workload conditions. Instead, they either rely on Reinforcement Learning or a classifier to determine which is the best configuration (specifically, in terms of employed replication protocol) for the workload currently generated by the application.

Also the solution proposed by Zhang et al. (Q. Zhang et al., 2007) suffers from some what-if capability limitations. In this work, CPU demands of transactions are inferred (i.e., not directly measured), by applying some Queuing Theory results, starting from the measurement of performance metrics like CPU utilization for the workload currently generated by the application. However, the CPU transaction demands that are derived from a workload mix that is very dif-

ferent from the one that is used in prediction might lead to inaccurate performance predictions, thus reducing the what-if capabilities of the proposed approach. This is, indeed, a limitation that this work shares with any black box approach: in that work, such limitation is descended from the adoption of black box techniques according to the *Parameter Estimation* gray box modeling approach (described in Section 3.1.3.2).

Note that, although only two among the solutions listed in Table 3.2 lack what-if support, Reinforcement Learning-based or even model free solutions for the autonomic provisioning and optimization of non transactional data platforms are very common (Tesauro et al., 2006; Tsoumakos et al., 2013; Cruz et al., 2013).

It is clear that the lack of what-if capabilities and quantitative modeling represents a major impairment to the exploitation of the Cloud business model: for example, without what-if support, it is impossible to estimate the operational costs to withstand a given workload. In order to meet this fundamental requirement, the proposed analytical models support what-if analysis on a number of parameters, including data platform's size, employed replication protocol and replication degree. Therefore, they can serve as building block for resource provisioning in the Cloud, and could equally find application in other contexts, e.g., anomaly detection.

Distribution, Data access pattern and Locality. Although the focus of this thesis is on performance modeling for DTPs, models for non-distributed transactional systems have also been reported, because of the different data access patterns that they encompass. On this regard, Table 3.2 reveals that the effects of non-uniform data access patters on the transaction conflict probability are only captured by performance models for single-node transactional platforms; in such models, the data access pattern is supposed to be known *a priori*, thanks to an off-line profiling phase.

In the light of this consideration, the analytical models proposed in this dissertation represent, to the best of our knowledge, the first white box solutions that address the modeling of skewed data access patterns (from the point of view of conflict likelihood) in a DTP. This is accomplished by introducing a novel abstraction, namely the Application Contention Factor (ACF), that allows to approximate a skewed data access pattern through an equivalent, uniform

one. In addition, obtaining the ACF does not require any expensive and intrusive off-line profiling of the application, needed by existing solutions: conversely, it can be computed on-the-fly from high-level statistics, which are cheap to collect and process.

Likewise, some models for DTPs consider non-uniform data access patterns for that concerns locality of accesses. However, only a few encompass shifts in locality as the platform's size changes and, unfortunately, also these models present some limitations: Knottenbelt et al. (Knottenbelt et al., 2001), for example, target a Distributed Lock Manager, in which transactions consists of single lock requests; other solutions, instead, only consider the case in which the locality is a property of a whole transaction, and not of a single access to a datum (Hwang et al., 1996; Gallersdörfer & Nicola, 1995).

On the other hand, the proposed models consider the shift of data locality as a consequence of the rescaling of the platform and the change of the replication degree; specifically, it targets the case in which the platform relies on consistent hashing to determine the mapping of data items to nodes. Moreover, it models locality at the granularity of the single data access, i.e., the case in which any single data access can be either local or remote.

In the light of these considerations, to the best of our knowledge, the proposed proposed models include the first white box models to jointly consider non-uniform data access patterns from the point of view of data conflict and shifts of data locality, at the granularity of the single data access, in a DTP.

Replication. Another aspect that differentiates the reviewed analytical models for DTPs is the degree of data replication that they support. As reported in Table 3.2, the vast majority of works only consider a sub-set of the possible replication policies, i.e., full, partial and no replication; only a few, including both white box (Ciciani et al., 1990; Nicola & Jarke, 2000) and black box (Di Sanzo et al., 2012) solutions encompass the whole spectrum.

The performance models proposed in this thesis encompass all three operational modes: in particular, they are tailored for replicated DTPs in which data replicas are distributed according to a consistent hash function, which is among the most widely used techniques to manage data distribution in modern data platforms (DeCandia et al., 2007; Marchioni & Surtani, 2012;

Oracle, 2011).

Network modeling. The final aspect used to classify existing solutions concerns the modeling of the networking layer. In distributed environments, network latencies are typically dominant over local processing times; therefore, the modeling of the networking dynamics is of paramount importance to determine the accuracy of any performance model for DTPs, given that inter-node synchronization is needed to retrieve remote data and to commit and to abort transactions.

Regarding this aspect, existing solutions either *i*) coarsely model network interactions as simple delays, by means of a fixed latency (Menascé & Nakanishi, 1982; Ciciani et al., 1990, 1992) or a queue with infinite servers (Ren et al., 1996; Shyu & Li, 1990; Kuang & Mukkamala, 1991), or *ii*) rely on the detailed knowledge of the network topology and the characterization of messages sizes and service demands (Raghuram et al., 1992; Sheth et al., 1985; Alonso et al., 1990).

The first approach clearly limits the scope of applicability of the resulting model, as it relies on an assumptions that do not normally hold in many operational scenarios: the utilization of the network layer has, in fact, a huge impact on real-world DTPs deployments (Padhye et al., 1998; Menasce & Almeida, 2001), and, thus, queueing effects cannot be overlooked. On the other hand, binding a performance model to the availability of a detailed network layer characterization results into the twofold drawback of *i*) precluding its applicability in virtualized environments, where little or no knowledge at all is available about the underlying physical infrastructure and *ii*) hindering its portability to heterogeneous deployment platforms.

The DTPs performance models proposed in this dissertation overcome such limitations by embracing the *Divide et Impera* gray box modeling approach. It allows them to accurately model the performance of the network over which the DTP is deployed, while overcoming the observability limitations posed by virtualization in Cloud environments: contention on CPU, and the effect of replication protocols and concurrency control algorithms are analytically modeled by means of Queueing Theory; performance of network-bound operations, instead, are predicted by means of a ML-based statistical model.

This dichotomy, as hinted, allows the proposed models to be applied to accurately predict performance of applications deployed over virtualized infrastructures, where little or no knowledge is available about the underlying hosting platform. At the same time, as the black box network model is unaware of the actual details of the hosting infrastructure, this gray box modeling approach allows the proposed models to be seamlessly employed in the management of DTPs deployed over different infrastructures.

To conclude this section, Table 3.3 summarizes the conducted analysis of the state-of-the-art, by reporting the main shortcomings affecting related works and highlighting how the proposed DTPs analytical models and gray box modeling techniques overcome such limitations.

Aspect	Limitations	Proposed Solution	Solution's effect
Modeling methodology	White box modeling relies on assumptions and approximations. In addition, it is cumbersome to apply in virtualized environments	Gray Box Modeling	The *Divide et impera* approach allows for treating unobservable/partially observable components as black boxes, and combines the resulting model with a white box one, which targets other aspects of the target system. The *Hybrid ensembling* approach, instead, allows a white box model to enhance its accuracy by incorporating factual knowledge collected from the operational system.
	Black box modeling suffers from curse of dimensionality		By reducing the extent of the problem targeted by means of ML, the *Divide et impera* approach reduces training time with respect to a purely black box approach. The *Hybrid ensemble* method, instead, can leverage the availability of a base white box model, which includes explicit domain knowledge. This translates into needing less training data to reach a given accuracy degree than a pure black box approach.
What-if analysis support	Only on-line optimization and no what-if analysis/off-line prediction capabilities	Proposed performance models based on *Divide et imperaa* (and the *Hybrid ensemble*-based ones developed on top of them)	Explicitly predict the performance of an application depending on workload and characteristics of the underlying DTP/hosting infrastructure, thus providing what-if analysis support.
Data access pattern	Uniform	Proposed performance models: ACF	Approximate non-uniform data access patterns by means of an equivalent uniform one.
	Non-uniform, but with access distribution known *a priori* via off-line profiling.		Can be computed on-line starting from statistics that are easy to collect and process.
Transaction unawareness	Lack of concurrency/replication protocol modeling	Proposed performance models	They explicitly model different concurrency/replication protocols.
Data replication	Full or no replication	Proposed performance models	It encompasses arbitrary replication degrees between no and full replication
Locality in partial replication	Fixed locality	Proposed performance models	It encompasses locality shifts given by the change of the scale or of the replication degree.
	Scale-sensitive locality only at the granularity of a whole transaction		Captures locality at the level of the single data access.
Network modeling	Fixed delay or no queueing effects	Divide et Impera	Network-bound interactions are modeled as a black box. This allows for modeling complex dynamics without having detailed knowledge about the hosting infrastructure; it also enable seamless portability of the proposed models to different infrastructures.

Table 3.3: Main limitations affecting existing performance models for DTPs and corresponding solutions proposed in this dissertation.

4 The Divide et Impera Approach

This chapter describes the *Divide et impera* gray box modeling technique, which consists in capturing the target performance function by means of several sub-models, such that each of them embodies the more convenient modeling paradigm (i.e., white box or black box).

In particular, this chapter introduces this novel gray box performance modeling methodology and describes its implementation to predict the performance of DTPs. The models presented in this chapter cover a wide spectrum of design choices among the ones introduced in Section 1. Specifically, they are intended to predict the performance of DTPs that employ a multi or single-master replication protocol and that rely either on an encounter-time locking scheme in full replication, or on a commit-time locking scheme in partial replication.

The novelty in these models does not lie solely into the implementation of the *Divide et impera* modeling technique. The analytical models at their basis, in fact, contribute by themselves to advance the state of the art in analytical modeling of DTPs, *i*) by providing analytical formulations of combinations of concurrency control algorithm/replication protocols that have never been discussed in literature and *ii*) by introducing the *Application Contention Factor* (ACF), namely a novel technique to model the data access pattern of transactional applications.

It is also noteworthy to point out that the *Divide et impera* hybrid modeling technique has also been implemented in a simulation framework (Di Sanzo et al., 2015) targeting the performance prediction of DTP applications. The discussion of this specific solution is out of the scope of this dissertation; however, the adoption of the *Divide et impera* methodology with an alternative white box modeling technique showcases its effectiveness and flexibility.

The remainder of this chapter is structured as follows: Section 4.1 presents the design of the *Divide et impera* technique; Section 4.2 describe how it is applied to derive performance models

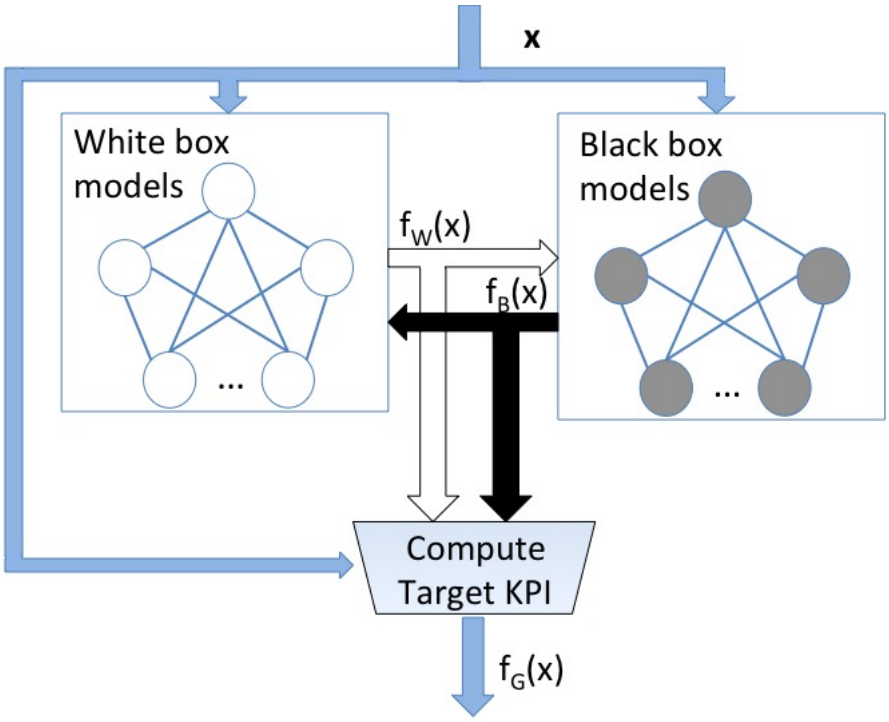


Figure 4.1: Design of the *Divide et impera* performance modeling technique.

for DTPs; finally, Section 5.5 concludes the chapter.

4.1 The Divide et impera performance modeling technique

This section introduces the *Divide et impera* performance modeling technique: specifically, Section 4.1.1 describes its design, and Section 4.1.2 provides an overview of how it is applied to build performance models of DTPs.

4.1.1 Design of the *Divide et impera* performance modeling technique

The *Divide et impera* performance modeling technique consists in decomposing the problem of predicting a target performance function into several, possibly inter-related, modeling sub-tasks, each corresponding to a specific aspect/component of the target system. These sub-tasks are tackled by means of the modeling technique that fits better the characteristics of the corresponding prediction problem.

The design of the *Divide et impera* technique is depicted in Figure 4.1: white box and black box models receive as input a vector $\mathbf{x}$, corresponding to the workload characterization and the target system's configuration parameters for which the model has to provide a performance prediction. A white box model can take as input the output of another white box one, and the same can happen for black box models. This kind of input dependency is depicted in Figure 4.1 by means of thin blue lines connecting "homogeneous" predictors, i.e., predictors based on the same modeling paradigm. Moreover, also "heterogeneous" models can mutually provide inputs to each other. This kind of relation is depicted by means of thick arrows, which correspond to the output of the white box models (noted $f_W(\mathbf{x})$) and black box ones (noted $f_B(\mathbf{x})$)

As already discussed in Section 3.1.3, the possibility to combine models with arbitrary input dependencies is a characteristic that distinguishes *Divide et impera* from other gray box performance modeling approaches, like the *Parameter Estimation* one in which black box models are only allowed to provide input to white box ones.

The outputs of the different white box and black box models, other than being provided as input to each other, are finally combined together to produce the target KPI of the system as a whole: in the proposed design, the component responsible for this task can be based on either modeling paradigm. Such component is also in charge of determining whether the outputs of the sub-models are "stable": in fact, as already mentioned in Section 3.1.1 regarding white box modeling, it can happen that the overall performance model requires an iterative resolution scheme, or that the input relations among white box and black box sub-model form a cyclic graph of dependencies

The overall model resolution scheme and the solution to the possible dependencies loop issue are specific to the target model: the following section and Section 4.2.4 shall describe how they are implemented in the proposed DTP models.

4.1.2 Overview of DTPs *Divide et impera* performance modeling

To accurately model the performance of DTPs, the proposed *Divide et impera* models implements the dichotomy between white and black box modeling in the following way. Analytical

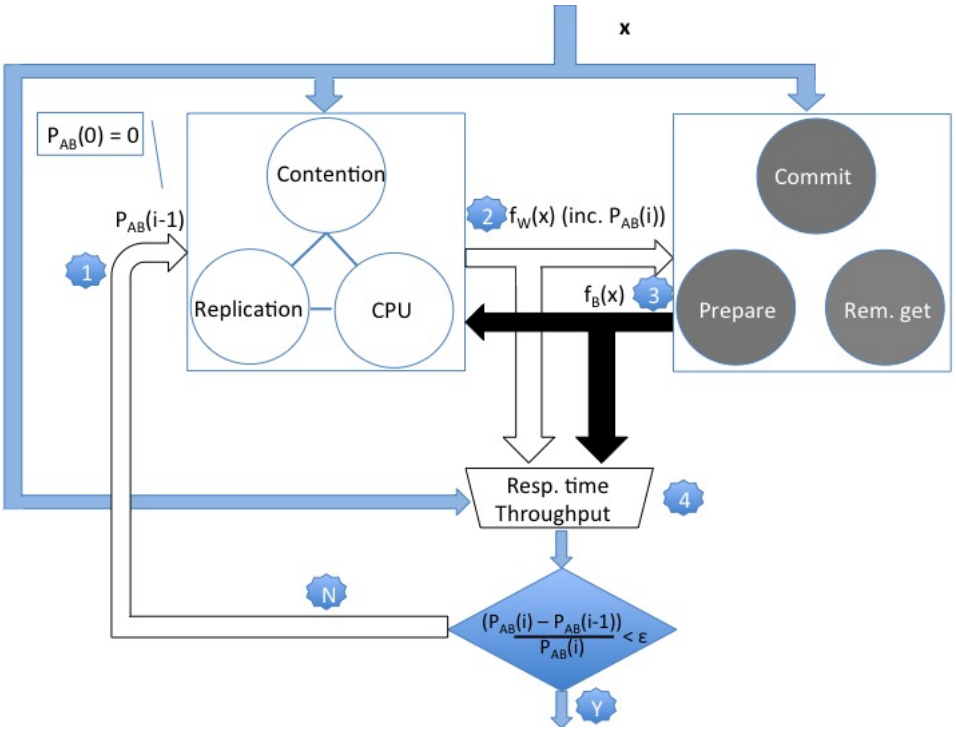


Figure 4.2: *Divide et impera* applied to DTPs performance modeling.

Modeling, based on Queueing Theory, is applied to model the interactions of the application's runtime with the CPU, and to predict how the concurrency control scheme and the enacted replication scheme (depending both on the employed protocol and replication degree) affect performance. This is made possible because the specifications of the employed concurrency control scheme and replication protocols are known, and because capturing CPU behavior time by means of typical white box abstractions has proven to yield a satisfactory level of accuracy (as shown by the experimental evaluation in Section 5.4).

On the other hand, the response time of operations that require distributed synchronization, e.g., remote gets and commits, is predicted by means of a black box model. The rationale behind this choice is that such operations are network-bound: as the proposed model targets (also) Cloud deployments, it cannot rely on detailed information about the technology and topology employed at the network layer to connect different nodes in the platform, or about service demands to process and send/receive network packets. Moreover, DTPs' transaction processing typically lies on top of a group communication toolkit — e.g. JGroups (Ban, 2012) in the Infinispan case — that provides several inter-process synchronization services (like failure detection,

group membership, remote procedure calls): the effect of the internal design and configuration of this layer on performance exacerbates the complexity of deriving a detailed white box model to predict the response time of network-bound operations.

Figure 4.2 portrays the organization of the different sub-models: *i*) white box ones are connected among themselves, meaning that they provide each others with inputs, whereas the black box ones are not, meaning that they are independent; *ii*) the model responsible for predicting the target KPIs (transactions response time and closed-system throughput) is a white box one; *iii*) white and black box models mutually provide some inputs to each other: this, as hinted in the previous section, gives rise to a loop of between the two counterparts. Specifically, white box models are used to predict the transactions' abort probability (noted P_{AB}); this is needed to compute some inputs for the black box networking models (e.g., the arrival rate of network-bound operations); at the same time, the output of the black box models is needed by the white box ones to compute P_{AB}, thus materializing the dependency loop.

Figure 4.2 depicts how these cyclic dependencies are broken, and the steps of the iterative resolution scheme of the overall model. The step 1 of an iteration of the resolution process starts by providing as input to the white box models a value for the abort probability P_{AB}: on iteration 0, this value is set to an arbitrary value, and specifically $P_{AB}(0) = 0$; on subsequent iterations, it corresponds to the output of the previous iteration. Such input value is used by the white box models to produce the inputs needed by the black box ones (step 2), which, in turn, provide back the predictions corresponding to the network-bound operations response times (step 3). These are needed by the white box models to compute an updated value of P_{AB}.

The whole set of outputs produced by white box and black box models is, then, fed as input to the white box model responsible for computing transactions' response time and system throughput (step 4). The values of the target KPIs are regarded as stable, and hence returned as result, only if the value of P_{AB} provided in input at the beginning of a resolution step is sufficiently close to the one produced at the end of the same step, i.e., if $\frac{|P_{AB}(i-1)-P_{AB}(i)}{P_{AB}(i)} < \varepsilon$, where $\varepsilon > 0$ is a threshold that can be used to tune the desired convergence accuracy. If this is not the case, then $P_{AB}(i)$ is used as input for the next iteration.

This resolution scheme consists in a fixed point recursion, a methodology that is employed

by many existing analytical models for transactional systems (Menascé & Nakanishi, 1982; Yu et al., 1993; Elnikety et al., 2009). Starting from an arbitrary value for a variable z, a fixed point recursive scheme searches for a value z^* such that $f(z^*) \simeq z^*$, by recursively evaluating the function f on the output provided by f at the previous iteration. In the case of the aforementioned resolution scheme, the function f coincides with P_{AB}. Details about the implementation and the convergence properties of this resolution scheme in the proposed DTPs models will be provided in Section 4.2.4.

4.2 Divide et impera performance models of DTPs

This section describes the application of the *Divide et impera* methodology to build performance models for DTPs embracing different design choices. Specifically, Section 4.2.1 provides a detailed description of the internals of Infinispan, namely the DTP employed to instantiate and validate the proposed models, and presents the system model; Section 4.2.2 and Section 4.2.3 present, respectively, the proposed white and black box models that constitute the building blocks of the proposed *Divide et impera*-based DTPs performance models; Section 4.2.4 describes the resolution scheme of the proposed models and provides details on how the white and black box components are reconciled to predict the overall performance of a DTP; Section 4.2.5 evaluates the accuracy of the proposed models as well as operational details related to their instantiation and employment in Infinispan, i.e., their querying time and workload characterization overhead; finally, Section 4.3 concludes the chapter.

4.2.1 System overview and model

This section provides some details about the concurrency control scheme and replication protocols that are targeted by the proposed DTPs models and that are integrated in Infinispan. This section is preparatory to the description of the performance models presented in the next sections, and introduces the system model, namely the representation of the system in the proposed analytical models, as well as core hypotheses and assumptions.

4.2.1.1 System overview

This section provides an overview of the combinations of concurrency control schemes and replication policies targeted by the proposed DTPs performance models.

4.2.1.1.1 Concurrency control scheme. The proposed performance models target DTPs that can run one of two concurrency control schemes: an Encounter-Time Locking (ETL) scheme, which, as hinted in Section 1.3, is supported only in full replication, and a Commit-Time Locking (CTL) one, which can be implemented both in full and partial replication.

ETL. In the ETL scheme, a transaction acquires the lock corresponding to a data item upon issuing a put operation against it. Employing this concurrency control scheme has two main ramifications. On one side, a transaction can detect conflicts on the node where it is executing even before reaching the distributed consensus phase, thus resulting into an eager conflict resolution policy. On the other side, locks are taken both during the local execution of a transaction's business logic and throughout the distributed commit phase, thus increasing the likelihood of conflicts.

Because of this prolonged lock hold times, the ETL scheme is not particularly suited to be integrated with a partial replication policy. In fact, in such a case, a transaction T running on a node ν could issue a put operation against a datum x which is not replicated on ν. As a result, T should communicate with the node that is responsible for locking x, incurring a network round-trip latency (plus some CPU processing costs). This would lead to even longer locks hold times, with a detrimental effect on achievable parallelism. Therefore, as already sketched in Section 1.3, the proposed models target DTPs that implement ETL only in conjunction with full replication: this combination allows T to acquire a local lock on x, without incurring the penalty of a costly distributed synchronization.

CTL. In the CTL scheme, locks acquisition is attempted by transactions only upon completing the local execution. This approach is more optimistic than the ETL counterpart: on one hand, this may lead to increased scalability; on the other, it can lead to worse performance in high

contention scenarios, because of late conflicts detection.

Given that locks are acquired only at commit time, the CTL scheme is not affected by the aforementioned limitations in terms of prolonged lock hold times. Therefore, the proposed models target DTPs which implement the CTL scheme both in full and partial replication.

In partial replication, replicas of data items are mapped to nodes of the DTP by means of a deterministic consistent hash function (Karger et al., 1997); this design choice is motivated by the fact that hash functions have been shown to be extremely effective to uniformly spread most accessed data across nodes in a system, allowing for a better load balancing (DeCandia et al., 2007; Karger et al., 1997; Stoica et al., 2001). Moreover, given its deterministic nature, such hash function allows any node to compute the owners of a datum without the need of any kind of dedicated directory services (You et al., 2011). Finally, the exploitation of a deterministic hash function allows for minimizing the number of data redistributed in the platform upon joins/leaves of nodes.

The consistent hash function is also used to assign a *primary owner* node with each data item: if a transaction wants to access a datum x that is not replicated on the node where it is being executed, then it contacts the node that is primary owner for x, in order to retrieve it and store it in its private context. Note that this retrieval is only private to the transaction that issues it, i.e., the node where such transaction is executed does not become a replica of the datum: replicas of data are only determined by the designated hash function.

In addition, CTL in partial replication also requires the definition of a designated *lock owner* node for every data item. Such a node is responsible for acquiring the lock corresponding to a written data item and, thus, regulating its concurrent manipulation. Lock ownership is implemented in different ways depending on the employed replication protocol; additional details, therefore, are going to be provided in the corresponding following sections.

As a final note, it is noteworthy to mention that the encompassed CTL variant guarantees a weaker isolation level than its ETL counterpart, as it suffers, for example, from the *update loss* anomaly (Bernstein, 1986). Suppose the value of element x is 10: a transaction T1 can tentatively set $x = 2x$; concurrently, a transaction T2 can tentatively set $x = 3x$. If T2 completes before T1 issues the commit, then T1's outcome will overwrite T2's one, and vice-versa, result-

ing into having $x = 20$ in the first case and $x = 30$ in the second. Such behavior is not allowed in ETL, as the tentative modification of x would be guarded by a both a local and a remote lock acquisition, resulting in at least one between T1 and T2 to abort, depending on the operations interleaving.

4.2.1.1.2 Replication Protocol. The proposed performance models target both multi and single-master replication protocols, and in particular Two-Phase Commit (2PC) and Passive Replication (PR).

Regardless of the locking scheme and replication protocol combination, the timeout on deadlock detection is set to 0, which means that transactions abort immediately upon detecting a conflict on a lock. This solution is a typical approach to achieve deadlock freedom, and is often preferred over more complex distributed deadlock detection schemes because of its simplicity and practicality (Agrawal et al., 1987).

2PC. In the 2PC replication protocol, each node in the DTP can serve both update and read-only transactions. This has the potential for allowing the DTP to withstand both read and update-intensive workloads, but require a (costly) distributed consensus phase to manage conflicts arising from transactions executing at different nodes.

2PC with ETL. In ETL, during the first phase of the distributed consensus protocol (also called *prepare* phase), locks acquisition is attempted at all the remote nodes, in order to detect conflicts with transactions concurrently executing on other nodes, as well as for guaranteeing transaction atomicity. If the lock acquisition phase is successful on all nodes, the transaction originator broadcasts a commit message, in order to apply the transaction's modifications on the remote nodes, and then it commits locally. Otherwise, it broadcasts a rollback message in order to induce the release of locks potentially held on remote nodes.

2PC with CTL. In CTL, during the prepare phase, a transaction first tries to acquire locally all the locks whose the local node is owner. Lock owners are mapped to nodes in the DTP by means of the same hash function that is used to determine primary owner replicas. If this step

succeeds, then the remaining locks are acquired on the corresponding owners. The commit phase is completed, depending on the outcome of this remote lock acquisition phase, in the same guise as in the ETL case.

Note that, although locks are acquired only on the lock owner nodes, all the nodes that store a replica of written data items are contacted at prepare-time. This allows for the implementation of schemes aimed at enabling the recoverability of a transaction in case of a failure of lock owners during the prepare phase.

PR. In the PR replication protocol, a single node, referred to as *primary* or *master*, is entitled to serve update transactions, and update transactions only; other nodes, referred to as *backups* or *slaves*, can only process read-only transactions (Budhiraja et al., 1993). As already discussed, this replication protocol simplifies conflicts detection and resolution, but may result into degraded performance in case of update-intensive workloads.

PR with ETL. In ETL, which is supported only in full replication, locks are only acquired on the master node, which serves all update transactions; therefore, the local concurrency control algorithm suffices to guarantee the isolation property of the transactions, and it does not require any distributed coordination at commit time. The commit phase, in fact, only consists in broadcasting to the backups the modifications performed locally; then, in order to enforce global consistency, the primary waits until all the replicas have modified their state before notifying to the application the successful commitment of the transaction.

PR with CTL. In this combination, unlike the 2PC case, the master node is designated as lock owner for all the data items. This allows it to regulate concurrency locally; the propagation of modifications to slave nodes is performed as in the ETL case, by contacting all the nodes that replicate at least one written datum.

4.2.1.2 System model

The proposed white box model relies on Average Value Approximation (Tay, 2013) to forecast the probability of transactions commit, the mean transactions duration, and achievable throughput. The aim of the model is to support what-if analysis on the application's performance when changing parameters such as the scale (number of nodes and possibly number of threads) and the replication degree, or shifts of workload characteristics, such as changes of the transactions' data access patterns or of the transactional mix.

The model considers the number of nodes in the system (noted N), the number of threads processing transactions at each node (θ) and the replication degree of data items (r) as input parameters. For the sake of simplicity, the nodes are assumed to be homogeneous in terms of computational power and RAM, and the model distinguishes only two classes of transactions, namely read-only and update transactions. As the model targets in-memory data stores, it assumes that the data set maintained at each node fits fully in RAM, and does not encompass interactions with persistent storage systems.

The system is modeled as open: transactions arrive according to a Poisson process with rate λ_{tx}. The percentage of update transactions is noted $\%w$; transactions belonging to this class perform, on average, N_g^{UP} read and N_w update operations on distinct data items; read-only transactions, which compose the other $1 - \%w$ of the transactional mix, read, on average, N_g^{RO} distinct data items. In 2PC, incoming transactions are evenly distributed across all the nodes; in PR, instead, the master processes only the update transactions, which account for a fraction $\%w$ of the overall load, and the remainder, namely read-only transactions, are evenly distributed across slave nodes.

The model relies on two sets of assumptions. The first concerns the independence of data accesses and re-executions of transactions. More precisely, it assumes that each read or written datum is chosen independently from each other; also, once a transaction is restarted upon experiencing an abort, it is indistinguishable from a transaction that joins the system for the first time.

The second kind of assumption concerns the mapping of data replicas onto nodes of the

system, and the data locality exhibited by transactions. The model relies on the definition of a primary owner function $P_O(N,r)$ that, given the number of nodes in the system and the replication degree, determines what is the probability that a transaction originated on a node ν accesses a datum for which ν is primary owner. The $r-1$ non-primary replicas of a datum are assumed to be scattered across the system uniformly at random. Hence, assuming that node ν is not the primary replica for a data item x, the probability that ν is a non-primary owner replica of x is $l = \frac{r-1}{N-1}$. Thus, the probability that a transaction originated on node ν accesses a datum stored by ν is:

$$\mathscr{L}(N,r) = P_O(N,r) + (1 - P_O(N,r))l.$$

To simplify notation, the arguments N and r will be omitted throughout the discussion, i.e., the aforementioned probabilities will be noted, respectively, P_O and $\mathscr{L}$. The model further assumes that whenever a transaction originated on a node ν accesses a datum x for which ν is not primary owner, any other node has the same probability $\frac{1}{N-1}$ of being primary owner of x.

Moreover, the model relies on the definition of the probability (noted L_O) that the local node of a transaction is the lock owner for a specific datum. In the 2PC protocol, for any data item, if a node is primary owner of a datum it is also lock owner for that datum, and vice-versa; In PR, instead, lock and data ownership are independent, as the primary node is owner of all the locks, but not of all the data.

Although the model encompasses the possibility for some data items to be more frequently accessed than others, it assumes that data hot spots are evenly distributed across nodes of the system. Note that this assumption is not unrealistic, as load balancing and data hot-spot avoidance are among the key advantages of the data distribution policies (e.g., consistent hashing (Karger et al., 1997)) that are typically employed by modern distributed NoSQL stores (DeCandia et al., 2007; Lakshman & Malik, 2010; Marchioni & Surtani, 2012). Overall, this assumption, together with the one on the uniformity of the transactional arrival rate, allows for considering all nodes as evenly loaded (except for master and slaves in PR).

Finally, the model assumes the system to be stable and ergodic: this means that all the parameters are defined to be either long-run averages or steady-state quantities; also, this implies that the transactions arrival rate does not exceed the service rate and equals the commit rate.

4.2.2 White box modeling in the proposed DTP models

This section describes the equations that are at the basis of the proposed white box models. The derivation of such equations is provided in a top-down fashion: first, the transactions' response time is obtained, assuming commit probabilities and execution time of single operations to be known; then, the data contention model is described, which explains how the model computes conflict probabilities; finally, the CPU model is presented, which provides the derivation of the single operations' response time starting from their service demands.

The proposed white box model encompasses different concurrency control schemes and replication protocols; therefore, each of the following subsections will provide, whenever necessary, specified set of equations depending on the operational mode being modeled (e.g., 2PC vs PR or ETL vs CTL). Table 4.1 reports the inputs of the model.

4.2.2.1 Transactions' response time computation

This section describes how the model computes transactions' response time for read-only and update transactions.

4.2.2.1.1 Read only transactions response time. In the target consistency level, which is the one guaranteed, for instance, by Infinispan, read-only transactions never abort: as a result, the equations modeling their response time are invariant with respect to the concurrency control scheme and replication protocol. Their response time, noted R_L^{RO}, is simply determined as the sum of the time spent to i) initialize the transaction (R^{beg}), ii) perform read operations (R^g), iii) execute the business logic of the transaction R_B^{RO}, and iv) commit (R_{com}^{RO}):

$$R_L^{RO} = R^{beg} + R_B^{RO} + N_g^{RO} R^g + R_{com}^{RO}.$$

R^g is computed as the average cost of performing a local read (R_L^g), and a remote one (R_R^g):

$$R^g = \mathscr{L} R_L^g + (1 - \mathscr{L}) R_R^g.$$

Class of parameter	Notation	Meaning
System	N	Number of nodes in the DTP
	r	Replication degree
	θ	# of threads per node serving transactions (only in closed system)
	ACF	Application Contention Factor
	P_O	Probability of the local node being primary owner for a datum
	L_O	Probability of the local node being lock owner for a datum
Workload	λ_{tx}	Transactions arrival rate
	$w\%$	Update transactions percentage
	N_g^{UP}	Avg. # distinct gets issued by an update transaction
	N_w	Avg. # distinct puts issued by an update transaction
	N_g^{RO}	Avg # distinct gets issued by a read-only transaction
CPU demands	D_{beg}	Begin of a transaction
	D_B^{RO}	Business logic of a read-only transaction
	D_B^{UP}	Business logic of an update transaction
	D_L^g	Local get operation
	D^P	Put operation
	D_L^γ	Retrieval of a remote datum
	D_R^γ	Remote get (on a remote node)
	D_{com}^{RO}	Commit of a read-only transaction
	D_{com}^{UP}	Commit of an update transaction
	D^{wb}	Write-back of modifications upon commit
	D_L^{prep}	Local prepare of a transaction
	D_L^{roll}	Rollback of a local transaction that aborts locally
	D_{LR}^{roll}	Rollback of a local transaction that aborts remotely
	D_R^{roll}	Rollback of a remote transaction

Table 4.1: Input parameters (divided by class) of the proposed performance models for DTPs.

On its turn, the cost for a remote get is expressed as the sum of a local computation (R_L^g) and the latency for retrieving the remote datum (R_R^γ):

$$R_R^g = R_L^g + R_R^\gamma.$$

4.2.2.1.2 Update transactions response time. Unlike read-only transactions, update transactions can abort while acquiring locks, either while executing the business logic (in ETL) or during the final validation phase (both in ETL and CTL). The response time of an update transaction (R^{UP}) is, thus, given by the sum of the response time of a successful execution plus the time spent in previous aborted executions, i.e.:

$$R^{UP} = N_{AR}(R_{AR}^{UP} + N_{AL}R_{AL}^{UP}) + N_{AL}R_{AL}^{UP} + R_C^{UP}.$$

In this expression, R_{AR}^{UP}, resp. R_{AL}^{UP}, denotes the response time of a transaction that aborts because of an abort that materializes on a remote node, resp. on the local node; N_{AR}, resp. N_{AL}, refers to the number of average re-executions of a transaction due to conflicts arising on a remote, resp. on the local, node; finally, R_C^{UP} is the response time of a successfully committed execution.

R_C^{UP} is computed, on its turn, as the sum of the local response time of the transaction (R_L^{UP}) plus the execution time of *i*) the local (R_L^{prep}) and global validation of the prepare phase (R^{prep}), *ii*) the write-back of updated data items (R^{wb}), and *iii*) the notification to remote nodes about the outcome of the transaction R^{dec}:

$$R_C^{UP} = R_L^{UP} + R_L^{prep} + R^{prep} + R^{wb} + R^{dec}.$$

The local execution time of a transaction, R_L^{UP}, consists of the time needed to execute the business logic of the transaction itself, plus the time spent to perform write (N_w on average) and read operations (N_R on average). Noting R^p the response time of a put operation, R_L^{UP} is computed as

$$R_L^{UP} = R^{beg} + R_B^{UP} + N_g^{UP} R^g + N_w R^p.$$

Note that a put operation does not require any inter-node synchronization: this is descended by the fact that, as previously discussed, the model targets only concurrency control schemes in which a lock is acquired at commit time only on its primary node (CTL) or in which a lock can always be acquired locally (ETL in full replication).

The response time of an aborted run that fails while acquiring a remote lock is obtained as

$$R_{AR}^{UP} = R_L^{UP} + R_L^{prep} + R^{prep} + R_L^{roll} + R^{dec},$$

where R_L^{roll} denotes the execution time of the rollback operations performed locally upon aborting a transaction.

The following discussion explains how the model computes R_{AL}^{UP}, namely the execution time of locally aborted runs of update transactions. Its analytical expression, in fact, depends on whether the DTP is configured to employ ETL or CTL: in the first case, a transaction can

abort before reaching the distributed commit phase, as locks are acquired upon issuing a put operation; in the second case, instead, conflict detection is deferred to the prepare phase.

R_{AL}^{UP} in ETL. The computation of R_{AL}^{UP} in ETL is the same for the 2PC and the PR case, as the local part of an update transaction is executed in the same fashion, regardless the employed replication protocol. In ETL, an update transaction can incur a lock conflict, and, hence, an abort, while trying to acquire the $i-$th lock, among the N_w ones that it acquires on average. Write operations are modeled as uniformly spread across the local execution of a transaction, i.e., the $i-$th put operation happens after a period corresponding to $\frac{iR_B^{UP}}{N_w}$. Therefore, noting $P(AL = i|AL)$ the probability for a transaction of incurring an abort at put operation i, conditioned to the event that the transaction is going to abort in the current run, R_{AL}^{UP} is computed as

$$R_{AL}^{UP} = \big[\sum_{i=1}^{N_w}(\frac{R_B^{UP}}{N_w} \cdot i) \cdot P(AL = i|AL)\big] + R_L^{roll}.$$

R_{AL}^{UP} in CTL. When operating according to the CTL mode, update transactions cannot incur aborts while executing locally. The only case in which an update transaction can abort before undergoing the global validation phase is if the node on which it is executed is primary owner for some of the updated objects and the corresponding locks are concurrently being held. Hence, the execution time of a locally aborted run of an update transaction in CTL is computed as

$$R_{AL}^{UP} = R_L^{UP} + R_L^{prep} + R_L^{roll}.$$

4.2.2.2 Contention model

This section presents the contention model, i.e., the set of equations that compute abort probabilities and expected number of re-runs for update transactions. First, the general locking model is presented; then it is applied to the specific DTP operational modes.

4.2.2.2.1 Locking model. Each lock is modeled as an independent $M/G/1$ server: a lock is acquired at a rate λ_{lock} and the average service time to complete a request, T_h, is the lock hold

time, namely the time since the lock acquisition and until its release (either caused by a commit or by an abort). Note that, in general, a transaction can hold more than one lock at a single time and, as described later, hold times for locks taken by the same transaction are correlated: such characteristics violate the independence assumption for the aforementioned server. The model, nevertheless, relies on this simplifying assumption for the sake of analytical tractability. By exploiting this assumption, the probability that a transaction incurs contention when trying to acquire a lock is computed as the utilization of the corresponding server. Such metric is defined as the fraction of time that a server is busy serving a request (Kleinrock, 1975) and, in the model, it is computed as

$$U = \lambda_{lock} T_h \tag{4.1}$$

(assuming $\lambda_{lock} \cdot T_h \leq 1$).

On its turn, λ_{lock} depends on the application's data access pattern: if every datum were equally likely to be accessed in a data-set of cardinality D, then, noting Λ_{lock} the total locks acquisition rate, λ_{lock} could be computed as $\lambda_{lock} = \frac{\Lambda_{lock}}{D}$. However, this uniformity assumption does not hold in general, as data access distributions are often skewed. Unfortunately, a full characterization of an application's data access pattern requires an extensive and costly profiling phase, typically performed off-line (Mozafari et al., 2013). In order to address this issue, the model relies on the abstraction of the Application Contention Factor (ACF) (Didona et al., 2014), a metric that allows for succinctly characterizing the skew of a data access pattern. The ACF is a scalar value that is inferred from the application's behavior, with minimal profiling, and represents the inverse of the cardinality of an equivalent dataset that, under a uniform data access pattern, would yield to the measured contention probability. Namely, noting P_{lock} the lock contention probability:

$$ACF = \frac{P_{lock}}{\Lambda_{lock} T_h}. \tag{4.2}$$

Section 4.2.5.2 will show that, under the hypotheses described in Sec. 4.2.1.2, the ACF can be considered an invariant of the application's workload. Therefore, it can be exploited by the model to speculate about the performance of the application when deployed over a different number of nodes, or when changing the replication protocol/degree in the platform. Moreover, given that its computation relies only on the profiling of average values, it is very lightweight to

compute on-line (Ciciani et al., 2012; Palmieri et al., 2012), thus allowing the adoption of the proposed model also in presence of time varying data access patterns.

4.2.2.2.2 Conflict probabilities. This paragraph is devoted at providing specializations of Equation 4.1 depending on the operational mode implemented by the DTP.

ETL. In ETL, update transactions acquire locks both during the local execution and the distributed commit phase, on each node (remember that this variant is only encompassed by the model in full replication). Therefore, the lock hold time depends on the index $i \in 1 \dots N_w$ of the corresponding put operation. It is clear that a lock acquired at operation j (noted L_j) has a higher lock hold time than a lock acquired at operation $k > j$ (L_k); as a result, without taking into account items popularity, it is more likely, for another transaction, to conflict on L_j rather than on L_k (Yu et al., 1993).

Therefore, the ETL model obtains the mean lock hold time of a transaction starting from the sum of the hold times of the locks acquired during its local execution (regardless of its final outcome). This sum is referred to as cumulative hold time and it is defined, for a transaction that acquires i locks, as

$$G(i) = \sum_{j=1}^{i} \frac{jR_B^{UP}}{N_w}.$$

The following sections provide specialization of the ETL contention model for the 2PC and PR cases, which both rely on Equation 4.2.2.2.2 for the cumulative lock hold time duration.

2PC. Update transactions running according to the 2PC replication protocol in ETL can experience lock conflict both during the local execution and the distributed commit phase. Therefore, the model encompasses two different contention probabilities: $P_{L,lock}$, for the former case, and $P_{R,lock}$ for the latter one.

In addition, in both cases, a transaction can contend locks both with co-local and remote transactions. In the model, the events of conflicting with a remote or a local transactions are disjoint: in fact, only one transaction can hold a lock at any given time and transactions immediately abort upon trying to acquire a busy lock (i.e., there are no queued transactions waiting

for the lock release). For this reason, the two aforementioned lock conflict probabilities are expressed as the sum of the probabilities of the single events of aborting because of a local or a remote transaction:

$$P_{L,lock} = P_{LL,lock} + P_{LR,lock}, \qquad P_{R,lock} = P_{RL,lock} + P_{RR,lock}.$$

The model further decomposes conflict probabilities, distinguishing between conflicts that arise with committing and aborting transactions (either remotely or locally) and computes the aforementioned probabilities as sum of the single sub-contributes.

For the local conflict, in particular, the conflict probability equations are the following:

$$P_{LL,lock} = ACF(\lambda^{C}_{L,lock} T^{C}_{Lh} + \lambda^{AL}_{L,lock} T^{AL}_{Lh} + \lambda^{AR}_{L,lock} T^{AR}_{Lh}),$$

$$P_{LR,lock} = ACF(\lambda^{C}_{R,lock} T^{C}_{Rh} + \lambda^{AR}_{R,lock} T^{AR}_{Rh}).$$

These probabilities take into account that, on the local node, a transaction T on node v can conflict with co-local transactions that *i*) commit (which have lock request rate equal to $\lambda^{C}_{L,lock}$ and lock hold time T^{C}_{Lh}) or *ii*) abort locally (which have lock request rate equal to $\lambda^{AL}_{L,lock}$ and lock hold time T^{AL}_{Lh} or *iii*) abort while being validated on a remote node (which have lock request rate equal to $\lambda^{AR}_{L,lock}$ and lock hold time T^{AR}_{Lh}); similarly, T can experience a conflict with remote transactions being validated on v that *i*) commit (whose lock request rate is $\lambda^{C}_{R,lock}$ and lock hold time T^{C}_{Lh}) or *ii*) abort because of a conflict arisen on a node $v' \neq v$ (whose lock request rate is $\lambda^{AR}_{R,lock}$ and lock hold time T^{AR}_{Rh}). Note that the model does not encompass conflicts between T and remote transactions that abort on v itself: in fact, during the remote validation, locks are acquired sequentially, without any business logic computation. Therefore, the lock hold time on v for remote transactions that abort on v itself is considered negligible.

The lock acquisition rates of the aforementioned five classes of transactions are expressed as the product of the corresponding transactions arrival rate and the expected number of acquired locks. Noting N_{AR} the expected number of re-runs due to remote aborts and N_{AL} the average number of re-runs needed to a transaction to reach the distributed commit phase, the model

computes the locks acquisition rates as follows:

$$\lambda^{C}_{L,lock} = \lambda^{UP}_{Ltx} N^{C}_{L,lock}, \qquad \lambda^{AL}_{L,lock} = N_{AL}(N_{AR}+1)\lambda^{UP}_{Ltx} N^{AL}_{L,lock}, \qquad \lambda^{AR}_{L,lock} = N_{AR}\lambda^{UP}_{Ltx} N^{AR}_{L,lock}$$

$$\lambda^{C}_{R,lock} = \lambda^{UP}_{Rtx} N^{C}_{R,lock}, \qquad \lambda^{AR}_{R,lock} = N_{AR}\lambda^{UP}_{Rtx} N^{AR}_{R,lock}.$$

On their turn, the arrival rate of update transactions to local, resp. remote, nodes are computed as

$$\lambda^{UP}_{Ltx} = \frac{\lambda\,\%w}{N}, \qquad \lambda^{UP}_{Rtx} = \lambda^{UP}_{Ltx}(N-1).$$

The expected number of locks acquired by transactions are, instead, obtained as follows.

Four out of the five classes of transactions considered so far acquire N_w locks on a node v where they are executed: committing transactions (either local or remote), local transactions that abort during the remote validation or remote transactions that are successfully validated on a node v' but abort on another node v'', i.e.,

$$N^{C}_{L,lock} = N^{AR}_{L,lock} = N^{C}_{R,lock} = N^{AR}_{R,lock} = N_w.$$

The number of locks acquired by locally aborting transactions, instead, depends on how much a transaction has progressed before experiencing a lock conflict, namely,

$$N^{AL}_{L,lock} = \sum_{i=2}^{N_w} (i-1) P(AL = i|AL).$$

The equations for P_{AL} and $P(AL = i|AL)$ are obtained by exploiting the independence between operations issued by transactions: the sequence of lock acquisitions is modeled as a Bernoulli process (Papoulis, 1991) of length N_w, with "lock successfully acquired" and "abort" as possible outcomes for each operation. Therefore, the abort probability is computed as the probability of the complement of the event "the transaction successfully acquires all the requested locks", i.e.,

$$P_{AL} = 1 - (1 - P_{L,lock})^{N_w}. \tag{4.3}$$

Therefore, by applying Bayes' theorem, the model computes $P(AL = i|AL)$ as

$$P(AL = i|AL) = \frac{P(AL = i \wedge AL)}{P(AL)} = \frac{P(AL = i)}{P(AL)} = \frac{P_{L,lock}(1 - P_{L,lock})^{i-1}}{P_{AL}}.$$

The lock hold times, instead, are computed in the following way

$$T^{C}_{Lh} = \frac{G(N_w)}{N_w} + R^{prep}, \qquad T^{C}_{Rh} = T^{AR}_{Rh} = R^{prep},$$

$$T^{AL}_{Lh} = \sum_{i=1}^{N_w} \frac{G(i-1)}{N_w} P(AL = i|AL), \qquad T^{AR}_{Lh} = T^{C}_{Lh}.$$

In particular, the lock hold time for remote transactions is obtained by approximating it with the time needed by the transaction's coordinator to perform the distributed prepare phase (R^{prep}).

The model follows a similar analysis for what concerns remote transactions. Specifically,

$$P_{RL,lock} = P_{LL,lock}, \qquad P_{RR,lock} = ACF(\lambda^{C}_{RR,lock} T^{C}_{Rh} + \lambda^{AR}_{RR,lock} T^{AR}_{Rh}).$$

Note that $P_{RL,lock}$ equals $P_{LL,lock}$ because a transaction T that joins a node ν can contend with transactions originated on ν regardless of whether T is a local or a remote transaction. Instead, the model needs to take into account that remote transactions originated on a node ν' cannot contend among themselves on a remote node ν, as they have successfully acquired locks on their local node and are, thus, non-conflicting. This means that they can contend with remote transactions only coming from $N-2$ nodes (i.e., all the nodes in the platform except ν and ν'). Therefore,

$$\lambda^{UP}_{RRtx} = \lambda^{UP}_{Ltx}(N-2)$$

and

$$\lambda^{C}_{RR,lock} = \lambda^{UP}_{RRtx} N^{C}_{R,lock}, \qquad \lambda^{AR}_{RR,lock} = N_{ARR} \lambda^{UP}_{RRtx} N^{A}_{R,lock}.$$

In the last equation, N_{ARR} is the expected number of times that a remote transaction successfully acquires all the locks on a node ν' but eventually aborts because of a conflict that has materialized on node ν''. The equation used to compute N_{ARR} will be provided later in this

section, once its generating probability will have been obtained. The equation corresponding to the number of acquired locks per remote transaction class, instead, can be already computed as

$$N^{C}_{R,lock} = N^{A}_{R,lock} = N_w.$$

It is now possible to obtain the expected number of retries N_{AL}, N_{AR} and N_{ARR}. By exploiting the assumptions that restarting transactions are indistinguishable from brand new ones, executions of transactions are modeled as Bernoulli processes. In particular, N_{AL} is obtained as follows:

$$N_{AL} = \frac{1 - P_{AL}}{P_{AL}}.$$

The computation of N_{AR} is, instead, more cumbersome, as the model must take into account that events corresponding to remote conflicts may occur for the same transaction on different nodes in parallel, i.e., are not statistically independent. In fact, if a transaction T remotely aborts due to conflict with transaction T' on a node v' then it is very likely that the same conflict arises also on other nodes.

In order to obtain an equation similar to the ones obtained for N_{AL}, which leverages the independence of the considered conflict events, the model resorts to the following approximation: the probability of T to incur remote conflict with T' coincides with the probability that such conflict arises on the node on which T' has been originated. Such probability coincides with $P_{RL,lock}$, which has been already obtained. In other words, the model computes the probability of a transaction T to abort remotely only as a function of the *unique* remote transactions in the system, and not of their several instantiations spawned on different nodes to perform system-wide validation.

Thanks to this approximation, the probability of a transaction to fail the remote prepare phase can be computed as follows:

$$P_{AR} = 1 - (1 - P_{RL,lock})^{(N-1)N_w} \tag{4.4}$$

By leveraging this approximation, the model also computes the probability needed to obtain

N_{ARR}, i.e., $P(N_{R,lock} = N_w|AR)$. This represents the probability that a transaction T successfully acquires all the locks on a specific remote node while being validated but fails on another one, given that T's validation remotely fails.

$$P(N_{R,lock} = N_w|AR) = \frac{P(N_{R,lock} = N_w \wedge AR)}{P_{AR}} \approx \frac{(1-P_{RL,lock})^{N_w}\left[1-(1-P_{RL,lock})^{(N-2)(N_w)}\right]}{1-(1-P_{RL,lock})^{(N-1)N_w}}. \quad (4.5)$$

Note that, because of the aforementioned simplification, the model overestimates the probability of acquiring all the locks on a given node. In fact, it is clear that a transaction T can abort with *any* transaction on a remote node v, and not just with the ones that are local to v. Instead, also in Equation 4.5, the model only considers conflicts between T and local transactions on v. This simplification is, again, introduced for the sake of analytical tractability.

Finally, it is possible to compute N_{AR} and N_{ARR}:

$$N_{AR} = \frac{1-P_{AR}}{P_{AR}}, \qquad N_{ARR} = \frac{1-P_{ARR}}{P_{ARR}}.$$

PR. In the PR replication protocol, concurrency among update transactions is regulated by the primary node, during the local execution phase. This implies that replicating the outcome of an update transaction cannot give rise to aborts due to concurrency on slave nodes. As a result the modeling of conflicts for the PR case can be obtained by performing simple modifications to the one already obtained for the 2PC one.

In particular, since a transaction T running on the master can experience conflicts only with collocated transactions, the local lock contention probability reduces to

$$P_{L,lock} = ACF(\lambda^C_{L,lock}T^C_{Lh} + \lambda^A_{L,lock}T^{AL}_{Lh})$$

and the remote one is zero.

Similarly to the 2PC case, the first part of this equation refers to conflict arising between T and other transactions that are going to commit; the second one is related with the probability of T to conflict with transactions that are going to abort. In particular, $\lambda^C_{L,lock}$ denotes the rate at which locks are acquired by committing transactions and T^C_{Lh} the mean lock hold

time for committing transactions; $\lambda^{A}_{L,lock}$ and T^{AL}_{Lh} are the respective counterparts for aborting transactions.

The equations corresponding to locks request rate and locks hold times coincide with the ones already obtained in the previous analysis of the 2PC model. The only modification pertains the equation corresponding to the arrival rate of local update transactions; in fact, in PR, the master node serves all the flow of incoming update transactions:

$$\lambda^{UP}_{Ltx} = \lambda_{tx}\%w.$$

CTL. In CTL, locks are not held during the local execution of the transaction and they are only acquired on the corresponding primary owner (in 2PC) or on the local node (in PR). As for the ETL case, the following discussion will first describe the model for CTL with the 2PC replication protocol, and then will show how to obtain the model for the PR case starting from equations already derived for 2PC.

2PC. In 2PC, local transactions running on v that write to data items whose v is not primary owner are not locally synchronized on the corresponding locks; therefore, unlike in the ETL case, local transactions can conflict among themselves on a remote node. As a result, there is no distinction between a lock requested on the local node or on a remote one. Therefore, a single lock contention probability, noted P_{lock} is computed in the model, and it is obtained the sum of the probability of contending with co-local and remote transactions, i.e.,

$$P_{lock} = P_{L,lock} + P_{R,lock} \tag{4.6}$$

Just like in the ETL case, instead, the model considers conflicts arising on a node v between a transaction T and any other transaction that successfully completes the lock acquisition phase on v. In fact, also in CTL, a transaction releases all the locks it holds on a node upon detecting a conflict on that node. Hence, the hold time of locks owned by an aborting transaction on node v up to the occurrence of the conflict on v itself is considered negligible.

For this reason, when computing the contention probability P_{lock} for a lock on node v, the model encompasses four kinds of transactions: *i*) local transactions that commit; *ii*) local transactions that abort during the remote validation phase; *iii*) remote transactions that commit; *iv*) remote transactions that successfully complete the locks acquisition phase on a node v, but abort on another node v'. Hence:

$$P_{L,lock} \approx P^C_{L,lock} + P^A_{L,lock} = N \cdot ACF(\lambda^C_{L,lock} T^C_{Lh} + \lambda^A_{L,lock} T^A_{Lh})$$

$$P_{R,lock} \approx P^C_{R,lock} + P^A_{R,lock} = N \cdot ACF(\lambda^C_{R,lock} T^C_{Rh} + \lambda^A_{R,lock} T^A_{Rh})$$

These last equations are specializations of Equation 4.6; however, here, the ACF is multiplied by the total number of nodes in the system. This is because, as detailed in the following discussion, the locks acquisition rates used in the previous equations refer to the node where the lock is requested. Given that, by hypothesis, each node stores the same amount of data and data hot spots are evenly spread across N nodes, the "per node" equivalent uniform dataset cardinality is $(N \cdot ACF)^{-1}$.

As in the ETL case, the locks acquisition rate on a node for each of the four classes is computed as the product of the transactions arrival rate λ for that class in the whole system and of the expected number of locks N_{lock} acquired by a transaction of that class on a specific node:

$$\lambda^C_{L,lock} = \lambda^{UP}_{Ltx} N^C_{L,lock}, \qquad \lambda^A_{L,lock} = N_{AR} \lambda^{UP}_{Ltx} N^A_{L,lock},$$

$$\lambda^C_{R,lock} = \lambda^{UP}_{Rtx} N^C_{R,lock}, \qquad \lambda^A_{R,lock} = N_{AR} \lambda^{UP}_{Rtx} N^A_{R,lock}.$$

In these equations λ^{UP}_{Ltx}, resp. λ^{UP}_{Rtx}, is the global arrival rate of local, resp. remote, update transactions, which can be computed as

$$\lambda^{UP}_{Ltx} = \frac{\lambda \%w}{N}, \qquad \lambda^{UP}_{Rtx} = (N-1)\lambda^{UP}_{Ltx}.$$

In order to compute the expected, per-class number of locks acquired on a node, the model first obtains the following set of probabilities: *i*) $P_{LL}(i)$, i.e., the probability that exactly i locks

are acquired on the local node of a transaction; *ii*) $P_A(i)$, i.e., the probability of aborting while trying to acquire i locks on a node; *iii*) P_{com}, i.e., the probability of successfully committing a transaction; *iv*) P_{AR}, i.e., the probability of successfully acquiring all the local locks and aborting during the two-phase commit. Exploiting the assumption that data are accessed independently, these are computed as follows:

$$P_{LL}(i) = \binom{N_w}{i} L_O^i (1-L_O)^{N_w-i}, \tag{4.7}$$

$$P_a(i) = 1-(1-P_{lock})^i, \qquad P_{com} = 1-P_a(N_w),$$

$$P_{AR} = \sum_{i=1}^{N_w} P_{LL}(i)(1-P_a(i))P_a(N_w-i).$$

The last probability has been computed as a weighted sum of the probability of requesting i local locks, acquiring all of them locally, and failing in acquiring at least one of the $N_w - i$ remote locks (with i ranging from 1 to N_w).

From these equations it is possible to obtain other intermediate probabilities that can be used to compute the expected number of acquired locks on a node per transaction type. For a local transaction T originated on node ν, the following probabilities are computed in the model *i*) $P(N_{L,lock} = i|C)$ the probability that T acquires i locks on ν, provided that it commits; *ii*) $P(N_{R,lock} = i|C)$ the probability that T acquires i locks on a remote node ν', provided that it commits; *iii*) $P(N_{L,lock} = i|AR)$ the probability that T acquires i locks on ν, provided that it remotely aborts; *iv*) $P(N_{R,lock} = i|AR)$ the probability that T successfully acquires all the i locks that it requires on a remote node ν', provided that it aborts remotely (i.e., on some other remote node). By using Bayes' theorem, these probabilities are computed as follows:

$$P(N_{L,lock} = i|C) = P_{LL}(i), \quad P(N_{R,lock} = i|C) = \sum_{j=0}^{N_w-i} P_L^\dagger(j,i,N_w)$$

$$P(N_{L,lock} = i|AR) = \frac{P_{LL}(i)(1-P_a(i))P_a(N_w-i)}{P_{AR}}$$

$$P(N_{R,lock} = i|AR) = \frac{\sum_{j=0}^{N_w-i-1} P_L^\dagger(j,i,N_w)P_A^\dagger(j,i,N_w)}{P_{AR}}. \tag{4.8}$$

In the last equations, $P_L^{\dagger}(i,j,k)$ denotes the probability that T acquires i locks locally, j locks on a node v' and $k-i-j$ locks on other remote nodes; $P_A^{\dagger}(i,j,k)$, instead, models the probability of successfully acquiring the i locally requested locks and the j locks requested on a remote node v', while failing to acquire the remaining $k-i-j$ locks on other remote nodes different from v'. Exploiting again the hypothesis of independent accesses to data, the former probability is computed starting from a multinomial distribution (Papoulis, 1991),

$$P_L^{\dagger}(i,j,k) = \frac{k!}{i!\,j!(k-i-j)!} L_O^i \left(\frac{1-L_O}{N-1}\right)^j \left(\frac{(1-L_O)(N-2)}{N-1}\right)^{k-i-j}$$

while the latter is obtained as

$$P_A^{\dagger}(i,j,k) = (1-P_a(i+j))P_a(k-i-j).$$

From these sets of probabilities the expected number of locks acquired on a node by the different transaction kinds can be finally obtained:

$$N_{L,lock}^C = \sum_{i=1}^{N_w} iP(N_{L,lock}=i|C), \;\; N_{L,lock}^A = \sum_{i=1}^{N_w} iP(N_{L,lock}=i|AR),$$

$$N_{R,lock}^C = \sum_{i=1}^{N_w} iP(N_{R,lock}=i|C), \;\; N_{R,lock}^A = \sum_{i=1}^{N_w-1} iP(N_{R,lock}=i|AR).$$

Next, lock hold times are obtained. Local transactions hold locks during the whole prepare phase and release them before sending the final commit/rollback message. As already mentioned, the transaction coordinator waits for all the cohorts' replies, regardless of the outcome of the distributed commit phase. Remote transactions, on the other hand, hold locks for the time necessary to send back to the coordinator their vote and to receive the final commit/rollback message. Also the ETL model, in order to simplify the analysis, adopts the simplification of considering this last latency comparable to the latency experienced by the coordinator to complete the prepare phase. Moreover, it considers negligible the impact that the lock acquisition/release phase has on hold time (as compared to distributed the commit latency). Thus:

$$T_{L,H}^C = T_{L,H}^A = T_{R,H}^C = T_{R,H}^A = R^{prep}.$$

Finally, it is possible to compute the expected number of aborts experienced locally and remotely by a transaction, noted, respectively, N_{AL} and N_{AR}. By leveraging on the hypothesis that a retrying transaction is indistinguishable from a transaction that enters the system for the first time,

$$N_{AL} = \frac{P_{AL}}{1 - P_{AL}}, \quad N_{AR} = \frac{P_{AR}}{1 - P_{AR}},$$

with

$$P_{AL} = \sum_{i=1}^{N_w} P_{LL}(i) P_a(i).$$

PR. As for the ETL case, also the CTL-PR model can be obtained by modifying the equations already provided for the contention model of the 2PC replication protocol. In particular, like in the previous case, the equation corresponding to lock contention only encompasses conflicts arising between local transactions; moreover, since there model considers negligible the lock hold times corresponding to transactions that locally abort, the lock contention probability equation only encompasses contentions between a running transaction T and a co-local, committing transaction T', i.e.,

$$P_{lock} = P_{L,lock} = P^{C}_{L,lock}.$$

Similarly to the ETL case, moreover, the equation corresponding to the arrival rate of local update transactions is modified, to account for the fact that, in PR, the master node serves all such transactions:

$$\lambda^{UP}_{Ltx} = \lambda_{tx} \% w.$$

Finally, as the master node is owner of all the locks in the system,

$$L_O = 1.$$

4.2.2.3 Remote nodes involved in the distributed commit phase.

This section describes how the model computes the average number of remote nodes *NR* contacted during the distributed commit phase, given that a transaction does not abort while ac-

quiring local locks. Such a metrics is trivial to compute for PR and 2PC in ETL because such concurrency control scheme is only supported in full replication, i.e., the number of contacted remote nodes is simply $N-1$. The derivation of NR in the CTL in partial replication is, instead, more elaborated.

CTL-2PC. in 2PC, NR is obtained as the product of the number of remote nodes, $N-1$, and the probability (noted $P(R \geq 1|\neg AL)$) that a remote node replicates at least one datum written by the transaction, given that the transaction does not abort locally prior to starting 2PC. This probability is obtained by marginalizing over the distribution of the number of acquired local locks given that the transaction does not abort locally:

$$P(R \geq 1|\neg AL) = \sum_{i=0}^{N_w} P(R \geq 1 \wedge N_{L,lock} = i|\neg AL).$$

For the law of total probability, this can be rewritten as

$$P(R \geq 1|\neg AL) = \sum_{i=0}^{N_w} P(R \geq 1|\neg AL \wedge N_{L,lock} = i)P(N_{L,lock} = i|\neg AL).$$

In this expression, $P(N_{L,lock} = i|\neg AL)$ is the probability that a transaction has requested i local locks given that it has reached the distributed prepare phase, i.e.:

$$P(N_{L,lock} = i|\neg AL) = \frac{P_{LL}(i)(1-P_a(i))}{1-P_{AL}}.$$

Note that $P(R \geq 1|\neg AL \wedge N_{L,lock} = i) = P(R \geq 1|N_{L,lock} = i)$ because the fact that a node replicates at least one datum is independent from the outcome of the transaction and only depends on the number i of local locks.

Finally, the model has to compute the probability that a remote node replicates a datum written by the local node. In the case the local node is primary owner for a datum, this probability coincides with l. If the local node is not primary owner of the datum, there are two cases: the local node replicates the datum, which yields a probability l^2, or does not replicate it, which yields a probability $\frac{(1-l)r}{N-1}$. $P(R|\neg P_O)$ denotes the sum of these two probabilities, i.e.,

$P(R|\neg P_O) = l^2 + \frac{(1-l)r}{N-1}$. Overall, when the local node writes i items, for which it is primary owner, and $N_w - i$ remote ones, the probability that a remote node replicates at least one of such items is

$$P(R|N_{L,lock} = i) = 1 - (1-l)^i (1 - P(R|\neg P_O))^{N_w - i}. \tag{4.9}$$

Thus, the expected number of remote nodes being contacted upon a prepare phase, given the prepare phase is reached, is

$$NR = (N-1) \sum_{i=0}^{N_w} P(R|N_{L,lock} = i) P(N_{L,lock} = i|\neg AL).$$

CTL-PR. In PR, lock and data ownership are independent: in fact, the master node is owner of all the locks, but not of all the data items. Therefore, when computing the probability that a transactions writes i data items whose the master is primary owner, there is no need to condition on the event that the transaction does not abort locally.

The expected number of remotely contacted nodes at prepare time by a transaction T is simply $(N-1)$ multiplied by the probability that a remote node replicates at least one datum written by T; as in the 2PC case, this probability depends on whether the master node is primary owner of a written datum or not. The probability that the master nodes writes i data items whose it is primary owner, noted — for similarity to the previous case — $P(N_{L,lock} = i)$, is, then, computed as

$$P(N_{L,lock} = i) = P_O^i (1 - P_O)^{(N_w - i)}.$$

Hence, Equation 4.9 to compute $P(R|N_{L,lock} = i)$ still holds and

$$NR = (N-1) \sum_{i=0}^{N_w} P(R|N_{L,lock} = i) P(N_{L,lock} = i).$$

4.2.2.4 CPU Model

Like in previous analytical models (Yu et al., 1993; Di Sanzo et al., 2010), the CPU of the nodes of the platform is modeled as a $M/M/K$ multi-class queue with FCFS discipline, where K is

the number of cores per CPU. In general, the CPU serves J classes of jobs: denoting as D_j, resp. λ_j, $j = 1 \ldots J$, the service demand, resp. the arrival rate, of jobs belonging to class j, it is possible to compute the CPU utilization, ρ, as

$$\rho = \sum_{j=1}^{J} \frac{\lambda_j D_j}{K}. \tag{4.10}$$

In particular, DTPs can serve four classes of jobs from the point of view of CPU demands: read-only (noted L^{RO}) and local update (L^{UP}) transactions, requests for serving remote gets (R^{RG}) and remote update transactions (R^{UP}). Therefore,

$$\rho = \frac{\lambda_L^{RO} D_L^{RO} + \lambda_L^{UP} D_L^{UP} + \lambda_R^{UP} D_R^{UP} + \lambda_R^{RG} D_R^{RG}}{K}.$$

Then, defining

$$\alpha = \frac{K\rho^K}{K!(1-\rho)}, \quad \beta = \sum_{i=1}^{K-1} \frac{K\rho^i}{i!}, \quad \gamma = 1 + \frac{\alpha}{K(\alpha+\beta)(1-\rho)},$$

the CPU response time R_i corresponding to a job with demand D_i, without taking into account the latency of network-related operations, is given by

$$R_i = \gamma D_i.$$

The following discussion provides the equations employed by the model to compute arrival rates and service demands depending on the adopted concurrency control scheme/replication protocol.

4.2.2.4.1 Transactions' service demand computation. The CPU service demands equations are described following the scheme adopted for the transactions' execution times in Section 4.2.2.1. The employed notation also follows the one used in that section.

Read-only transactions' CPU service demand is, then, computed as the sum of the CPU service time required to start, execute and commit a transaction, plus the service time corre-

sponding to local and remote get operations:

$$D_L^{RO} = D^{beg} + D_B^{RO} + \sum_{i=1}^{N_r} [\mathscr{L} D_L^g + (1-\mathscr{L})(D_L^g + D_R^\gamma)] + D_{com}^{RO}.$$

The CPU demand for remote update transactions is computed as

$$D_R^{UP} = D_R^{prep} + D_R^{com} + N_{AR}(D_R^{prep} + D_R^{rol}),$$

where D_R^{prep}, D_R^{com} and D_R^{rol} are, respectively, the CPU demands of remote prepare, commit and rollback operations.

Note that this CPU demand is computed as the sum of the cost of one successful remote validation plus the demand corresponding to N_{AR} failed validations.

Local update transactions' service demand is, instead, obtained as the sum of the CPU demands corresponding to possible transactions re-executions:

$$D^{UP} = D_C^{UP} + N_{AR}(D_{AR}^{UP} + N_{AL} D_{AL}^{UP}) + N_{AL} D_{AL}^{UP} + D_{com}^{UP},$$

with the single contributes being

$$D_C^{UP} = D_L^{UP} + D_L^{prep} + D^{prep} + D^{wb} + D^{dec},$$

$$D_{AR}^{UP} = D_L^{UP} + D_L^{prep} + D^{prep} + D_L^{rol} + D^{dec},$$

and

$$D_L^{UP} = D^{beg} + D_B^{UP} + N_w D^p + \sum_{i=1}^{N_R} [\mathscr{L} D_L^g + (1-\mathscr{L})(D_L^g + D_R^\gamma)].$$

Just like in Section 4.2.2.1, the equation for D_{AL}^{UP} varies depending on the employed concurrency control scheme.

D_{AL}^{UP} in ETL. In ETL, the CPU service demand of an aborted update transaction T depends on

how much T has progressed in its local computation before incurring a lock conflict. Therefore,

$$D_{AL}^{UP} = \sum_{i=1}^{N_w} [D_L^{roll} + (\frac{D_B^{UP}}{N_w} i) P(AL = i|AL)].$$

D_{AL}^{UP} in CTL. Given that a local transaction cannot abort before reaching the prepare phase, in CTL D_{LAL}^{UP} is computed as

$$D_{AL}^{UP} = D_L^{UP} + D_L^{prep} + D_L^{roll}.$$

4.2.2.4.2 CPU jobs arrival rates computation. This section obtains the arrival rates for the different CPU job classes depending on the locking scheme and replication protocol.

ETL. In computing CPU jobs arrival rates, the most important trait of the considered ETL scheme is that it is supported for full replication and it does not encompass the concept of primary/lock owner.

2PC. In 2PC all nodes serve the same workload. Therefore, each node serves a fraction $\frac{1}{N}$ of local update and read-only transactions, namely,

$$\lambda_L^{RO} = \frac{\lambda_{tx}(1-w)}{N}, \qquad \lambda_L^{UP} = \frac{\lambda_{tx}(w)}{N}.$$

The arrival rate of CPU jobs corresponding to remote transactions on node v, instead, is computed by taking into account that, upon being spawned for the first time, a transaction will or will not contact any other node, during its N_{AR} re-runs and final successful execution. Therefore, λ_R^{UP} is computed as the product of the arrival rate of transactions that are generated on other nodes and the probability that a remote transaction will try to access at least one datum replicated by v (noted $P(R \geq 1)$):

$$\lambda_R^{UP} = (N-1)\lambda_L^{UP} P(R \geq 1).$$

In ETL, clearly,

$$P(R \geq 1) = 1.$$

Moreover, as ETL is supported only in full replication,

$$\lambda_R^G = 0.$$

PR. In PR, CPU jobs arrival rates are different for master and slaves. For the former,

$$\lambda_L^{UP} = \lambda w, \quad \lambda_R^{UP} = 0, \quad \lambda_L^{RO} = 0;$$

for the latter ones

$$\lambda_L^{UP} = 0, \quad \lambda_R^{UP} = \lambda w, \quad \lambda_L^{RO} = \frac{\lambda(1-w)}{N-1}.$$

Clearly, for both

$$\lambda_R^G = 0.$$

CTL. As far as CPU jobs arrival rates are concerned, the only difference between the ETL and the CTL cases lies in the arrival rate corresponding to remote update transactions.

Following the same rationale used to compute the number of remote nodes contacted at commit time in Section 4.2.2.3, $P(R \geq 1)$ is computed, equally for 2PC and PR, as

$$P(R \geq 1) = \sum_{i=0}^{N_w} P(R \geq 1 | N_{L,lock} = i) P(N_{L,lock} = i),$$

where $N_{L,lock} = i$ is the probability that the local node is primary owner of i written data items. This probability corresponds to $P_{LL}(i)$ (Equation 4.7) and the analytical formulation for $P(R \geq 1 | N_{L,lock} = i)$ has already been obtained in Equation 4.9.

Finally, the average arrival rate or remote get requests, each having a CPU demand equal to D_R^{RG}, is obtained, taking into account both requests coming from read-only and update transac-

tions:

$$\lambda_G^R = \frac{1-\mathscr{L}}{N-1}\left(\lambda_L^{RO} N_r + \lambda_{Rtx}^{UP} N_R (N_{AR} + N_{AL}(N_{AR}+1))\right).$$

4.2.3 Black box modeling in the proposed DTP models

Developing a white box network communication model capable of accurately predicting the response time of network-bound operations in complex applications, deployed over virtualized cloud infrastructures is a very challenging task. First, in this type of infrastructures, little or no knowledge is available about the underlying network topology, hardware infrastructure and virtualization software overhead: this affects the possibility of measuring resource demands accurately, and makes the analytical derivation of response times cumbersome (Whiteaker et al., 2011). Moreover, complex applications' software stack typically lies on top of group communication toolkits that provide several inter-process synchronization services (like failure detection, group membership, remote procedure calls) the configuration and the internal design of this layer also affect performance in a way that is hard to predict (Couceiro et al., 2010).

For these reasons, the proposed models rely on ML to predict the latency of network-bound operations, i.e., of the remote get (R_R^{γ}), prepare (R^{prep}), and final decision phases (R^{dec}).

Specifically, the ML employed to model network-bound operations' execution time is Cubist[1], a Decision Tree (DT) regressor that approximates multivariate functions by means of piece-wise linear approximations. Analogously to classic decision tree based classifiers, such as C4.5 and ID3 (Quinlan, 1986, 1993a), Cubist builds decision trees choosing the branching attribute such that the resulting split maximizes the normalized information gain. However, unlike C4.5 and ID3, which contain elements in a finite discrete domain (i.e., the predicted class) as leaves of the decision tree, Cubist places a multivariate linear model at each leaf.

In particular, three Cubist models are generated for each target feature: each model is trained over samples corresponding to a specific replication setting: full replication (i.e., replication degree = N), partial replication, and data partitioning (i.e., replication degree = 1). The rational behind this choice is that the replication degree affects other input features for the ML

[1] https://www.rulequest.com/cubist-info.html

model in a strongly non-linear fashion. For instance, for small values of the replication degree, small shifts of this parameter typically lead to large changes of the arrival rate of the remote gets. Conversely, when close to full replication, the remote gets' arrival rate drops sharply to 0.

Building separate models on the basis of the replication degree allows Cubist to isolate samples in the parameters' space corresponding to the case of "extreme" replication policies, i.e., full replication and data partitioning. The three Cubist models are built and queried separately: depending on the replication degree setting corresponding to the specific invocation, the appropriate Cubist model is employed.

This solution is related in spirit to Ensemble Learning techniques (Mendes-Moreira et al., 2012), in which multiple single learners are combined together to generate a stronger, more accurate model. In this case, the single learners are trained on disjoint training sets, in order to increase their predictive power in a reduced portion of the parameters' space, in which the relations between input and output are subject to more linear, and hence more easily deductible, dynamics.

In order to build an initial knowledge base to train the machine learner, the proposed models rely on a suite of synthetic benchmarks that generate heterogeneous transactional workloads in terms of mean size of messages, CPU utilization and network load. The set of input features provided as input to Cubist characterizes the workload from the point of view of network utilization, as dynamics relevant to data contention are captured by the white box model that we described in the previous section. Specifically, for each of the three predicted network latencies we build an independent ML-based model, based on the following features: number of nodes in the system, average number of nodes contacted during the 2PC, average size of the messages exchanged in remote interactions (i.e., prepare and remote gets), the rate at which these interactions occur, CPU utilization, number of active threads on each node. Moreover, given that Cubist only exploits linear approximations in the leaves of the decision tree, the set of input features is widened by providing also metrics that are obtained as product of these basic features and that are known, from previous analytical network communication models, to be highly correlated with the response time of network-bound operations (e.g., the throughput, in byte per second, of sent and received messages) (Nicola & Jarke, 2000; Raghuram et al., 1992).

4.2.4 Model resolution

From the analysis carried out in previous sections, it is clear that there are some interdependencies among the CPU, network and data contention models: the CPU and network response times are influenced by the lock contention probability, which depends on the lock hold times, which, on their turn, depend on network and CPU response times. This cyclic dependency is solved through a fixed point recursion on the set of abort probability values ($P_{L,lock}$ in the PR case, $P_{L,lock}$ and $P_{R,lock}$ for the encounter time 2PC case and P_{lock} for the commit time 2PC one).

On the first iteration, abort probabilities are set to 0; the value in input at iteration i is computed by solving the model with metrics obtained at iteration $i-1$; this process ends when the relative difference between values computed on two consecutive iterations falls under a threshold (set to 1%) and typically converges in a few iterations. It is out of the scope of this work to demonstrate the convergence of this iterative solution method, which has been adopted to solve several previous performance models of concurrency control protocols (see, e.g, (Yu et al., 1993; Di Sanzo et al., 2010; Menascé & Nakanishi, 1982)); as in previous studies, it has been observed that it always converges in a few iterations, provided that the input assignment defines a stable system.

In a similar fashion, it is clear that the analytical and ML models, built separately, are closely intertwined. The training set of the ML, in fact, can be built by gathering measurements of the required metrics while deploying target benchmarks using different values for the platform scale, load, data replication degree, message size, etc. However, at query time, some of the input features are not known, as they depend on the sought after performance of the application in the target configuration. Assume, for instance, to predict the network latencies incurred for a target configuration using 80 nodes, while observing these metric on a platform with 3 nodes. In this case, it is necessary to predict, among other metrics, the number of nodes involved in the distributed commit phase, or the rate at which prepare phases would be initiated in the target configuration, which can vary drastically from the one measured in the current configuration.

The *impera* component of the *Divide et impera* gray box modeling technique materializes in the way in which the employed white box and black box models are reconciled in such a way to address the aforementioned issue and obtain the prediction of the system as a whole.

More specifically, given an input transactions arrival rate, set of abort probabilities and workload characterization, the analytical model can predict metrics like CPU utilization, expected number of retries (and, hence, rate of prepare/commit/rollback operations) and number of nodes involved in the distributed synchronization phases. Such information are exploited to provide the ML all the inputs it needs in order to produce its forecasts about execution time of network bound operations. The output of the ML is, finally, given back in input to the analytical model, which can, thus, compute the full execution time of transactions.

4.2.4.1 Predicted KPIs

The discussion conducted so far has shown how the presented performance models are able to predict transactions' response time, abort probability and latency of network-bound operations. This section illustrates how the model can be also exploited to obtain an approximation for the closed-system throughput, i.e., the throughput delivered by the application when deployed over a set of N nodes having each θ active threads that process transactions with zero think time.

This metric can be computed by exploiting Little's law (Little, 1961) in an iterative fashion: at each iteration, the closed-system throughput is obtained starting from the average transactions' response time (Yu et al., 1993). At the first iteration, a low value of transactions arrival rate λ is provided as input to the model. Next, the open model is solved to compute R^{UP}, R^{RO} and the transactions' average response time as $R^{avg} = \%wR^{UP} + (1 - \%w)R^{RO}$. The closed-system throughput for iteration i is, then, obtained by applying Little's law as described in the following. In the 2PC case (for both ETL and CTL), it is noted $T_C(i)$ and it is obtained by simply computing $T_C(i) = \frac{N\theta}{R^{avg}}$. For the PR case, instead, the model computes the corresponding metrics separately for the master $T_C^M(i) = \frac{\theta}{R^{UP}}$ and the slaves $T_C^S(i) = \frac{(N-1)\theta}{R^{RO}}$; the overall closed system throughput is obtained as the minimum between the two.

The arrival rate for the next iteration λ is obtained using a first order Newton's method defined as $\lambda = T_C(i-1) + \frac{(T_C(i) - T_C(i-1))}{2}$, where $T_C(i)$ is specialized according to previous definitions.

This process typically completes in a few iterative steps, except for high contention scenarios, which may yield to convergence problems. This is a typical issue that arises when adopting

such a recursive resolution algorithm for analytical models of transactional systems (Yu et al., 1993). To cope with such an issue, the model solving algorithm implements a fall-back solution that spans, at a given granularity (set to 10 tx/sec), all possible arrival rate values within a configurable interval. This algorithm returns the solution which minimizes the error between the input arrival rate an the output closed system throughput. This guarantees convergence to the desired accuracy (set to 1%) in a bounded number of steps.

Finally, note that, when solving the open model to obtain closed system throughput, the model explicitly takes into account the fact that a transaction generated by a given thread cannot contend locks or physical resources with itself (similarly to the MVA algorithm (Reiser & Lavenberg, 1980)). This is achieved by means of simple modifications to the provided equations, which are not provided for the sake of presentation; however, to give an example of such modifications, the rate of incoming co-local transactions with which a local transaction can contend with is scaled by a factor of $\frac{\theta-1}{\theta}$.

4.2.5 Models evaluation

This section is devoted at assessing the effectiveness of the *Divide et impera* approach when applied to the problem of DTPs performance modeling, and in particular to the Infinispan DTP. Section 4.2.5.1 describes the experimental platform and the transactional applications and workloads employed throughout the evaluation study; Section 4.2.5.2 evaluates the soundness of the ACF abstraction; Section 4.2.5.3 evaluates the accuracy of the standalone black box learners; Section 4.2.5.4 is devoted to evaluate the *Divide et impera* approach, by assessing the accuracy of the hybrid proposed models; Section 4.2.5.5 compares the two models with pure off-the-shelf black box approaches; finally, Section 4.2.5.6 discusses operational aspects of the models, like convergence speed of the resolution algorithm described in Section 4.2.4 and the instrumentation overhead incurred by Infinispan in order to collect the input parameters needed by the models.

4.2.5.1 Experimental test-bed

Experimental platform. One of the main motivations behind the choice of employing a ML-based model for network-bound operations in DTPs is portability: by avoiding to model via white box techniques the network-bound interactions, the resulting *Divide et impera* performance model is seamlessly applicable over heterogeneous (and possibly virtualized) infrastructures. In order to back this claim, the evaluation of the proposed models has been conducted over different infrastructures, encompassing both virtualized and bare-metal environments and both private and public clouds. The deployment infrastructures are hereafter described, together with the notation employed to refer them in the following plots and discussion.

1. **Bare-metal private cluster (PC-B):** it is composed by 10 servers equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) processors and 8 GB of RAM and interconnected via a private Gigabit Ethernet. Each machine runs Ubuntu 12.04 Linux distribution with a 3.2.0 kernel.

2. **Virtualized private cloud (PC-V):** it is composed by 140 Virtual Machines (VM), equipped with 1 Virtual CPU and 2GBs of RAM; each VM runs a Fedora 17 Linux distribution with 3.3.4 kernel. The underlying physical infrastructure consists 18 servers equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) processors and 32 GB of RAM and interconnected via a private Gigabit Ethernet. The employed virtualization software is Openstack Folsom[2].

3. **FutureGrid Infiniband cloud (FG-I)** [3]**:** it is composed by 100 VMs, each equipped with one virtual core and 2 GB of RAM and running a Fedora 17 Linux distribution with 3.3.4 kernel. VMs are interconnected via an Infiniband network. The employed virtualization software is Openstack Folsom.

4. **FutureGrid Ethernet cloud (FG-E):** it is composed by 100 VMs, each equipped with one virtual core and 2 GB of RAM and running a Fedora 17 Linux distribution with 3.3.4

[2]http://www.openstack.org/software/folsom/
[3]FutureGrid is a public, non commercial, cloud (Fox et al., 2012; Laszewski et al., 2010).

kernel. VMs are interconnected via an Ethernet network. The employed virtualization software is Openstack Havana [4].

5. **Amazon EC2 cloud (EC2):** it is composed by 20 Extra Large Instances, which are equipped with 15GB of RAM and 4 virtual cores with 2 EC2 Compute Units each. Each VM runs Ubuntu 13.04.

Applications and workloads. The accuracy of the proposed models has been evaluated by means of three benchmarks.

1. **Radargun (RG)** [5]**:** a benchmarking framework specifically designed to test the performance of distributed, transactional key-value stores. Workloads generated by Radargun applications are synthetic, namely consist only of put/get operations; the employed dataset is either composed by 100K keys (LA) or by 1K (SM).

2. **TPCC**[6]**:** a standard benchmark for OLTP systems, which simulates the activities of a wholesale supplier and generates mixes of read-only and update transactions with strongly skewed access patterns and heterogeneous durations. Two transactional mixes are considered: a read-dominated (TPCC-R) and a write-intensive (TPCC-W) one. Note that, given that the standard implementation of TPCC is defined over a relational database, the benchmark has been ported to the key-value data model exposed by Infinispan.

3. **YCSB** (Cooper et al., 2010)**:** it represents the *de facto* standard benchmark for key-value stores. Also in this case, the benchmark represents a porting of the original one: in fact, YCSB does not encompass transactions, but only single put/get operations issued against a key-value store. The baseline YCSB workloads considered in this study are A, B and F: workload A has a mix of 50/50 reads and writes; workload B contains a 95/5 reads/update mix; in workload F records are first read and then modified within a transaction. In order to evaluate the accuracy of the models in predicting a wider set of transactional workloads,

[4] http://www.openstack.org/software/havana/
[5] https://github.com/radargun/radargun
[6] http://www.tpc.org/tpcc/

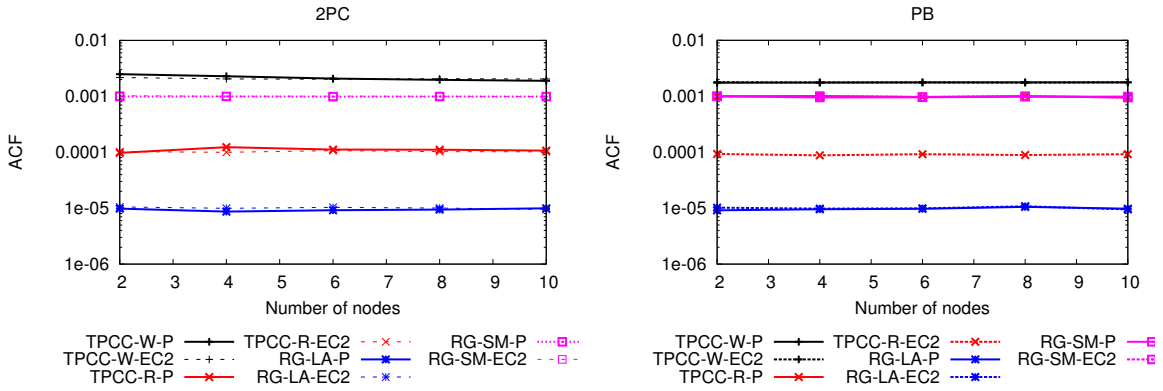
also variants of the aforementioned workloads have been employed, in which the number of performed operations varies. Finally, two data access patterns are considered: zipfian and hot-spot. In the first one, the popularity of data items follows a zipfian distribution (with YCSB's *zipfian constant* set to 0.7); in the second one, 99% of the data requests are issued against the 1% of the whole data set.

YCSB's workloads will be referred to by using the notation N-D-P-I: N refers to the original workload's YCSB notation (Cooper et al., 2010); D is the number of distinct data items that are read by a read-only transaction; for update transactions, it is the number of distinct data items that are written (for the F workload, which exhibits a read-modify-write pattern, data are both read and written); P encodes the data access pattern (Z stands for zipfian, H for hot-spot); finally, I specifies the cloud infrastructure over which the benchmark has been run (PC-V or FG-E). For all the experiments, the data platform is populated with 500000 keys and data are scattered across nodes according to Infinispan's default consistent hash function

The Key Performance Indicators employed to evaluate the accuracy of the models are transactions' commit probability (measured as $\frac{\#committed\ xact}{\#total\ xact}$) and closed-system throughput (measured as transactions/second). To this end, a workload generator is deployed on each node and consists of three threads (and TPCC) or one thread (for Radargun and YCSB) that inject requests against the collocated Infinispan instance, in closed loop. The accuracy of the models in predicting values for both these KPIs is evaluated by reporting the Mean Average Percentage Error (MAPE), computed as $Avg(\frac{|real\ kpi - pred\ kpi|}{real\ kpi})$.

4.2.5.2 ACF validation

The Application Contention Factor (ACF), introduced in Section 4.2.2.2, is the abstraction employed by the proposed models to characterize, in a concise and lightweight fashion, the data access pattern exhibited by a transactional application. This section is dedicated to validate the soundness of the ACF abstraction as well as to provide a more formal explanation behind it; in particular, the following evaluation shows that ACF can be considered as a workload invariant, and thus used to serve what-if queries, and that is approximates a non-uniform data access



(a) 2PC. (b) PR.

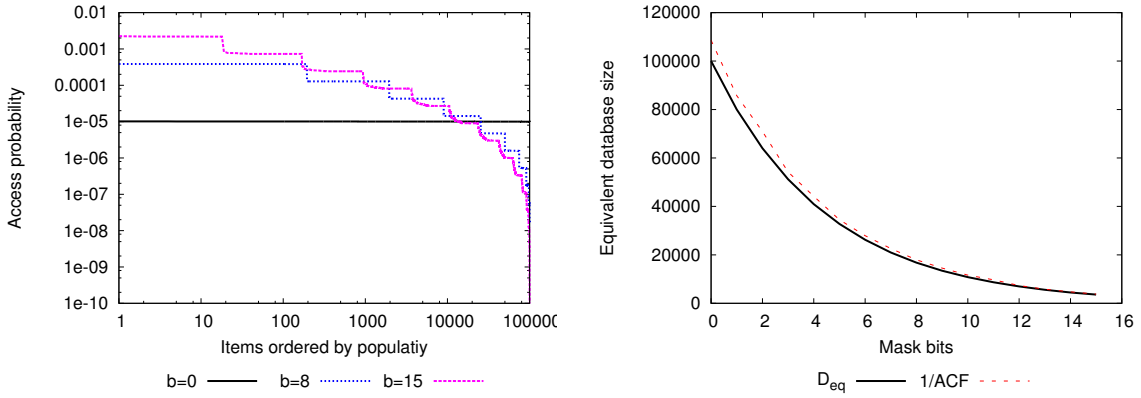
Figure 4.3: ACF using heterogeneous benchmarks and platforms.

pattern by means of an equivalent uniform one.

ACF as workload invariant. Figure 4.3 depicts the ACFs obtained when running the TPCC and Radargun workloads on EC2 and on the bare-metal private cluster. The plot on the left reports the ACF values obtained using the 2PC replication scheme, whereas the one on the right reports the ACF computed using the PR scheme. The plots show that the ACF represents an invariant of a given workload, i.e., that it is not affected by *i*) the scale and nature (private vs public) of the platform on which the transactional data grid is deployed; *ii*) the data replication protocol employed by the transactional data grid (2PC or PR). This confirms that the ACF can be measured in a given configuration, and used to speculated about performance delivered by the same workload running according to another Infinispan configuration.

ACF as equivalent uniform dataset. In Section 4.2.2.2, the ACF has been described as the inverse of the size (D_{eq}) of a dataset that, if uniformly accessed, would yield the same abort probability of the actual dataset, accessed according to an arbitrary pattern.

In order to validate this statement, the following experiment has been carried out: *i*) the Radargun benchmark has been set to execute transactions composed by a single write operation on a datum selected among a set of 100K items on the basis of a non-uniform distribution; *ii*) the ACF has been obtained starting from measurements collected during the experiment; *iii*) the



(a) Data popularity

(b) Equivalent dataset size and comparison with $\frac{1}{ACF}$

Figure 4.4: NuRand's data access pattern varying the bit mask.

size of the uniform equivalent dataset for the case of the target workload (D_{eq}) is analytically obtained; *iv*) the resulting value has been compared with the inverse of the measured ACF.

The employed non-uniform data access distribution is the NuRand one, which is defined as follows:

$$NuRand(A,x,y) = (((random(0,A)|random(x,y)))\%(y-x+1))+x.$$

In the above equation, A is a all-1's bit-mask and x,y are the extrema values that the random variable can take. Essentially, the NuRand is based on the bitwise OR of two uniformly drawn random values: given that the OR function between two bits evaluates to 1 in 75% of the possible inputs, the NuRand skews the drawn value according to the input bit-mask. The NuRand choice has been motivated by two reasons: *i*) the NuRand distribution is employed in TPCC, and is therefore representative of realistic data access pattern skew; *ii*) its relatively simple formulation allows for the derivation of its equivalent uniform distribution, when employed to determine the data access pattern in a transactional application.

Employing the NuRand in the aforementioned workload essentially implies that transactional accesses to the dataset are split in $b+1$ classes, corresponding to $b+1$ disjoint and uniformly accessed sub-datasets. Figure 4.4a shows the impact of the mask on the data access pattern, by plotting the access probability of every datum in the dataset after having sorted them by decreasing popularity. The picture shows both the skew growing with the size of the mask

as well as the aforementioned partition of the dataset.

Figure 4.4b, instead, compares the analytically obtained D_{eq} with the inverse of the measured ACF. In the plot, the number of 1's in the NuRand bitmask, b, varies between 0 (uniformly accessed dataset) and 15 (the maximum skewness achievable by NuRand over 100K keys), and for each value of b, the corresponding D_{eq} is depicted. Computing D_{eq} for $b = i$ requires first obtaining the probability of a transaction to belong to that class (p_i) and the size of the sub-dataset accessed by that class of transactions (d_i). The computation of d_i and p_i follows straightforwardly from the definition of NuRand[7]. Then, D_{eq} is obtained as follows:

$$P(lock\ contention) = \sum_{i=0}^{b} P(xact\ is\ of\ class\ i \wedge lock\ is\ taken\ by\ another\ xact\ of\ class\ i)$$

$$\frac{\lambda_{lock} \cdot T_H}{D_{eq}} = \sum_{i=0}^{b} p_i \cdot \frac{p_i \lambda_{lock} \cdot T_H^i}{d_i}.$$

Given that, in the experiment, the transactions are uniform in terms of execution time (i.e., $T_H^i = T_H, \forall i \in [0..b]$),

$$\frac{1}{D_{eq}} = \sum_{i=0}^{b} p_i \cdot \frac{p_i}{d_i}. \tag{4.11}$$

The plots in Figure 4.4a show that, for every value of b, the analytically obtained D_{eq} and the inverse of the measured ACF are very close. It is noteworthy to highlight that computing D_{eq} analytically, as suggested by previous work (Tay et al., 1985), requires the *a priori* knowledge about b, p_i and d_i; conversely, such information is not needed to compute the ACF.

4.2.5.3 ML validation

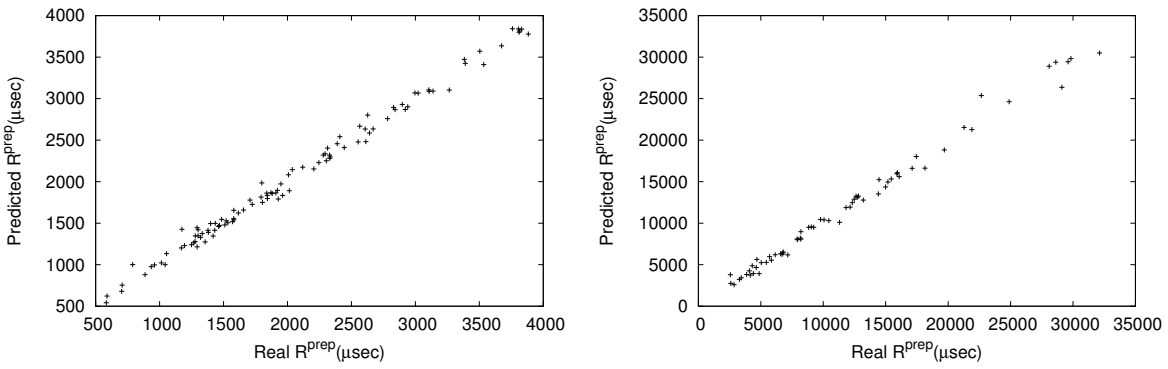
This section is dedicated to assess the accuracy of the standalone black box components employed in the proposed models to predict latencies of network bound operations. That is, the following accuracy results only refer to the ability of the pure ML model to forecast response

[7]Refer to the TPC-C specification (TPC Council, 2011) for a detailed description of NuRand and to the paper by Leutenegger and Dias (Leutenegger & Dias, 1993) for a comprehensive analysis of its properties

time of network operations when all the correct input values are provided, and not when the ML is coupled with the analytical model to obtain them (as described in Section 4.2.4).

The evaluation proposed in this section in split in two parts: the first focuses on the simpler case of full replication; the second, instead, describes the additional challenges that arise when coping with partial replication, and evaluates the accuracy of the solution proposed to tackle the corresponding more complex prediction task.

4.2.5.3.1 Black box modeling in full replication. In full replication, black box modeling is only applied to the problem of forecasting the latencies of network-bound operations executed during the distributed commit phase (i.e., R^{prep} and R^{dec}).



(a) PC-B (b) EC2

Figure 4.5: Accuracy of the ML-based R^{prep} predictions in full replication (ETL).

The scatter-plots in Figure 4.5 report the accuracy (in 10-fold cross validation) of the Cubist model built to predict R^{prep}; specifically, the plots refer to a 2PC ETL Infinispan deployment on the PC-B and EC2 hosting infrastructures. The plots highlight that, on both the private cluster and on EC2, the ML attains a high prediction accuracy. Specifically, the correlation factor is around 99% in both cases, with an average absolute error equal to 500 micro-seconds for EC2 and around 60 micro-seconds for the private cluster. Note that, in practice, the relative error is similar on both the platforms, since, on EC2, the maximum value of R^{prep} is around 10 times larger than the maximum R^{prep} value on the private cluster.

4.2.5.3.2 Black box modeling in partial replication. In DTPs, the replication degree is an important parameter that has a strong impact on both the contention and the CPU models.

Likewise, it also affects input features for the ML model in a strongly non-linear fashion. For instance, for small values of the replication degree, small shifts of this parameter typically lead to large changes of the arrival rate of the remote gets. Conversely, when close to full replication, the remote gets' arrival rate drops sharply to 0.

As hinted in Section 4.2.3, the proposed models tackle this increased complexity by employing separate models to forecast the network-related performance functions corresponding to the case of "extreme" replication policies, i.e., full replication and data partitioning. As a result, for each target feature, three different black box models are built, each trained over disjoint data sets, corresponding to the case of data partitioning (i.e., replication degree = 1), partial replication and full replication (i.e., replication degree = N).

The effectiveness of this approach in increasing predictive accuracy has been tested using Cubist and several other ML techniques included in the Weka (Hall et al., 2009) suite, namely MultiLayerPerceptron (based on Neural Networks), SmoReg (based on Support Vector Machines) and M5Rules (a Decision Tree regressor like Cubist) (Bishop, 2006).

The results of the tests are shown in Figure 4.6, which reports the 10 fold cross-validation accuracy in terms of median and 90-th percentile of the prediction for the R^{prep} feature of a 2PC CTL Infinispan deployment on the PC-V infrastructure, when using the single model (S) or the "multi-model" obtained by training the learners on disjoint data-sets (M)[8]. The results confirm the effectiveness of the employed technique in increasing the accuracy, as well as its general applicability. In fact, the multi-model consistently outperforms the single model in terms of both median and 90-th percentile accuracy, regardless the employed learning algorithm. Finally, Figure 4.7 shows the scatter plots contrasting the predicted vs the actual values of the R^{prep} feature, on the PC-V and on FG-E test-beds obtained using 10 folds cross-validation. The plots confirm the viability of employing black box modeling to predict the response time of network-bound operations also in presence of highly non-linear performance functions stemming from the adoption of partial replication.

[8]Results relevant to impact of this technique on the accuracy of the models for other features (e.g., R_R^γ) and for the FutureGrid infrastructure are not shown as they are similar to the presented ones.

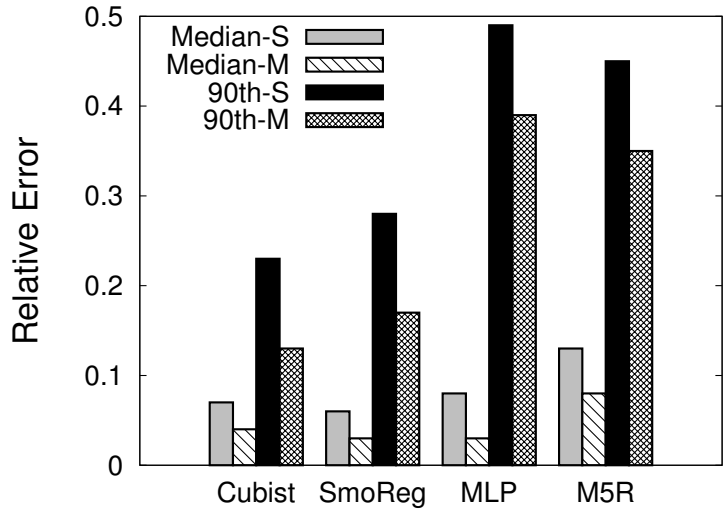
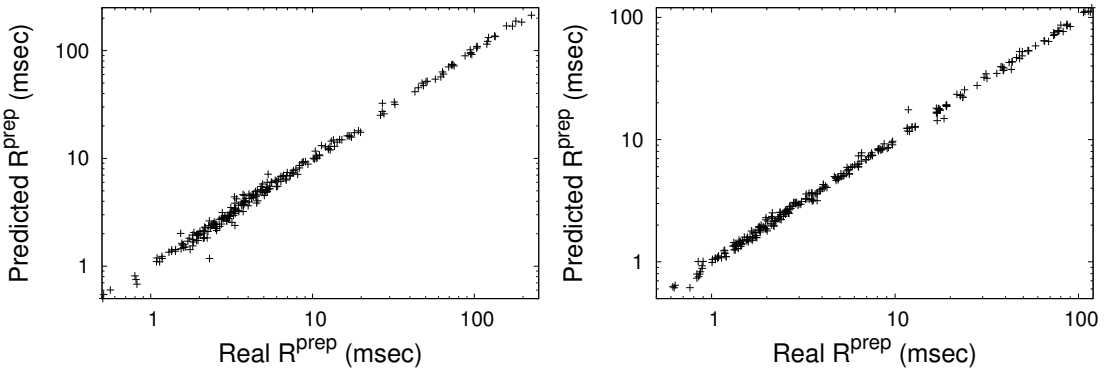


Figure 4.6: Single and multi-model validation for different MLs



(a) PC-V

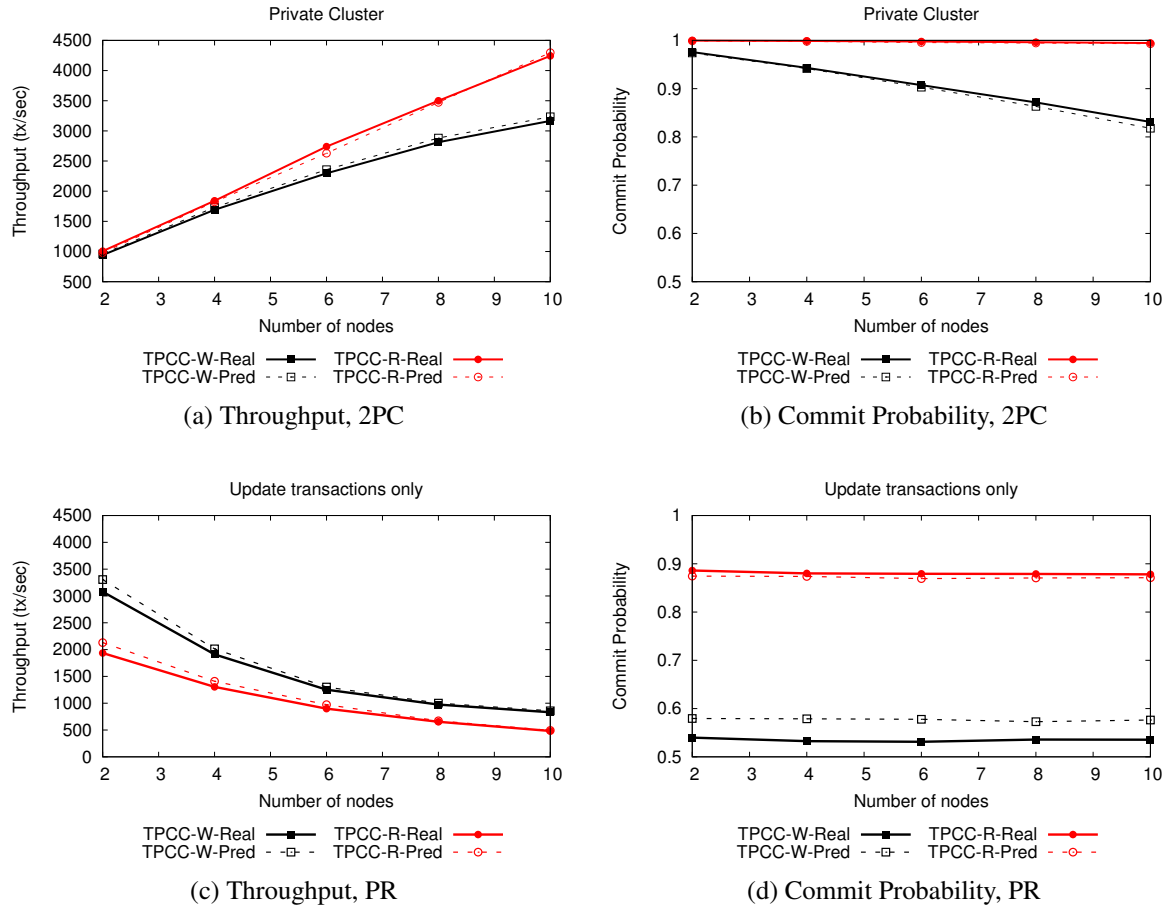
(b) FG-E

Figure 4.7: Accuracy of the ML-based R^{prep} predictions in partial replication (CTL).

4.2.5.4 Validation of the hybrid performance models

This section evaluates the accuracy achieved by performance predictors as a whole, i.e., when the underlying white and black box models are coupled according to the scheme described in Section 4.2.4. Similarly to the ML validation case, also this section first focuses on the ETL case, which encompasses full replication deployments based on both 2PC and PR, and then reports accuracy results for the CTL case, which also entails partial replication.

4.2.5.4.1 ETL validation. The validation of the ETL models has been performed using the PC-B, FG-I and EC2 infrastructures, and has been aimed at assessing the accuracy of the model



(a) Throughput, 2PC

(b) Commit Probability, 2PC

(c) Throughput, PR

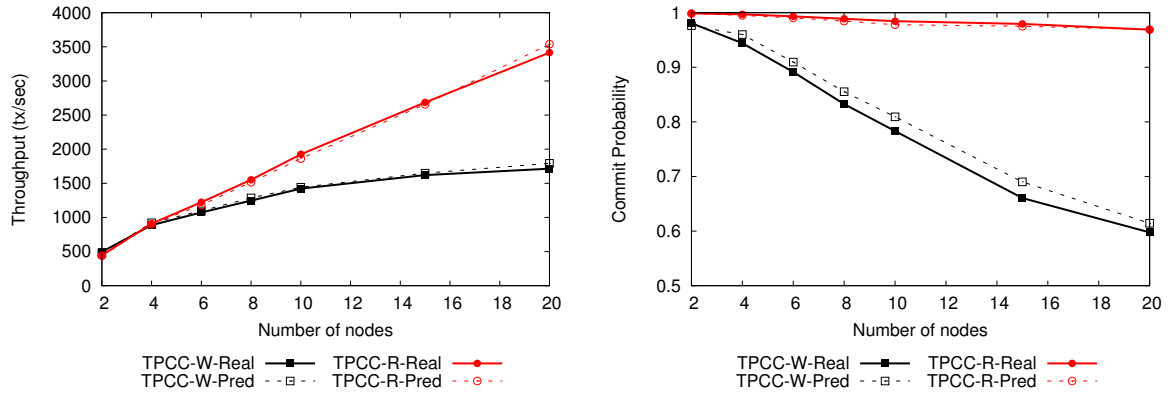
(d) Commit Probability, PR

Figure 4.8: Validation of the ETL models using the TPC-C benchmark deployed on the PC-B infrastructure.

to predict throughput and commit probability of transactions executed according to both the 2PC and the PB replication protocols.

Figure 4.8 compares actual and predicted performance of the two considered TPC-C workloads deployed over PC-B: top plots refer to 2PC and bottom ones to PR; left plots refer to throughput and right ones to commit probability. Figure 4.9, instead, depicts the model's prediction capabilities in predicting performance of the two TPC-C workloads when deployed over Amazon EC2, for the 2PC case.

The two sets of plots showcase the proposed models' high accuracy in predicting performance for heterogeneous workloads when running according to different platform configurations (in terms of nodes and replication protocol) and when deployed over different infrastruc-



(a) Throughput (b) Commit probability

Figure 4.9: Validation of the ETL-2PC model using the TPC-C benchmark deployed on the EC2 infrastructure.

tures.

Figure4.10 refers to a large scale deployment of several Radargun workloads, characterized by different transactional mixes ($w = x$ in the labels indicate that the workload is composed by $x\%$ update transactions). The workloads are deployed over the FG-I infrastructure and run transactions according to the 2PC replication protocol. In addition to the aforementioned KPIs (throughput and commit probability, from Figure 4.10a to Figure 4.10d), the figure also depicts (in Figure 4.10e and Figure 4.10f) the accuracy of the models in predicting the distributed commit latency when coupling the black box learner with the white box model.

Overall, the plots confirm the accuracy of the models in predicting throughput, abort probability and distributed commit latencies of applications deployed over a large-scale data platform. The MAPE attained for throughput, commit probability and distributed commit latency is, respectively, 6%, 0.7% and 13%. Fig. 4.11 also reports the cumulative density functions (CDF) of the absolute relative error for these KPIs, showing that the 90-th percentile for throughput is less than 10%, the 80-th percentile for abort probability is less than 1% and that the 85-th percentile for the distributed commit latency is less than 20%.

The proposed models also result to be highly effective in predicting whether the primary scalability bottleneck for an application lies in the physical layer or in the level of incurred data contention. This capability of the model is demonstrated, in particular, in Figs 4.10b, 4.10d

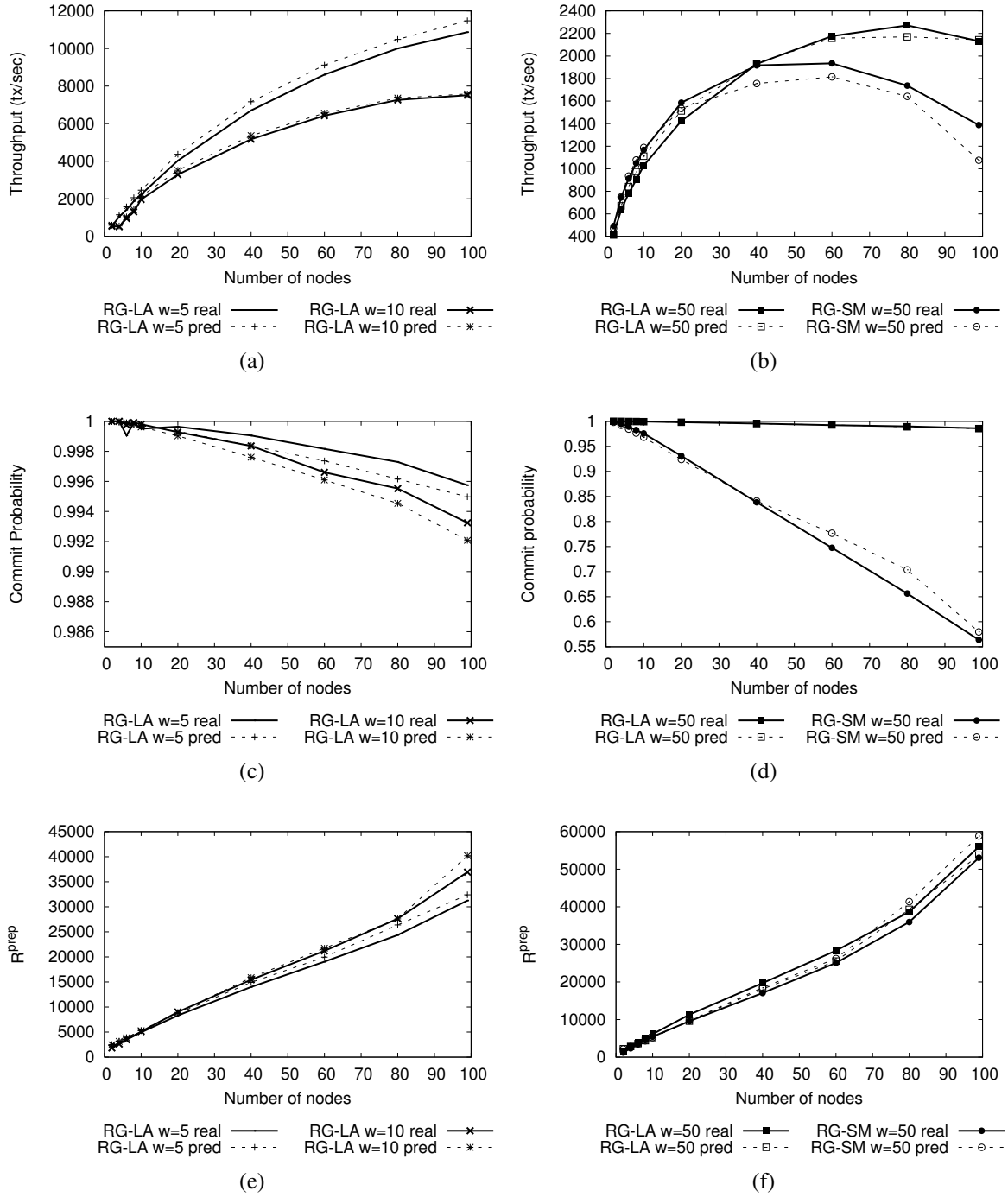


Figure 4.10: Validation of the ETL-2PC model using the Radargun benchmark on the FG-I infrastructure.

and 4.10f, which report predictions for the write intensive Radargun workloads (w=50). The plots show that the models are able to predict that, while being characterized by similar commit latencies, the RG-SM workload scales half as much as the RG-LA workload (particularly, up to

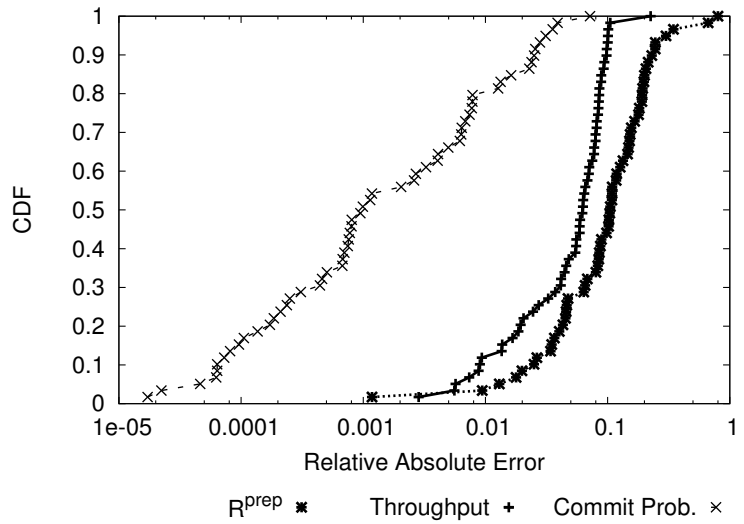


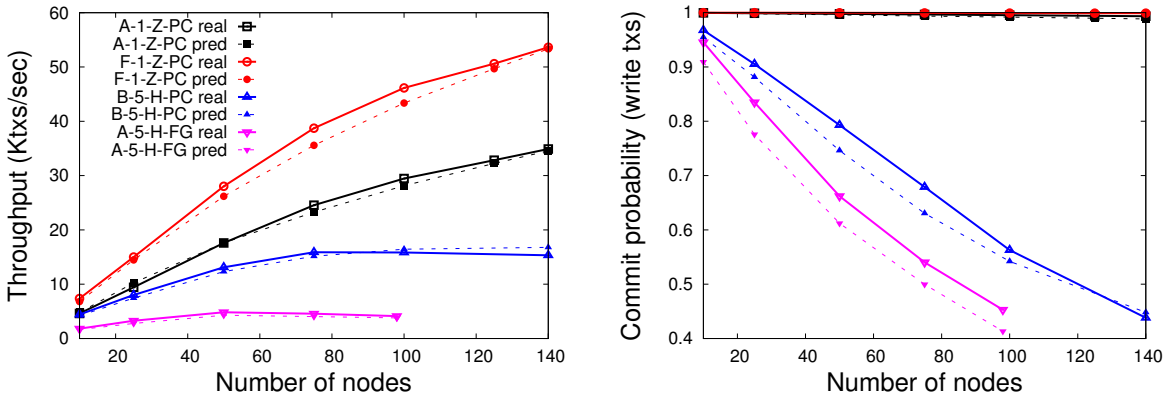
Figure 4.11: Absolute relative error's CDF of the predictions produced by the ETL-2PC models.

only 40 nodes vs 80), due to its much higher level of data contention.

4.2.5.4.2 CTL validation. The validation of the CTL models has been performed on the PC-V and the FG-E infrastructures, and has been aimed at assessing the accuracy of the model to predict throughput and commit probability of transactions running according to the CTL concurrency control scheme and the 2PC replication protocol. Infinispan has been configured to employ its default consistent hash function. According to this function, primary owners and replicas of keys are scattered uniformly at random across nodes in the system: this is captured by setting, in the 2PC case, $L_O = P_O \frac{1}{N}$ in the model, with N being the number of nodes in the system.

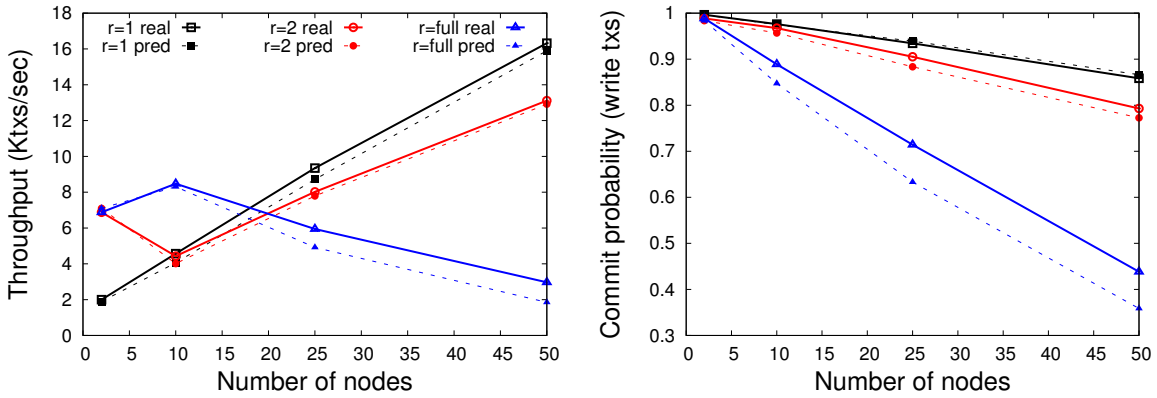
Specifically, this section assesses the accuracy of the CTL-2PC model in predicting the closed-system throughput and the abort rate of various YCSB workloads while varying the number of nodes in the system, the replication degree and the underlying cloud infrastructure.

Figure 4.12 shows the accuracy of the models in predicting the performance achieved with workloads characterized by very different scalability trends (with a replication degree fixed to 2). The model is able to predict that workloads A/F-1-Z-PC are network bound, and that their abort probability is negligible: in fact, though the abort probability of update transactions is negligible, linear scalability is hampered by the interactions needed to fetch remote data (for workload F, since it also performs a get operation) and to atomically commit transactions.



(a) Throughput (b) Commit probability (write txs)

Figure 4.12: Accuracy of the CTL-2PC model for different workloads ($r = 2$)

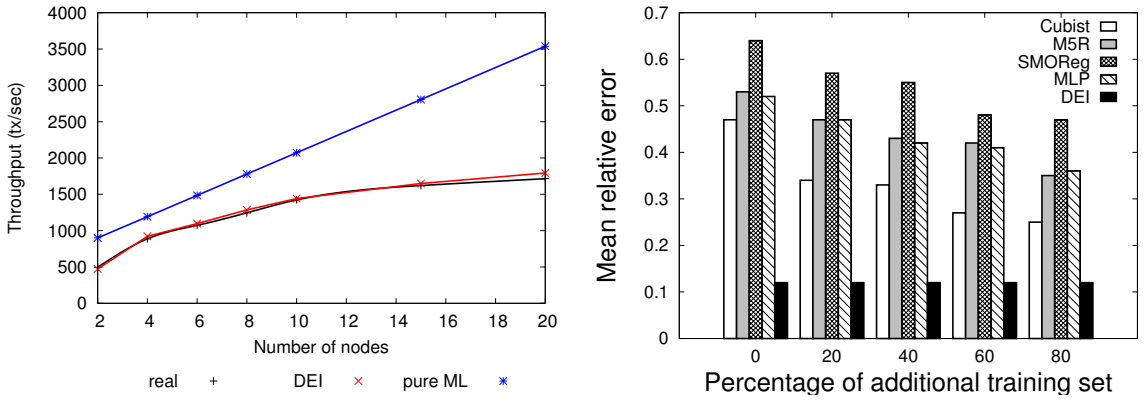


(a) Throughput (b) Commit probability (write txs)

Figure 4.13: Accuracy of the CTL-2PC model while varying r (B-5-H-PC)

Analogously, the model is able to predict that data contention is what limits the scalability of workloads B-5-H-PC and A-5-H-FG, as it leads system's performance to degrade as the degree of concurrency in the system grows.

Figure 4.13, instead, shows the model's capability of predicting the performance of an application when changing replication degree and scale. To this end, it shows the throughput and commit probability for workload F-5-H-PC while varying the scale and the replication degree. It is possible to see that a higher replication degree is better for small deployments (up to 15 nodes): in fact, though it yields, on average, a higher number of nodes to be contacted at prepare time, it reduces (or eliminates, in the case of full replication) the generation rate of remote get operations, whose cost is dominant at small scales for this workload. As the size



(a) ETL-2PC model vs ML on TPC-C (EC2)

(b) CTL-2PC model vs ML on YCSB (FG-E)

Figure 4.14: Comparing the *Divide et Impera* models with purely ML-based predictors.

of the system grows, however, the latency of the distributed commit phase at high replication degree becomes the dominant cost, and yields higher lock hold times that result into higher data contention; hence, a lower replication degree yields better overall performance. As shown in the plots, the proposed model is able to quantitatively capture the effect of the shift of replication degree and scale on performance: this suggests the viability of the proposed model not only for resource provisioning, but also for the self-tuning of distributed transactional in-memory platforms.

4.2.5.5 Comparison with pure ML approaches

This section is devoted at comparing the accuracy attained by the proposed models with the one achieved by off-the-shelf ML-based predictors, as proposed in several recent works in the area of automatic elastic scaling for DTPs (Chen et al., 2006; Ghanbari et al., 2007).

Figure 4.14a reports the result of an experiment aimed at carrying out such a comparison between the proposed ETL-2PC *Divide et Impera* model (noted *DEI*) and a black box performance model based on Cubist. To this end, Cubist has been trained on the TPCC-R workload, deployed on Amazon EC2, while varying the number of nodes from 2 to 20 and the incoming load (in closed loop with non-zero think time) from 100 requests per second until reaching 0 think time between subsequent requests issued by the same workload injector thread. The input features for the ML included features like the percentage of update transactions and the mean

number of locks they request, ACF, workload intensity, along with number of nodes and active threads per node. As in the previous evaluation study, closed-system throughput (with 0 think time) has been set as target KPI.

The test dataset, instead, is composed by TPCC-W samples, which generates a significantly higher data contention level with respect to TPCC-R. Further, unlike TPCC-R, TPCC-W exhibits a non-linear scalability trend. As expectable (Chen et al., 2006; Q. Zhang et al., 2007), in these conditions, the pure ML-based approach manifests its limits in terms of reduced accuracy when working in extrapolation. In fact, the plots in Figure4.14a clearly highlight that the pure ML-based solution tends to mimic the linear scalability trend that it has observed during its training phase. As a consequence, it blunders when faced with workloads (like the TPCC-W) that *i*) have previously unobserved input characteristics (e.g., in terms of ACF), and *ii*) exhibit significantly different performance trends.

This problem might be tackled to come extent by increasing the coverage of the training phase. However, achieving a good accuracy across a wide range of workloads may require a prohibitive increase of the ML training time. In fact, data contention dynamics in a (distributed) transactional systems are influenced by a wide range of parameters (Couceiro et al., 2015), and it is well known that the training time of ML techniques grows exponentially with the number of input features (the, so called, *curse of dimensionality* (Bishop, 2006)).

The analytical model employed by the proposed hybrid model, on the other hand, can exploit the *a priori* knowledge on the dynamics of data consistency mechanisms to achieve higher accuracy when working in extrapolation. Further, it allows for narrowing the scope of (and hence for simplifying) the problem tackled via ML techniques, reducing the dimensionality of the ML input features' space and, consequently, the duration of the training phase.

A similar behavior can be witnessed by analyzing the results of a comparative experiment carried out between the *Divide et Impera* CTL-2PC model (noted again *DEI*) and pure black box approaches, specifically, the same ML tools that have been experimented with when building and evaluating its black box network model. In the experiment, an initial training set and test set are defined: the former consists of the same knowledge base that has been used to build the network predictor of the hybrid CTL-2PC model, i.e., without data contention; the latter

is composed by YCSB workloads. The accuracy of the proposed model is compared, over all the workloads in the test set, with the accuracy of the ML techniques when progressively trained with additional, randomly selected, portions of the training set. Specifically, samples corresponding to the 20%, 40%, 60% and 80% of the test set are randomly selected, removed from the test set and added to the training set of the MLs (but not of the proposed model). The MLs are, hence, trained with the updated training set and they are evaluated over the remaining test set.

Figure4.14b reports the result of this evaluation, where each bar is the average of ten runs.As expectable, the mean relative error of the MLs decreases as more samples are added to the training set. However, the accuracy of the hybrid model with the original training set is still twice as high as the one achieved by the best performing learner trained with the 80% of additional training set (12% vs 25%).

This confirms that the hybrid modeling technique employed in the proposed models can produce reliable predictions requiring a smaller training set (and hence a lower training time) than pure ML-based approaches, even outperforming them in terms of accuracy.

4.2.5.6 Measurements overhead and models resolution time

To conclude the evaluation of the presented *Divide et Impera* DTPs performance models, this section reports the results of the experimental evaluation carried out to assess the overhead incurred by Infinispan to collect input parameters of the described models and their resolution time.

Overhead analysis. The proposed models require detailed information about the target application to forecast its performance: this information consists of CPU demands for basic operations (e.g., puts and gets), which can be profiled only once off-line, but also of workload-dependent characteristics (e.g., ACF), which need to be profiled at run-time to allow for the reconfiguration of the platform in the case of workload change.

The presented models are currently integrated in the Cloud-TM data platform (Romano

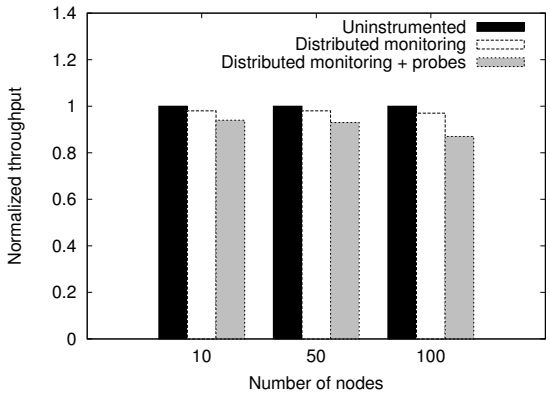
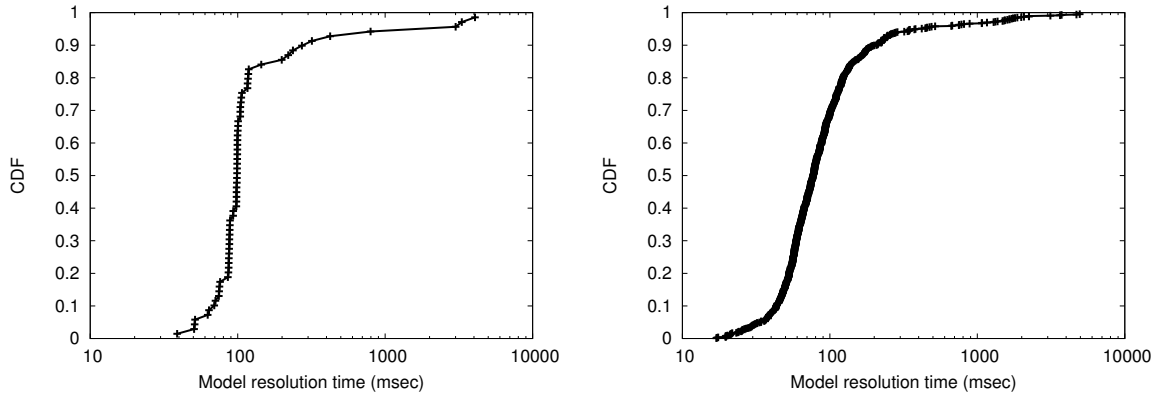


Figure 4.15: Monitoring overhead.

et al., 2010), and are responsible for its self-tuning (Didona & Romano, 2014c). Therefore, they can rely on the availability of the Cloud-TM workload characterization framework, which consists of two components: *i*) a set of probes that are integrated in Infinispan to collect detailed statistics on the transactional workload and on the usage of CPU/memory/network resources; and *ii*) the distributed monitoring framework used to aggregate statistics gathered from all the nodes in the system (Palmieri et al., 2011; Ciciani et al., 2012; Didona et al., 2014; Di Sanzo et al., 2015).

Figure 4.15 shows the throughput when activating only the distributed monitoring framework, and when activating also the probes for detailed statistics collection, normalized to the throughput achieved using a non-instrumented version. In order to evaluate the impact of these mechanisms in platforms of heterogeneous scales, deployments encompassing 10, 50 and 100 nodes have been considered.

The plots show that the overhead of the distributed monitoring framework remains unaltered (about 2%) as the size of the data platform varies; conversely, the one introduced by the probes grows with the number of nodes running the application, achieving a maximum of 12% for 100 nodes. Overall, the results show that the monitoring overheads remain largely acceptable even in very large scale systems. However, it should be noted that these overheads represent indeed an upper bound on the actual performance loss that would be introduced in case the probing system avoided tracing *every* transaction, and implemented a more efficient/optimized sampling strategy that collected statistics at a lower frequency (e.g., tracing only a percentage



(a) ETL-2PC. (b) CTL-2PC.

Figure 4.16: Model resolution time.

of the transactions (Trushkowsky et al., 2011)).

Models resolution time. To evaluate the resolutive algorithm presented in Section 4.2.4, this section reports experimental data corresponding to the time it takes to query the proposed models. For the sake of symmetry with previous sections, this evaluation is performed separately on two case studies: *i*) query time of the ETL-2PC model to predict Infinispan performance for the Radargun benchmark deployed over the FG-E infrastructure and *ii*) query time of the CTL-2PC model to predict YCSB performance on the PC-V infrastructure. In both cases the convergence thresholds on abort probabilities and on closed-system throughput computation has been set to 0.01; the minimum/maximum arrival rates have been set, respectively, to 10/100K tx/sec, and the fall-back exhaustive search algorithm has been set to operate at the granularity of 10 tx/sec. The models have been queried on a machine equipped with a 2.7 GHz Intel Core i7, 8GB of 1333 MhZ DDR3 RAM and running Mac OS X 10.7.5.

Figure 4.16 reports the results of this evaluation, by depicting the Cumulative Density Function of the resolution time of the models. The plots show that, in both cases, the 90-th percentile is less than 250 msec. The heavy tail in the CDFs is due to the resolution times of the model for scenarios characterized by performance degradation caused by excessive contention on data and high network latencies: solving the model for such scenarios, in fact, requires the employment of the exhaustive search algorithm introduced in Section 4.2.4, which is naturally more

computationally demanding.

Overall, the presented results confirm the viability of the proposed models not only to perform off-line capacity planning but also on-line self-tuning, by being able to quickly serve what-if queries.

4.3 Conclusion

This chapter has presented the *Divide et impera* gray box modeling technique, which enables the joint usage of white and black models targeting the performance prediction of distinct, but possibly inter-related, modules of the system being modeled.

Such technique has been applied to model the performance of applications deployed over DTPs, by implementing the following white/black box modeling dichotomy. On one side, white box modeling has been employed to capture the effects of CPU utilization and of the concurrency/replication protocol; black box modeling, on the other side, has been exploited to obtain a performance model of the network-related dynamics of the target platform.

The rationale underlying this design choice is that, by knowing the implementation of the internals of the DTP, it is possible to explicitly model how the deployed application interacts with processing resources and with the implemented protocols. On the other side, the virtualization layer adopted in Cloud environments and the complexity of the communication toolkits hide most of the details about the interactions between the application and the networking layer, thus making it cumbersome to derive a detailed white box model.

The performed experimental evaluation has confirmed the viability of the approach, and has shown that a *Divide et impera*-based performance model may be instantiated with less training samples than a pure black box learner, while also delivering higher accuracy.

5 The Hybrid Ensemble Approach

This chapter is devoted at describing and evaluating the Hybrid Ensemble performance modeling technique, which is based on the idea of combining the output of a set (i.e., an *ensemble*) of white and black box models with the purpose of generating a single model with a higher predictive accuracy than any of its constituent parts.

Specifically, a Hybrid Ensemble performance predictor relies on the availability of one or more base white box performance models for the target application and envisages to complement them with one or more black box models. The ultimate goal of this *ensemble* of models it to enhance the accuracy of the base predictors, correcting their possible inaccuracies by training black box models over samples gathered from the operational system.

In general, as discussed in Section 3.1.3, such a correction can be implemented according to different principles: this dissertation explores two of them, namely *Selection* and *Patching*. The former consists in employing the performance model (either white or black) that is expected to maximize accuracy depending on the configuration/workload whose performance is being predicted; the latter entails progressively learning how to improve the predictions generated by the white box models by correcting the errors of such models in various regions of their parameter space.

This dissertation presents two *Hybrid Ensemble* techniques that implement the *Selection* principle, namely *Hybrid KNN* and *Probing*, and two that embody the *Patching* principle, namely *Bootstrapping* and *Hybrid Boosting*.

The proposed performance modeling techniques are applied to the problem of DTPs performance forecasting by employing, as base white box predictors, the DTPs models proposed in the previous Chapter. Note that, despite the gray nature of these models, they can be used in the same guise of pure white box models to build *Hybrid Ensemble* predictors. Additional details

on how the *Divide et impera* DTPs performance predictors are employed as building blocks in *Hybrid Ensemble* performance models are going to be provided in Section 5.4.1.3.

In addition, this chapter also provides and evaluation of the proposed *Hybrid Ensemble* techniques using an additional case study, namely a Total Order Broadcast (TOB) primitive. TOB primitives incarnate a fundamental problem in distributed computing, namely the consensus one (Cachin et al., 2011), and represent a key building block in a number of DTPs (Couceiro et al., 2009; Pedone et al., 2003). This additional evaluation is performed using a pure white box predictor as base model in the ensemble, and allows for assessing the effectiveness of the proposed techniques when applied to two very different performance prediction tasks (as described in Section 5.4.1.3).

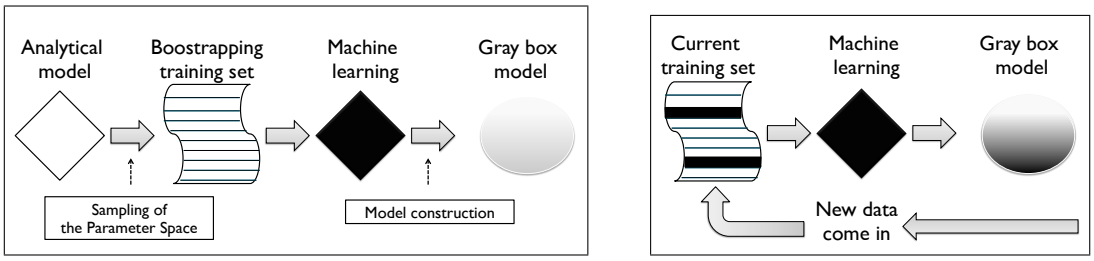
The remainder of this chapter is organized as follows: Section 5.1 introduces the notation employed in the description of the proposed techniques; next, the Hybrid Ensemble methodologies are described: first, in Section 5.2, the techniques implementing the *Patching* principle are presented, namely *Bootstrapping* and *Hybrid Boosting*; then, in Section 5.3, the *Selection*-based approaches are described, namely *Hybrid KNN* and *Probing*; Section 5.4 is dedicated to evaluating the four proposed techniques over the two aforementioned case studies; finally, Section 5.5 concludes the chapter.

5.1 Notation

This section presents the notation employed throughout the discussion to describe the proposed Hybrid Ensemble techniques. The Greek letter Γ represents, in general, a performance predictor: the $i-$th white, resp. black, box performance predictor in a hybrid ensemble is noted Γ^i_{WB}, resp. Γ^i_{BB}.

In the case that a specific hybrid ensemble method does not encompass the simultaneous co-existence of more than one white or black box model, then the single instance of the given modeling methodology will be referred to without the need of the indexing superscript.

Moreover, performance models regarded as white box are treated as immutable, i.e., the function they encode is not dependent on any amount of available training data. On the other



(a) Initialization phase. (b) Updating

Figure 5.1: Main phases of the Bootstrapping technique.

hand, black box predictors are trained starting from available data and a learning algorithm: the learning algorithm corresponding to a Γ^i_{BB} is noted γ^i_{BB}.

Finally, sets of elements, which can be either models or learning algorithms, are noted with the letter Ω: whenever necessary, the subscript WB, resp. BB, is used to indicate that the set refers to a collection of white, resp. black, box models; if needed, additional superscripts are used to discriminate between a set of black box predictors and the set of corresponding learning algorithms that are used to train them. The superscript Γ is employed in the former case, and γ in the latter.

Additional, technique-specific, notation needed to simplify the presentation will be employed, and properly introduced, whenever necessary.

5.2 Patching-based Hybrid Ensemble Techniques

This section describes the proposed Hybrid Ensemble techniques that embody the *Patching* principle: according to this principle, black box techniques are applied on the output of a base white box predictor so as to detect and ultimately correct its prediction inaccuracies.

Specifically, Section 5.2.1 presents the Bootstrapping technique, which initially trains a black box learner over the output of a base white box one and then incorporates in the training set factual knowledge about the actual performance of the target system; Section 5.2.2, instead, describes the Hybrid Boosting technique, which employs black box techniques to learn the error distribution of the provided white box model in order to generate a complementary function that compensates for its inaccuracies.

Algorithm 1 Bootstrapping main loop

```
1:  Algorithm γ                                          ▷ Black box learning algorithm
2:  Model Γ_BB                                           ▷ Black box model ultimately used predict performance
3:  Model Γ_WB                                           ▷ White box model used to bootstrap the black box one
4:  function MAIN(Algorithm γ, Model Γ_WB, int initSize, double ε)
5:      DataSet ST = initKB(γ, Γ_WB, initSize, ε)        ▷ Generate the synthetic training set
6:      Γ_BB = γ(ST)                                     ▷ Train the ML over the synthetic training set
7:      while true do
8:          DataSet D = collectSamples()                 ▷ Collect samples at runtime
9:          updateKB(ST, D)                              ▷ Incorporate the new samples in the knowledge base
10:         Gamma_BB = γ(ST)                             ▷ Re-train the ML over the updated training set
11:     end while
12: end function

13: function QUERY(Configuration x)
14:     return Γ_BB(x)
15: end function
```

5.2.1 Bootstrapping

This section describes the Bootstrapping technique in a top-down fashion: first, a specification of the technique is presented, in which several relevant building blocks are encapsulated into abstract primitives. Next, Sections 5.2.1.1 and 5.2.1.2 shall discuss in detail the key parametric and algorithmic trade-offs associated with each of these primitives.

As reported in the pseudo-code of Algorithm 1, the Bootstrapping technique consists of two main phases: the initialization of the black box model based on the predictions of the analytical one (Lines 5-6), and its update, which is performed every time that new samples from the running application become available (Lines 7-11).

The initialization phase, depicted in Figure 5.1a and detailed in Section 5.2.1.1, is composed, in its turn, of two steps:

i) *generation of the synthetic training set (Line 5):* a subset T of the white box's parameter space (PS) is generated and is used to bootstrap the knowledge base of a black box learning algorithm. As already mentioned, the number of samples in PS that are necessary to characterize an arbitrary function defined over such space grow, in the worst case, exponentially with the dimensionality of PS. A first challenge addressed in this step is, thus, to determine *which* samples to include in the initial synthetic training set in order to have a sufficient coverage of the whole parameter space.

Once T has been obtained, the white box model is queried to compute a prediction of the performance of the application for each of its elements. The output of this phase is a new set

Algorithm 2 Initialization phase

```
1: function INITKB(Algorithm γ, Model Γ_WB, int initSize, double ε)
2:     double error                                   ▷ Fitting error of ml over Γ_WB's function
3:     int currSize = initSize                        ▷ Current size of the synth. training set
4:     do
5:         Set T = SampleConfigSpace(currSize)        ▷ Training configurations
6:         DataSet ST = ∅                             ▷ AM-based training set
7:         for all x ∈ T do
8:             ST = ST ∪ {x, Γ_WB.query(x)}           ▷ Query the analytical model
9:         end for
10:        error = estimateFittingError(ST, γ)        ▷ Evaluate γ's fitting error over Γ_WB
11:        currSize = nextSize()                      ▷ Select a new value for the size of T
12:    while (!isAccurate(error, ε))                  ▷ Ensure ml has learnt am's function
13:    return ST
14: end function
```

ST, whose elements are tuples of the form $< x, \Gamma_{WB}.query(x) >$, where $x \in T$ and $\Gamma_{WB}.query(x)$ is the corresponding prediction computed by the white box predictor. This step will be detailed in Section 5.2.1.1;

iii) *black box model construction (Line 6)*: the black box algorithm is trained on ST and produces a statistical model of the application's performance; note that the training process can be based on alternative algorithms, e.g., Decision Trees (DT), Artificial Neural Networks (ANN) and Support Vector Machines (SVM) (Bishop, 2006).

The update phase, shown in Figure 5.1b, consists of 3 steps:

i) *collection of real samples (Line 8):* a new dataset D of $< x, perf(x) >$ tuples is collected, where x is a configuration/workload of the target application, and $perf(x)$ is the real performance, i.e., measured on the live system, corresponding to x. Such factual knowledge can either be spontaneously originated by the on-line production system (e.g., corresponding to workloads that are generated by end users), or can be collected during a dedicated off-line training phase, thus steering the workloads and configurations to experiment with;

ii) *update of the training set (Line 9)*: the ST set is updated in order to incorporate knowledge represented by the samples in D. In general, this operation can be performed in several ways, which will be discussed in Section 5.2.1.2;

iii) *black box model update (Line 10)*: the black box learning algorithm is trained on the updated ST and outputs a new application performance model.

5.2.1.1 Synthetic Knowledge Base Initialization

The first step of the Bootstrapping technique is embodied by the INITKB function, whose pseudo-code is reported in Algorithm 2. This function iteratively performs two main operations. The first one consists in selecting a subset T of samples from the whole space of possible configurations for the application. The second one consists in generating a "synthetic" training set ST by exploiting the predictions output by the white box model for each of the elements in T. The goal of the function is to output a synthetic training set ST that is representative of the target performance function to be modeled, i.e., such that the black box learner trained over it is able to accurately represent the performance function embedded in the base white box model.

The *initKB* function takes as input a white box model of the target application (Γ_{WB}), a black box learning algorithm (γ), an initial value for the size of the synthetic training set (*init*) and a threshold value (ε). Then, proceeding iteratively, it aims to find a value x such that training γ over a synthetic training set of size x produces a black box model which well approximates Γ_{WB}.

To this end, the *initKB* function relies on 4 primitives. Their high-level functionalities are described in the following: their implementation is, instead, provided in Section 5.4.2.1, which presents the evaluation study.

•SAMPLECONFIGSPACE: it determines *which* samples of the feature space to include in the synthetic training set, given its size. This function can embody arbitrary sampling strategies, based, e.g., on random sampling or Active Learning (Settles, 2009).

•ESTIMATEFITTINGERROR: it estimates how much the performance model obtained by training γ over T is similar to the one embedded by Γ_{WB}. Also this primitive lends itself to several possible instantiations, e.g., leave-one-out or cross-validation (Arlot & Celisse, 2010).

•ISACCURATE: it returns *true* if the model obtained from γ approximates Γ_{WB} sufficiently well; *false* otherwise.

•NEXTSIZE: it determines the size of the set to sample from the whole parameter space at the next iteration. The *initKB* function basically aims at minimizing the difference between Γ_{WB}'s and the black box model's predictions as a function of ST's size. Therefore, this primitive can

Algorithm 3 Update phase

1: **function** UPDATEKB(DataSet D)
2: $setWeight(D, w)$ ▷ *Set the weight to the new samples*
3: **function** update = any function in {MERGE,RNN,RNR,RNR2}
4: $update(D)$;
5: $\Gamma_{BB} = \gamma(ST)$ ▷ *Retrain the ML with the new dataset*
6: **end function**

7: **function** MERGE(DataSet D)
8: $ST = ST \cup D$ ▷ *Add the real samples*
9: **end function**

10: **function** RNN(DataSet D)
11: **for all** $(x,y) \in D$ **do**
12: $(x_r, y_r) = \underset{(x',y') \in ST}{\text{argmin}} \{dist(x', x)\}$ ▷ *Find the NN*
13: $ST = ST \setminus \{(x_r, y_r)\}$ ▷ *Remove the NN*
14: $ST = ST \cup \{(x,y)\}$ ▷ *Insert the real sample*
15: **end for**
16: **end function**

17: **function** RNR(DataSet D, double c)
18: *DataSet* $D\_NR = \emptyset$ ▷ *Temporary NN set*
19: **for all** $(x,y) \in D$ **do**
20: $D\_NR = \{(x_t, y_t) \in ST : dist(x, x_t) \leq c \wedge isSynthetic(x_t, y_t)\}$
21: $ST = ST \setminus D\_NR$ ▷ *Remove the NNs*
22: **for all** $(x,y) \in D$ **do**
23: $ST = ST \cup \{(x,y)\}$ ▷ *Add real samples*
24: **end for**
25: **end for**
26: **end function**

27: **function** RNR2(DataSet D, double c)
28: *DataSet* $D' = D$ ▷ *Temporary set of real samples still unmatched*
29: **for all** $(x_t, y_t) \in ST$ **do**
30: $(x_r, y_r) = \underset{(x,y) \in D}{\text{argmin}} \{dist(x_t, x)\}$ ▷ *Find the NN*
31: **if** $dist(x_r, x_t) \leq c \wedge isSynthetic(x_t, y_t)$ **then**
32: $ST = ST \setminus \{(x_t, y_t)\}$ ▷ *Remove the NN*
33: $ST = ST \cup \{(x_t, y_r)\}$ ▷ *Add NN with modified output*
34: $D' = D' \setminus (x_t, y_t)$ ▷ *Remove the real sample from the temp. set*
35: **end if**
36: **end for**
37: **for all** $(x,y) \in D'$ **do**
38: $ST = ST \cup \{(x,y)\}$ ▷ *Add unmatched samples*
39: **end for**
40: **end function**

implement different search techniques to identify a cardinality of ST such that, under the provided sampling algorithm, the obtained black box model is similar to the provided white box one, e.g., iterative or binary search.

These primitives are employed in the *initKB* function as follows: *i*) sample the parameter space via the *sampleConfigSpace* primitive, to obtain a set T of cardinality *currSize* (Line 5); *ii*) for each element in T, query Γ_{WB} to obtain the corresponding performance prediction, and generate the synthetic set ST (Lines 6-9); *iii*) train γ over ST and evaluate its accuracy in predicting the performance function encoded by Γ_{WB}, by means of the *isAccurate* primitive

(Line 10); *iv*) if the fitting error is less than a given threshold ε, then return ST; else determine the next value for *currSize* and go to step *i*) (Lines 11-13)[1].

Note that, given a sampling algorithm, the cardinality of ST plays a role of paramount importance in determining the effectiveness of the Bootstrapping methodology. It represents, in fact, a key trade-off between the accuracy with which the function encoded by the white box model can be approximated by the black box learner, and the effectiveness with which the latter can incorporate new knowledge deriving from the availability of samples collected from the operational system.

Reducing the number of samples can, in general, yield several benefits. These include reducing the duration of the initial training phase of the black box learner; also, it may favor the subsequent update phase of the training set: the lower the number of synthetic samples, the higher the relative density of the real samples in the updated training set. This can reduce the time it takes for the real samples to outweigh the synthetic ones, and correct possible errors of the white box model.

However, using a lower number of synthetic samples also yields the black box model to approximate more coarsely the original white box one, which may degrade accuracy. On the other hand, a very large training set provides more detailed information to the black box learner on the function embodied by the white box model, and can favor a better approximation of such function. However, it comes with the downside of an increased training time and may induce a longer transient phase before runtime samples can take over synthetic ones.

Note that the initial training phase of the black box model is performed over the output of Γ_{WB} on a sampling of the *whole* parameter space PS of the target application. Hence, even if provided only with a set R of real samples corresponding to narrow regions of PS, the bootstrapped learner still inherits the predictive power of the base white box model when working in extrapolation with respect to R (as evaluated in Section 5.4.2.3).

[1]For simplicity, we do not show how to handle cases in which the fitting error never goes below the ε. Coping with this case could simply entail returning the ST that minimizes the error after a given number of attempts.

5.2.1.2 Update of the Knowledge Base

The UPDATEKB function, reported in Algorithm 3, is the core of the Bootstrapping methodology, as it allows for the incremental refinement of the initial performance model. This function is responsible for incorporating real samples coming from the running application into the initial synthetic training set, thus allowing the black box model to gradually correct inaccurate performance predictions by the white box model.

The UPDATEKB function takes as input the dataset D containing new samples and injects them into the current training set. The key issue here is that the new samples contained in D may contradict the synthetic samples generated by the AM that are in the training set. This happens whenever D contains samples belonging to regions of the feature space in which the AM achieves unsatisfactory accuracy: in such a case, in fact, the AM generates outputs (i.e., performance predictions) that may differ significantly from the corresponding values in D (i.e., having similar or identical input mapped to different output). In this work, we consider two techniques that aim at reconciling possible divergences between synthetic and actual samples: *weighting* and *replacing*.

Weighting is a well-known and widely employed technique in the ML area (Cost & Salzberg, 1993): the higher the weight for a sample, the more the ML will try to minimize the fitting error around it when building the statistical model. In the Bootstrapping case, weighting can be used to suggest the ML to give more relevance and trust to real samples than to synthetic ones. The replacing approach consists in removing preexisting "close enough" (synthetic) samples from the training set, whenever new real samples are incorporated. We consider four implementations of the UPDATEKB function, which incorporate new knowledge according to different principles.

1) Merge. This is the simplest considered variant, and it consists in adding the new samples to the existing set *ST* (lines 7-9). This implies the possible co-existence of real and synthetic samples that map very similar (or equal) input features to very different performance. Hence, the use of weights is the only means to induce the ML to give more importance to real samples over (possibly contradicting) synthetic ones.

2) Replace based on Nearest Neighbor (RNN). This variant consists of two steps, which are

repeated for each element (x,y) in D: i) find the element (x_r,y_r) that is closest (according to a distance function) to (x,y) in ST (line 12) and ii) replace (x_r,y_r) with (x,y) (lines 13-14). Also in this case the newly injected sample is given a weight w. Note that, once an element from D is inserted in ST, it becomes eligible to be evicted from the set, even in favor of another sample contained in D itself. This algorithm aims at progressively replacing all the synthetic samples from ST with real ones; by switching a real sample with its nearest neighbor in ST, moreover, this algorithm aims at keeping unchanged the density of samples in ST.

3) Replace based on Nearest Region (RNR). This algorithm represents a variant of RNN. A first difference is that, in order to avoid "losing" knowledge gathered from the running system, RNR policy only evicts synthetic samples from the training set. Moreover, instead of replacing a single sample in ST, a sample in D replaces all the ones in ST whose distance from it is less than a given cut-off value c. If a sample in D does not replace any sample in ST, it is added to ST, as it is considered representative of a portion of the feature space that is not covered by pre-existing elements in ST. On one side, this implementation speeds up the process of replacement of synthetic samples with real ones; on the other side, depending on the density of the samples in ST and on the cut-off value, it may cause imbalances in the density of samples present in the various regions of the feature space for which T contains information. In fact, a single sample from D may potentially take the place of many others in ST.

4) Replace based on Nearest Region (RNR2). This algorithm represents a variant of RNR. Also RNR2 policy, in fact, only evicts synthetic samples from the training set; however, it differs from RNR in the way samples corresponding to actual measurements are incorporated in the training set. For each element $(x,y) \in ST$, the closest neighbor $(x_r,y_r) \in D$ is found (line 29): if the distance between the two is less than a cut-off value c (line 30), then the output relevant to x is changed from y to y_r (lines 31-32). Like in RNR, if a sample in D does not match any sample in ST, it is added to ST. This implementation inherits from RNR the speed in replacing samples in ST with real, new ones, but avoids its downside of changing the density of samples in ST: instead of removing samples from ST, for each element (x_r,y_r) in D, the target value of all the points in the training set for which it is nearest neighbor and within distance c is approximated with y_r.

Algorithm 4 Hybrid Boosting

Set $\Omega^{\gamma}_{BB,R} = \{\gamma^1_R, \ldots, \gamma^M_R\}$ ▷ *ML regressors for residue prediction*
Set $\Omega^{\Gamma}_{BB,R} = \{\Gamma^1_{BB,R}, \ldots, \Gamma^M_{BB,R}\}$ ▷ *Black box models for residue prediction.*
Set $\Omega^{\Gamma}_{P} = \{\Gamma^0_P, \ldots, \Gamma^M_P\}$ ▷ *Models for overall performance prediction*

function INIT(White Box model Γ_{WB}, Data Set D_{tr})
    $\Gamma^0_P = \Gamma_{WB}$ ▷ *Set Γ_{WB} as the first predictor in the chain.*
    for $m = 1 \rightarrow M$ **do**
        $D_m = \emptyset$
        for each $<\mathbf{x}_n, y_n> \in D_{tr}$
        $y_{m,n} = y_n - \Gamma^{m-1}_P(x_n)$ ▷ *Compute the residual error of the previous performance predictor*
        $D_m = D_m \cup < x_n, y_{m,n} >$
        end for each
        $\Gamma^m_{BB,R} = \gamma^m_R(D_m)$ ▷ *Train on the residuals.*
        $\Gamma^m_P = \Gamma^{m-1}_{BB,R} + \Gamma^m_{BB,P}$ ▷ *Set the m-th predictor*
    end for
end function

function FORECAST(x_s)
    return $\Gamma^0_P(\mathbf{x}_s) + \sum_{m=1}^{M} \beta_m \Gamma^m_{BB,R}(x_s)$
end function

5.2.2 Hybrid Boosting

The Hybrid Boosting (HyBoost) is based on a well-known technique from the literature on ensembles of black box learners, known as Boosting (Bishop, 2006), ad adapts it in order to support the joint usage of white and black box models. In a nutshell, the Boosting technique aims at building one *strong* learner by training a set of *weak* learners in an iterative fashion, i.e., having each of such learners to compensate for the prediction error of its predecessor. In particular, the starting point for such hybrid methodology is the *Adaptive Logistic Regression* technique.

The HyBoost technique, as reported in Algorithm 4, exploits the availability of a single white box model Γ_{WB} and a set of M black box ones. The possibility of exploiting only a single white box model stems from the fact that, as hinted, the Boosting paradigm entails a learner to be trained according to the error of the previous one. Since a white box model is immutable by definition, it is not possible to change the function that is encodes depending on the behavior of other models. Black box regressors, conversely, can compensate for the error of another model, by being trained over the error exhibited by another one (called residue) instead of targeting the prediction of the target performance function.

The key intuition at the basis of Boosting, when applied to black box learners, is that it may be easier to learn the prediction error of a given ML algorithm trained on a target function, than learning the target function itself. The HyBoost extends this concept also to the case of white box models, entailing the iterative refinement of the predictions of a white box model by means of sequential black box corrections.

More in detail, HyBoost uses two (ordered) sets of predictive models, noted $\Omega^{\Gamma}_{BB,R}$ and Ω^{Γ}_{P}, composed by, respectively, m and $m+1$ models. The single white box model Γ_{WB} is set to be the model with index 0 in Ω^{Γ}_{P} (Line 6). The model with index 1 in Ω^{Γ}_{P}, instead, is obtained by training the first regressor γ^{1}_{R} with the residual training set D_0, which characterizes the absolute prediction error of Γ_{WB} for each point in the original training set D_{tr}. Any other model $\Gamma^{i}_{BB,R}$, with $i \in [2,M]$, is trained over the residual prediction errors of the model $\Gamma_{BB,P}i-1$, which incorporates the knowledge of the white box model and of the first $i-1$ ML-based learners by means of the following recurrence equation (Line 14) [2]:

$$\Gamma^{m}_{P} = \Gamma^{m-1}_{P} + \Gamma^{m}_{BB,R}. \tag{5.1}$$

Note that, analogously to Bootstrapping, also Hybrid Boosting can exploit black box models based on different learning algorithms. Moreover, it may be further extended and optimized using well-known techniques in the literature on boosting ML algorithms, such as adaptively weighting the elements in the training set of the i-th learner in order to focus it on minimizing its fitting error on samples over which the $i-1$-th learner incurred the largest errors (Schapire, 1999).

5.3 Selection-based Hybrid Ensemble Techniques

This section presents the proposed Hybrid Ensemble techniques that implement the *Selection* principle: unlike the *Patching* one, according to this principle, several white and black box

[2] In another work (Didona et al., 2013b) we also explore a multiplication-based variant of HyBoost: the residual error of a model over a sample is not the absolute difference between the real value r and the predicted one p, but the ratio $\frac{r}{p}$. In that case, Equation 5.1 takes the form $\Gamma^{m}_{P} = \Gamma^{m-1}_{P} \cdot \Gamma^{m}_{BB,R}$

Algorithm 5 Hybrid K Nearest Neighbors

Set $\Omega = \emptyset$ ▷ *Set of models to use*
Set $\Omega_{WB} = \Gamma^1_{WB}, \ldots, \Gamma^N_{WB}$ ▷ *Set of white box models*
Set $\Omega^{\gamma}_{BB} = \{\gamma_1, ..., \gamma_M\}$ ▷ *Set of ML regressors*
Set $D_{val} = \emptyset$ ▷ *Validation set*

function INIT(Set Ω_{WB}, Training Set D_{tr})
    $\Omega = \{\Omega_{WB}\}$ ▷ *Initialize with white box models*
    Set $D_{regr} = RandomSampling(D_{tr})$ ▷ *Build the training set for ML regressors*
    $D_{val} = D \setminus D_{tb}$ ▷ *Build a validation set (disjoint w.r.t. D_{regr})*
    for $m = 1 \rightarrow M$ **do**
        $\Gamma^m_{BB} = \gamma_m(D_{tb})$ ▷ *Train m-th regressor*
        $\Omega = \Omega \cup \{\Gamma^m_{BB}\}$
    end for
end function

function FORECAST($\mathbf{x}_s$)
    Set D_k=$\{<\mathbf{x}_i,\mathbf{y}_i> \in \text{KNN}(\mathbf{x}_s, D_{val})$ s.t. $||\mathbf{x}_i,\mathbf{x}_s|| < c\}$
    for each $\Gamma_i \in \Omega$ **do**
      ERR[i] = compute prediction error of model Γ_i on set D_k
    end for each
    $\mu = \underset{i=1...M+N}{\text{argmin}} \; ERR[i]$ ▷ *Find learner with lowest RMSE*
    return $\Gamma_\mu.query(\mathbf{x}_s)$
end function

performance models are maintained in parallel and, depending on the input sample, only the model that is supposed to minimize the predictive error is employed.

More in detail, Section 5.3.1 presents the Hybrid KNN technique, in which the expected most accurate predictor is determined by evaluating the accuracy of the available models on training samples that are similar to the input one; Section Section 5.2.2, instead, describes the Probing technique, in which the predictor to employ is determined by relying on an arbitrary classifier.

5.3.1 Hybrid K Nearest Neighbors

The key idea at the basis of the Hybrid K Nearest Neighbor (KNN) ensemble technique is to train several models independently and to use, depending on the incoming query, the one with the lowest expected predictive error. In particular, in order to estimate the error in predicting the performance for a new sample x, this technique evaluates the accuracy of the available models on a set of samples D_k that are as similar as possible to x.

As detailed in the pseudo-code in Algorithm 5, the KNN technique encompasses the presence of an arbitrary number of both white and black box performance predictors. Such predictors can be selected according to different principles, but in general their choice should promote model diversity (Dietterich, 2000). On the black box side, this can be accomplished, for example, by considering different ML algorithms and corresponding parameterization; similarly, on the white box side, diversity can be achieved by encompassing predictors relying on different methodologies, e.g., Analytical Modeling and Simulation.

The KNN algorithm is initialized via the INIT function, by providing the set of initial white box models and a set of data samples, $D_{tr} =< \mathbf{x}_i, y_i >$, which convey information on the performance $y_i \in C$ of the target system over a set of observed configurations $\mathbf{x}_i \in F$. The data set D_{tr} is not entirely used to train the set Ω^{Γ}_{BB} of regressors: conversely, D_{tr} is split into two disjoint data sets, namely a training and a validation set, noted, respectively, D_{regr} and D_{val}.

D_{regr} is used as training set for the learning algorithms in Ω^{γ}_{BB}, and is obtained by extracting a random subset amounting to a percentage p_{regr} of D_{tr}. D_{val} is obtained as the complementary subset of $D_{regr} \in D_{tr}$, which ensures the disjointness of the two sets D_{regr} and D_{val} by construction. The role of D_{val} and the importance of having disjoint training and validation set is explained in the following.

The D_{val} set is used at query time (function FORECAST), when the *Hybrid Ensemble* model is queried to predict the expected performance of the target system, noted y_s, in the configuration $\mathbf{x}_s$. To this end, it is first computed the set D_k that contains the k nearest training samples $\{\mathbf{x}_1,\ldots,\mathbf{x}_k\} \in D_{val}$ within distance c from point $\mathbf{x}_s$. The samples in D_k, for which the corresponding actual performance is known, are then used to compute the average accuracy of each of the models in the set Ω (Line 18). This allows for identifying the model, noted Γ_{μ} in the pseudo-code (Line 20), which is expected to maximize prediction accuracy in the region surrounding $\mathbf{x}_s$. Based on this geometric interpretation, the c parameter can be interpreted as a cut-off threshold, which allows for discarding samples of the validation set that are considered too far away from $\mathbf{x}_s$ and which, thus, may not be representative of the target configuration $\mathbf{x}_s$.

The relevance of ensuring the disjointness of D_{val} and D_{tr} is now clear: estimating the accuracy of a black box performance model Γ^{i}_{BB} using samples that were used during its training

phase could potentially lead to a significant overestimation of its accuracy. On the other hand, by ensuring $D_{val} \wedge D_{tr} = \emptyset$, evaluating the accuracy of Γ^i_{BB} over any subset of D_{val} corresponds to assess the generalization error of Γ^i_{BB} for previously unseen inputs. Therefore, the expected prediction error for a new sample x can be inferred by evaluating the accuracy of Γ^i_{BB} in predicting performance corresponding to previously unknown sample in D_{val} that are similar to x, i.e., D_k.

5.3.2 Probing

Similarly to the KNN case, the last of the proposed hybrid ensemble technique, named *Probing*, aims at serving a specific query with the model that is expected to minimize the prediction error. However, Probing presents two main distinctive features with respect to KNN. First, Probing encompasses the combination of a set of white box models with only one black box regressor, which is intended to be highly specialized in predicting the target performance function in the specific regions of the inputs space where none of the available white box models proves to be sufficiently accurate. Second, Probing relies on a generic classifier to determine which underlying model is going to be used depending on the query, instead of estimating models' accuracy over a set of neighbors.

More specifically, as detailed in Algorithm 6, the Probing technique uses a set Ω_{WB} of white box learners, that are built independently; a classification algorithm, noted γ_{cls}, to learn *where* (i.e., in which regions of the feature space) the provided white box models are not sufficiently accurate (based on a parametric threshold c over the absolute percentage error); a regression algorithm, noted γ_{reg} which is trained to learn the performance function of the target systems exclusively in the regions in which the white box models do not achieve adequate accuracy.

The initialization phase of a Probing-based predictor, summarized by the *INIT* function in Algorithm 6, encompasses the labeling of each sample $\mathbf{x}_i$ in the training set depending on the accuracy achieved by the white box models in predicting its corresponding performance. A sample $\mathbf{x}_i$ is labeled as belonging to class k if the $k-$th white box model in Ω_{WB} is the one achieving the lowest relative prediction error for $\mathbf{x}_i$ and such error is lower than the c threshold (Lines 11-12). If no such white box model in Ω_{WB} exists, then $\mathbf{x}_i$ is labeled as belonging to the

special class 0, corresponding to the black box learner; at the same time, the tuple $< \mathbf{x}_i, y_i >$ is added to the training set of the black box learning algorithm γ_{reg} (Lines 14-15 of the pseudo-code). At the end of the initialization phase, both the classification and the regression algorithm are executed on their respective training sets. As a result, a classifier Γ_{cls} and a regressor Γ_{reg} are built: the former determines which is the predictive model to be used to serve a specific query; the latter is intended to predict performance for configurations similar to the ones for which no white box models has proven to be sufficiently accurate.

The prediction phase corresponding to an input $\mathbf{x}_s$, is summarized in the *xx* function of Algorithm 6: first, Γ_{cls} identifies the predictor that is expected to be the most accurate in predicting performance corresponding to $\mathbf{x}_s$ (Line 22); then, such predictor is invoked to serve the query (Lines 23-26).

The intuition underlying the Probing technique is that, if the errors of the available white box learners are concentrated in restricted and easily identifiable regions of the input space (via Γ_{cls}), it is, then, possible to specialize the training phase of a black box learner exclusively on those regions. As a result of narrowing the domain of the performance function to be learnt by γ_{reg}, the accuracy of the resulting Γ_{reg} may be increased.

5.4 Evaluation

This section is devoted to assess the effectiveness of the four proposed Hybrid Ensemble techniques by means of an extensive experimental evaluation based on the two selected case studies, namely the Infinispan DTP and the Total Order Broadcast primitive of the Appia Group Communication Toolkit.

In particular, this section is organized as follows: Section 5.4.1 provides details about the experimental test bed; in addition, Section 5.4.1.3 provides some additional details about the aim and the execution of the evaluation process; Section 5.4.2 to Section 5.4.3 evaluate the four techniques alone, focusing on assessing their sensitivity to the amount of available training data and settings of internal parameters; finally, Section 5.4.6 provides a mutual comparison of the techniques.

Algorithm 6 Probing

Set Ω_{WB} ▷ *Set of white box models*
Classification algorithm γ_{cls} ▷ *Detects when no white box model in Ω_{WB} is accurate enough*
Classifier Γ_{cls} ▷ *Built by training γ_{cls}*
Regression algorithm γ_{reg} ▷ *Learns the performance function in areas where models in Ω_{WB} are inaccurate*
Regressor Γ_{reg} ▷ *Built by training γ_{reg}*
Set $D_{reg}, D_{cls} = \emptyset$ ▷ *Initialize training data sets*

function INIT(Set Ω_{WB}, Data Set D_{tr})
  for each $<\mathbf{x}_n, y_n> \in D_{tr}$
  $k = argmin_{i=1...|\Omega_{WB}|}\{AVG\_ERR(\Gamma^i_{WB}, <\mathbf{x}_n, y_n>)\}$ ▷ *Find white box model with lowest predictive error*
  if $|(y_n - \Gamma^i_{WB}(\mathbf{x}_n))/y_n| \leq c$ **then** ▷ *The k−th white box model is accurate enough for the sample*
    $D_{cls} = D_{cls} \cup \{< x_n, k >\}$
  else ▷ *None of the white box models is accurate enough for the sample*
    $D_{cls} = D_{cls} \cup \{< x_n, 0 >\}$ ▷ *0 is the index corresponding to Γ_{reg}*
    $D_{red} = D_{reg} \cup \{< x_n, y_n >\}$
  end if
  end for each
  $\Gamma_{cls} = \gamma_{cls}(D_{cls})$ ▷ *Train the classifier*
  $\Gamma_{reg} = \gamma_{reg}(D_{bad})$ ▷ *Train the regressor*
end function

function FORECAST($\mathbf{x}_s$)
  $k = \Gamma_{cls}(x_s)$ ▷ *Identify the predictor with lowest expected predictive error*
  if $k == 0$ **then**
    return $\Gamma_{reg}(\mathbf{x}_s)$
  else
    return $\Gamma^k_{WB}(\mathbf{x}_s)$
  end if
end function

5.4.1 Experimental test-bed

This section is devoted at describing the experimental test-bed used to validate the effectiveness of the proposed Hybrid Ensemble techniques. As already mentioned, the empirical assessment of such techniques is performed using two case studies: throughput prediction of applications deployed over the Infinispan DTP and message delivery time in a Total Order Broadcast (TOB) primitive.

The DTP case study has been introduced in Chapter 2, and has been widely discussed in Chapter 4. The following section is, thus, dedicated to introducing the TOB case study; Section 5.4.1.2, instead, provides additional information on the performance models employed as base predictors integrated into Hybrid Ensemble models; finally, Section 5.4.1.3 discusses the relevance of the two employed case studies. This last section especially highlights why they

can be considered as representative of a broad class of applications and, thus, ideal candidate to validate the effectiveness of the proposed Hybrid Ensemble techniques in different scenarios.

5.4.1.1 Total Order Broadcast Overview

TOB introduction. The TOB primitive allows a group of distributed processes to reach agreement on a common order of delivery of messages in presence of concurrent broadcasts by any process of the group. TOB represents a key building block at the basis of a number of fault-tolerant replication mechanisms for databases (Pedone et al., 2003), transactional memory (Couceiro et al., 2009) and highly-available objects (Meling et al., 2008) and several solutions to model and self-tune its performance have been proposed in literature, based on white box (Santos & Schiper, 2013), black box (Couceiro et al., 2011; Didona, Carnevale, et al., 2012) and even gray box techniques —specifically, on-line Bootstrapping (Romano & Leonetti, 2012).

The case study considered in this thesis, particularly, is a Sequencer-based implementation of TOB (STOB) (Cachin et al., 2011), implemented in the Appia Group Communication toolkit (Miranda et al., 2001); STOB algorithms represent a particularly attractive class of consensus protocols as they achieve the minimum bound on message latency for these types of problem (Lamport, 2003).

The message patterns generated by STOB algorithms to reach consensus is similar to the one of the Paxos algorithm (Lamport, 1998). This class of algorithms, in fact, relies on a special node, called *sequencer*, to impose a total order on the stream of messages broadcast by the group of processes. A total order broadcast of a message starts with the execution of a plain broadcast of the message by the sender process; when a process receives a message from the network, however, it cannot immediately deliver it to the application: in order to guarantee group-wide agreement on the final delivery order, in fact, it has first to wait to receive from the sequencer the corresponding *sequencing* message, and to ensure that all previously ordered messages have been delivered.

STOB algorithms have their key strength point in that they are optimal in terms of the number of communication steps necessary to establish the total order. On the down side, their

main limitation is that their maximum throughput is upper bounded by the capacity of the sequencer to generate sequencing messages. In fact, in Local Area Networks, or in typical data centers, which represent the use case for this thesis, the bottleneck is typically represented by the sequencer's CPU (Santos & Schiper, 2013).

Batching, also known as message packing (T. Friedman & Renesse, 1997), is a well-known optimization technique that aims at coping precisely with this issue: by buffering messages, the sequencer can amortize the sequencing cost and achieve higher throughput; the message delivery latency, however, can be negatively affected at low load, due to the additional time spent by the sequencer waiting (uselessly) for the arrival of additional messages. Therefore, the effectiveness of the batching technique strongly depends on identifying a good batching level b depending on the workload. The optimal batching level b^* is the one that maximizes achievable throughput while minimizing total-ordered messages delivery latency.

The modeling task corresponding to this case study consists in predicting the total-ordered messages delivery latency at the sequencer node depending on the global arrival rate of messages in the system, λ, and a given batching rate b. Such a model can be, then, exploited to derive in closed form the optimal batching value for a given workload (Romano & Leonetti, 2012).

Figure 5.2 depicts the surface of the function corresponding to the total-order messages delivery latency at the sequencer. Clearly, the function exhibits a strong non-linear behavior, and its projections over single dimensions are characterized by very steep curves: this is typical of queuing systems, whose response time quickly grows to infinite when the message arrival rate approaches the maximum service rate sustainable by the sequencer (given the current batching level b) (Kleinrock, 1975). These strong non-linear trends make it challenging to model this performance function by either white box or black box techniques alone.

5.4.1.2 Introduction on the Employed Base Models

TOB. The white box performance model for the TOB case study is an analytical model, originally proposed by Romano and Leonetti (Romano & Leonetti, 2012). The target performance function represents the self-delivery time of a batch of totally-ordered messages and is defined

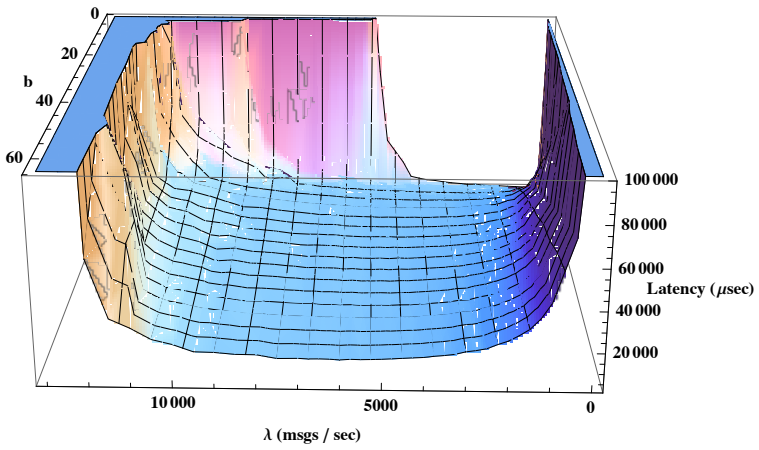


Figure 5.2: Messages delivery time of the STOB service as a function of λ and b

over a two dimensions, corresponding to the rate of messages generation to be totally-order delivered and the batching level at the sequencer side.

In a nutshell, in the considered analytical model the sequencer node is represented as a $M/M/1$ queue, for which each job corresponds to a batch of messages of size b. The message delivery latency is computed as the response time for a queue that is subject to an arrival rate λ corresponding to the frequency of arrival of a batch of messages of size b and whose service time μ accounts both for the CPU time spent for sequencing a message of size b and for the average time waited by a message to see its own batch completed (Romano & Leonetti, 2012).

This analytical model takes as input the CPU costs of processing the first and subsequent messages in a batch. In this study, the estimation of these costs has been performed by finding the values that minimize the analytical model's prediction error over *all* the samples in the available data-set. Therefore, this baseline allows for assessing the effectiveness of the proposed Hybrid Ensemble techniques in improving the accuracy also of *parameter estimation*-based gray box models, introduced in Section 3.1.3.2. Since the fitting has been performed over the whole set of available measurements, the accuracy of this baseline already represents an upper-bound of the one achievable by the *parameter estimation* technique in this use case. The following evaluation, therefore, shows that Bootstrapping can also improve over this other gray box modeling technique.

The dataset corresponding to $\langle input, output \rangle$ tuples, which is used for both training and testing purposes, comprises nearly 500 samples, collected on a cluster of 10 machines equipped

with two Intel Quad-Core XEON at 2.0 GHz, 8 GB of RAM, running Linux 2.6.32-26 server and interconnected via a private Gigabit Ethernet. During experimentation, the batching level has been varied between 1 and 64, and 512-byte messages were injected at arrival rates ranging from 200 msgs/sec to 13K msgs/sec.

DTP. The reference model employed as base predictor for the DTP case study is the *Divide et impera* CTL-2PC model described in the previous chapter, and the target performance function is the closed-system throughput of the system. As already discussed in the previous chapter, this model represents by itself an instance of gray box modeling: in this experimental study, the black box predictor embedded in the gray box model is static, and coincides with the one employed in the evaluation study described in Section 4.2.5. The exploitation of a gray box predictor, and specifically a *Divide et impera*- based one, as building block for proposed hybrid ensemble techniques showcases how different gray box techniques can be further combined together, with the objective of delivering superior accuracy.

The dataset corresponding to $\langle input, output \rangle$ tuples is composed by approximately nine hundred samples, collected by deploying Infinispan on a cloud infrastructure composed by 140 Virtual Machines (VM) equipped with 1 Virtual CPU and 2GBs of RAM; each VM runs a Fedora 17 Linux distribution with kernel 3.3.4- 5.fc17.x86 64. The physical infrastructure hosting the cloud is composed by 18 physical servers equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) processors and 32 GB of RAM and interconnected via a private Gigabit Ethernet; the employed virtualization software is Openstack Folsom.

The considered application is a porting of YCSB (Cooper et al., 2010), the *de facto* standard benchmark for key-value stores, which has been modified in order to support transactions. The generated workloads are based on the A, B and F original YCSB profiles: workload A has a mix of 50/50 reads and writes, and models a session store recording recent actions; workload B is the one of a photo tagging application, which contains a 95/5 reads/update mix; workload F models a user database, in which records are first read and modified within a transaction. In order to generate a wider set of workloads, the number of reads and writes performed by transactions vary between 1 and 5. Finally, two different data access patterns are considered:

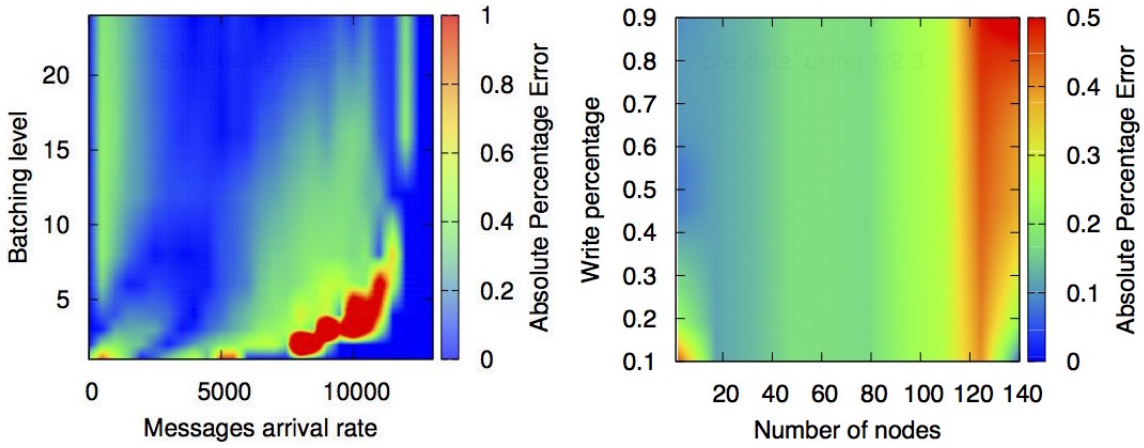
Zipfian, i.e., the popularity of data items follows the zipf distribution (with zipfian constant 0.7), and Hot Spot, according to which the x% of the data accesses are biased towards the y% of the data items (with $x = 99$ and $y = 1$ in our case); the data set is always composed of 500K keys. One YCSB workload generator is collocated with each Infinispan instance, and it consists in a single thread that injects requests in closed loop. The samples relevant to the application's throughput are collected while varying workloads and the data platform configuration, deployed on a number of nodes, noted N, ranging from 2 to 140 and set up with a replication factor in the set $\{1,2,3,\frac{N}{2},N\}$.

Note that the samples collected for this evaluation are a superset of the ones used to validate the *Divide et impera* performance modeling approach in the previous chapter. In particular, this set includes samples corresponding to configurations in which the assumptions and approximations at the basis of the proposed models are challenged, even strongly. For example, the workloads characterized by a zipfian data access pattern and by more than one put operation per transaction do not match the assumption of independent access to data items; moreover, given the high abort rate they induce, such workloads also challenge the employment of the $M/G/1$ queue as abstraction to model a lock. This challenging workloads, for which the devised *Divide et impera* performance models are not able to provide accurate predictions, have been added in order to evaluate the effectiveness of the proposed Hybrid Ensemble techniques in building a more robust performance predictor.

5.4.1.3 Preliminary considerations on the evaluation

This section has the twofold aim of *i*) providing evidence of the relevance of the chosen case study, by highlighting the differences not only between the target performance functions, but also between the white and black box models that serve as baseline to evaluate the gray box ones; and *ii*) discussing important methodological aspects about the evaluation study, in particular regarding the choice of the ML algorithms employed to build the hybrid ensembles and used as baseline competitor.

Relevance and heterogeneity of the case studies. The DTP and TOB case studies are repre-



(a) TOB (b) DTP

Figure 5.3: Error distribution of the base white box models of the two case studies.

sentative of a wide set of different performance functions, modeling approaches and learning problems.

The first characteristic that differentiates one case study from the other is the dimensionality of the target performance function: the messages delivery latency in the TOB case is expressed as a function of only two input parameters (namely, size of the batch and messages arrival rate); conversely, the closed-loop throughput of DTP applications is predicted by the proposed *Divide et impera* models requiring 7 input parameters. This translates into very diverse learning problems to be tackled by a ML algorithm. In particular, as explicitly shown in the next sections, in the TOB case the reference black box model is able to outperform its white box competitor already with 40% of the total available training samples. On the other hand, in the DTP case, the baseline black box predictor's accuracy only approaches the one of the white box counterpart at 90% of the total available training samples.

Moreover, as depicted in Figure 5.3, also the error distribution of the base performance models greatly varies depending on the case study. In particular, Figure 5.3a shows the error distribution of the analytical model of the considered TOB primitive; Figure 5.3b, instead, depicts a projection of the error distribution of the DTP model over two dimensions, namely the number of nodes in the platform and the percentage of write transactions. In both cases, the error is evaluated over all the available training observations). The two error distributions are

sensibly different: in the TOB case, the highest errors are concentrated in a narrow portion of the inputs space, whereas the accuracy of the DTP model sharply decreases as the number of nodes and the transactions' write percentage increase. As detailed in the next sections, this heterogeneity plays a role of paramount importance in determining the effectiveness of the proposed Hybrid Ensemble techniques.

Finally, the base analytical model for the TOB case describes the system performance at a high level of detail by means of a simple $M/M/1$. The DTP model, conversely, goes to great lengths to capture the complex dynamics of a number of internal components of the system (e.g., concurrency control, data locality and replication). Hence, the selected case studies allow for assessing the effectiveness of the proposed Hybrid Ensemble techniques when using base white box models that differ significantly in terms of design complexity and level of detail at which they capture the dynamics of the target system.

Choice and parameterization of the employed ML algorithms. As already mentioned, the proposed Hybrid Ensemble techniques may employ more than one black box learner to compose the ensemble, either based on the same ML algorithm or on different ones. Indeed, the literature on pure black box ensemble learning indicates that the diversity of employed ML algorithms and corresponding parameterizations is an important means to achieve high predictive accuracy; moreover, it shows that the choice of which and how many black box learners to employ and their parameterization can significantly affect the delivered accuracy (Dietterich, 2000; Mendes-Moreira et al., 2012).

The evaluation carried out in the scope of this study, however, is based on a single black box learner, namely Cubist, a DT regressor that approximates non-linear multivariate functions by means of piece-wise linear approximations (a so-called *rule-based model*) (Quinlan, 2012). In particular, a single instance of Cubist is employed as baseline black box predictor and, similarly, only one Cubist model is employed to complement the base white box predictor. Moreover, Cubist encompasses some tuning parameters, namely the possibility to generate mixed rule-based/nearest neighbor models or committee models made up of several rule-based models (Quinlan, 2012). Throughout the evaluation, every Cubist learner is trained with the

default parameters, which correspond to building a single rule-based model. The choice of using Cubist over other available ML algorithms is motivated by the fact that it resulted to deliver the highest accuracy, when compared, during a preliminary experimentation, to alternative ML algorithms —specifically the Weka (Hall et al., 2009) implementation of ANN and SVM.

The rationale behind this choice is to allow the evaluation to focus only on assessing the effectiveness of the Hybrid Ensemble techniques in combining white and black box modeling. Therefore, this choice allows for purely evaluating the impact that the parameters of the proposed hybrid techniques have on accuracy, rather than the impact of the composition and parameterization of the black box part of the ensemble.

Indeed, the problem of finding the ML algorithms composition and parameterization that maximizes accuracy given a training set is a general problem, which falls beyond the sole boundaries of the Hybrid Ensemble techniques, and that can be addressed with standard techniques based on cross-validation, e.g., random search or Bayesian Optimization (Bergstra et al., 2011; Thornton et al., 2013).

Finally, it is important to stress that the evaluation conducted with Cubist as unique black box learner is still fair, in the sense that, when comparing an Hybrid Ensemble-based learner with a pure black box one, the same data-set and the default parameterization is used to train the Cubist model.

5.4.2 Bootstrapping

The assessment of the Bootstrapping technique consists of three part: Section 5.4.2.1 describes and evaluates the proposed *initKB* implementation to build the synthetic training set; Section 5.4.2.2 is devoted at assessing the accuracy achievable by using the different update algorithms; finally, Section 5.4.2.3 discusses and evaluates the predictive capabilities of a bootstrapped learner in extrapolation.

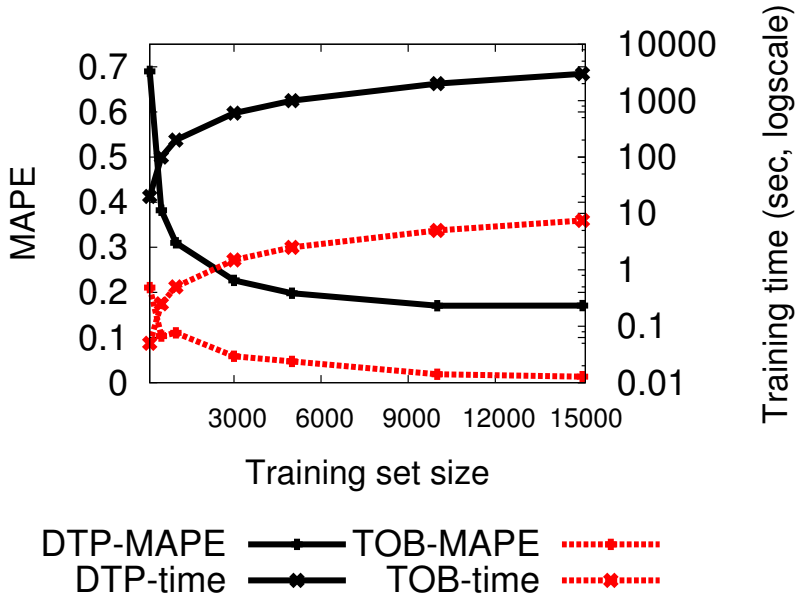


Figure 5.4: Fitting the white box model via ML: training time vs MAPE.

5.4.2.1 Initialization

The evaluation study begins by evaluating the gray model's capabilities of approximating a base white box learner and construction time depending on the number of samples of the feature space used to populate the initial synthetic training set.

For this study, the implementations of the primitives invoked in the initKB function, used to create the initial synthetic training set ST are the following:

• SAMPLECONFIGSPACE is based on a uniform random sampling of the parameter space: the rationale behind this choice is that a random policy is the most simple sampling strategy to implement, yet it has been shown to be very effective (Bergstra et al., 2011).

• ESTIMATEFITTINGERROR implements 10-fold cross-validation, i.e, it i) partitions ST into 10 bins $ST_1 \dots ST_{10}$; ii) iteratively $\forall i = 1 \dots 10$, trains the black box learner over $ST \setminus S_i$ and evaluates its accuracy against S_i and iii) returns the average accuracy.

• ISACCURATE is based on a simple predicate that returns *true* if the accuracy of the black box model has not increased enough (i.e., more than ε in relative value) over the last n iterations (with $\varepsilon = 0.01$ and $n = 3$ in this evaluation).

• NEXTSIZE returns the size of ST at the current iteration plus a fixed value (set to 500 in this

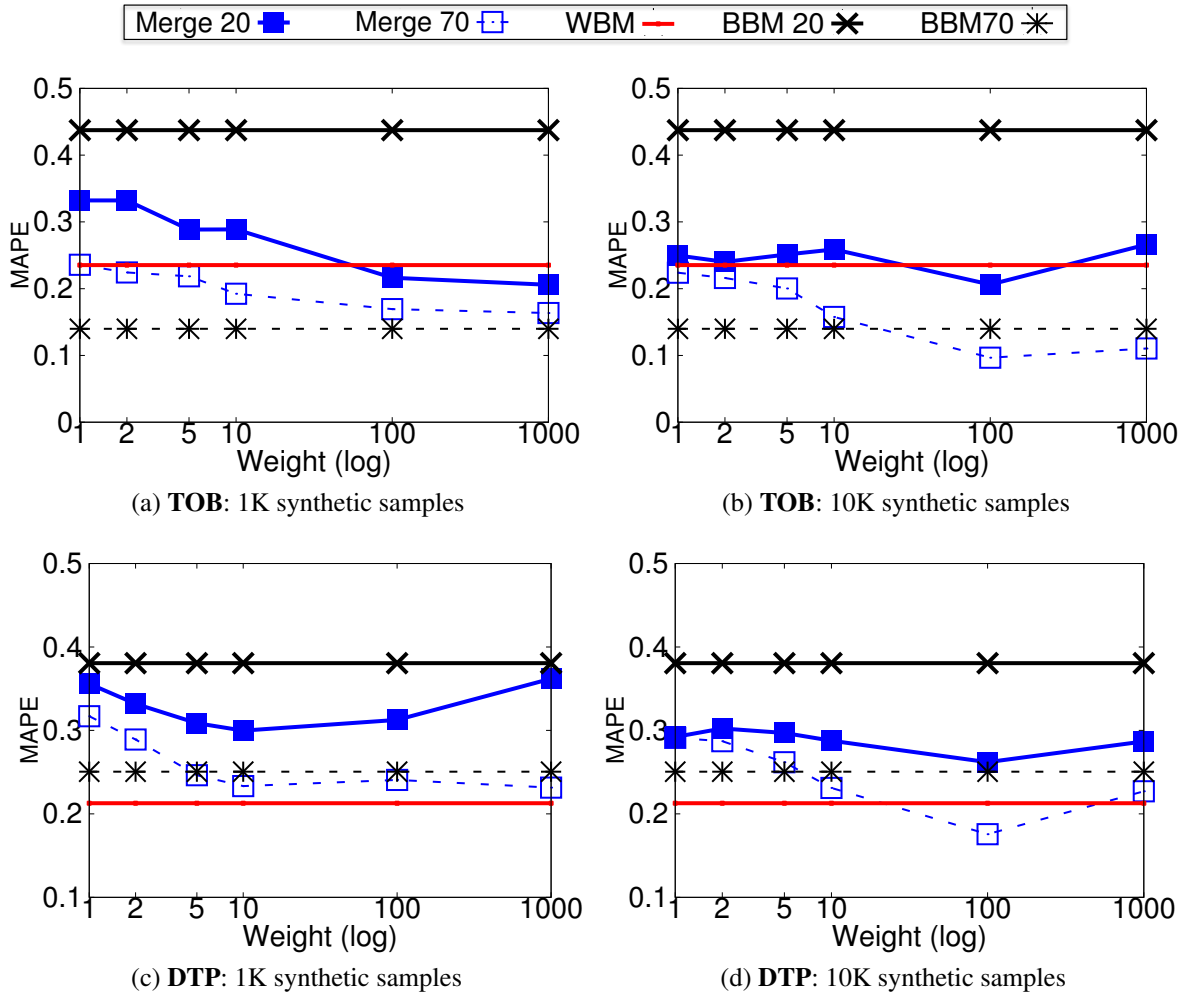
evaluation).

Therefore, the implemented initKB evolves by increasing the size of ST by a fixed step and randomly sampling the parameter space to build the new ST; it terminates when the accuracy on the black box learner trained over ST plateaus.

Figure 5.4 reports, for both case studies, the gray box model building time and the Mean Average Percentage Error (MAPE), computed as $Avg.(\frac{|real-pred|}{pred})$, of the gray box model with respect to the predictions produced by the base predictor evaluated by means of ten-fold cross validation. On the x-axis there is the number of initial synthetic samples included in the training set of the gray box model. For both case studies, the *initKB* function returns a synthetic training set of 10K samples, as it detects no noticeable improvements in the black box model's fitting accuracy. Indeed, Figure 5.4 shows that even proceeding up to 15K samples the accuracy gain is negligible.

The model building time portrayed in the plots corresponds to the sum of the time needed to query the base predictor in order to generate the synthetic data set of a given cardinality plus the time needed to train the black box model over such set. In this study, in which Cubist is used as black box learning algorithm, the training time for both the target application has been less than half a second; the gray box model building time in the plots is, thus, largely dominated by the cost needed to query the base predictor. As shown by Figure 5.4, in the DTP case this cost is much higher than in the TOB one, as the corresponding base predictor is solved through multiple iterations (Didona & Romano, 2014b) (as discussed in Section 4.2.4. However, it should be noted that the cost to query the base model has to be paid only once, upon initializing the bootstrapped learner, as the update phase only requires to re-train the black box learner.

Figure 5.4 shows that, by fitting the base predictor using black box techniques, a loss of accuracy is unavoidable. The actual extent of this accuracy degradation depends on factors such as the number of samples used to construct the initial synthetic training set and the intrinsic capability of the learner to approximate the target function. The plot shows that, as expectable, larger training sets yield a lower approximation error, at the cost of a longer training time; it also shows that Cubist is able to fit the TOB response time function encoded in the base predictor very well (3% of MAPE with a 10K samples training set) but it is unable to achieve similar



(a) **TOB**: 1K synthetic samples

(b) **TOB**: 10K synthetic samples

(c) **DTP**: 1K synthetic samples

(d) **DTP**: 10K synthetic samples

Figure 5.5: Impact of the weight parameter for the Merge updating policy, using 1K and 10K synthetic samples.

accuracy for the DTP case. This arguably depends on the fact that Cubist approximates non-linear functions by means of piece-wise linear approximation in the leaves of the decision tree that it builds. Such model may be unable to properly approximate the performance function of the base DTP performance model, which is defined over a multi-dimensional space and exhibits strongly non-linear behaviors.

5.4.2.2 Updating

This section evaluates the alternative algorithms for the updating of the knowledge base, that have been presented in Section 5.2.1.2: it first assesses the sensitivity of each algorithm to its key parameters and finally compares their accuracy assuming an optimal tuning of such parameters.

Figure 5.5 reports the results of a study aimed at assessing the impact of the weight parameter on the resulting accuracy of the bootstrapped model, while considering synthetic training sets of different initial sizes, namely 1K (Figure 5.5c and 5.5a) and 10K samples (Figure 5.5d and 5.5b).

Two scenarios are considered, which correspond to relying on the availability of 20% and 70% of the entire data set composed of collected, real samples; these are fed as input to both the Merge algorithm and to the plain black box model (noted BBM in the plots), which serves as first baseline. As a second reference, the plots also show the accuracy achieved by using the based predictor (noted WBM in the plots), which incurs a MAPE that is independent of the initial size of the synthetic training set. On the x-axis there is the weight parameter of the Merge algorithm, and y-axis reports the MAPE computed with respect the whole set of actual samples (i.e., unlike in the previous section, here the MAPE is not computed with respect to the output of the analytical models).

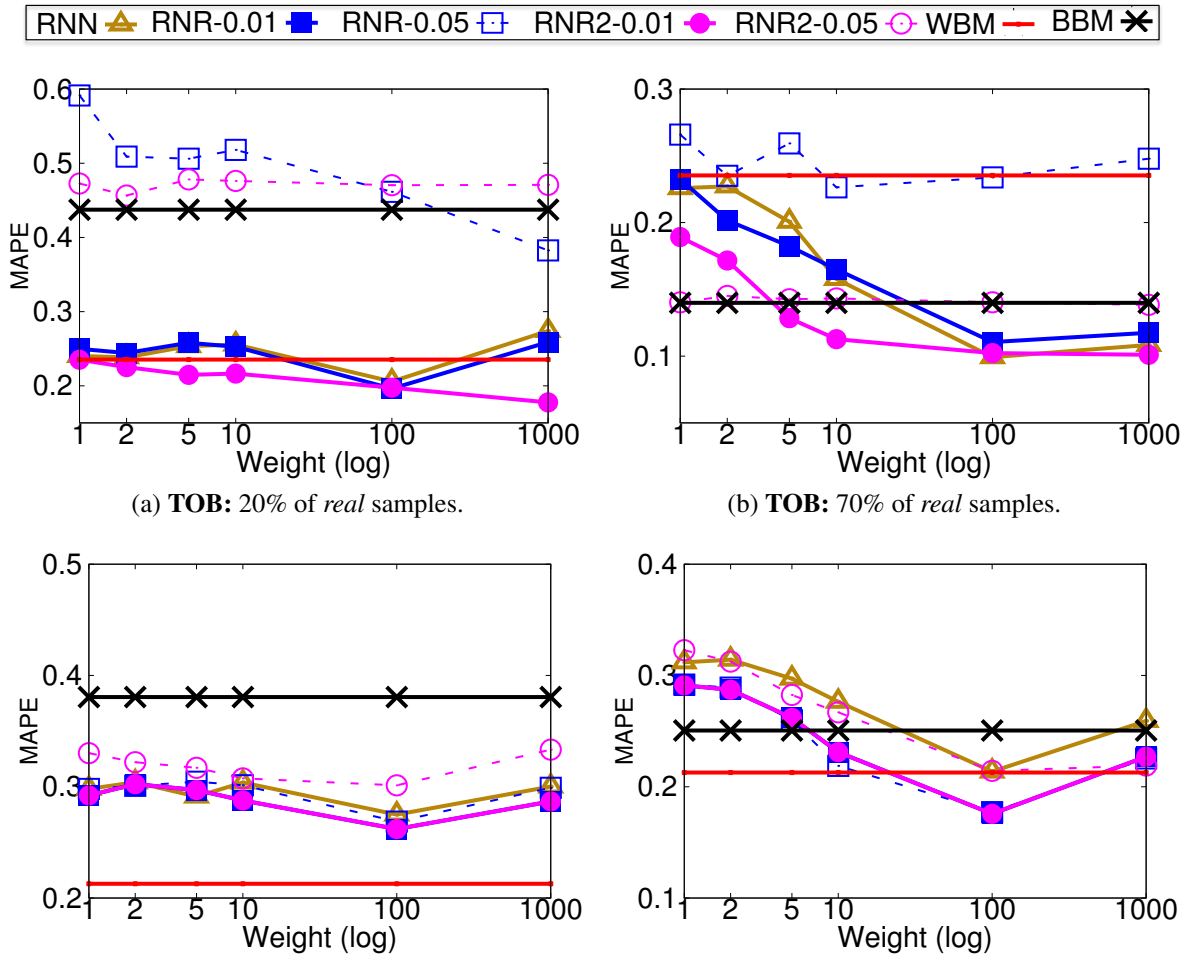
The first finding revealed by the plots is that employing 10K synthetic samples is beneficial for the accuracy achieved by the bootstrapped learner. This happens because, as already discussed, larger synthetic training sets allow the black box learning algorithm to encode better the base predictor's function; this, in turn, yields to inherits a predictive accuracy that is closer to the one exhibited by the base model itself.

The other very evident finding highlighted by the plots is the relevance of correctly tuning the weighting parameter, regardless of the size of the initial synthetic training set. However, it is possible to observe, by comparing Figures 5.5c and 5.5d, that the best setting of this parameter may be relatively larger in the case of larger synthetic training set than for the case of smaller one. This can be explained by considering that, by increasing the size of the initial training set, the ratio of real vs synthetic samples correspondingly decreases. From the ML perspective this translate into decreasing the relevance of the real samples with respect to that of the "surrounding" synthetic samples. In fact, the Merge update method never evicts synthetic samples from the knowledge base: if the initial synthetic training set is significantly larger than the number of available real samples, these are always surrounded by a large number of synthetic ones, which end up obfuscating the information conveyed by the real ones. By increasing the weight of

the samples gathered from the running system, the statistical learner is guided to minimize the fitting error on these points. On the other hand, using excessively large weight values can be detrimental, as it makes the learner more prone to over-fitting.

Overall, the experimental data show that both with large and small initial synthetic training set, Merge achieves significantly higher accuracy than both Cubist and the base model, when provided with 70% of the data in their training set. When the training set percentage is equal to 20%, the scenario is rather different. In both scenarios, the gray box model still achieves a higher accuracy than a pure ML-based technique. However, the gray box is only marginally better than the base predictor with the large initial synthetic training set, and slightly worse than then the base model with small initial synthetic training set. This can be explained by considering that the gain achievable using the 20% training set is relatively small, and can be even outweighed by the loss of accuracy introduced by the learning of the initial base model (see Section 5.4.2.1). This is also confirmed by the fact that the MAPE with respect to the base model of the bootstrapped one using a synthetic training set of 10K samples is significantly lower than with 1K samples, as shown in Figure 5.4.

The plots in Figure 5.6 focus the comparison on the updating policies RNN, RNR, and RNR2. Unlike Merge, these techniques aim at avoiding the coexistence in the training set of "neighboring" synthetic and real samples, by removing or replacing synthetic samples close enough to the real ones. The intuition underlying these approaches is that the information conveyed by the base predictor may be erroneous, and can hence contradict the real samples, therefore introducing noise in the black box learning process. With the exception of the RNN method, which uses exclusively the weight parameter, RNR and RNR2 also use a cut-off parameter, which defines the relative amplitude (normalized over a maximum distance) of the radius that is used to determine which synthetic points are to be removed (RNR) or updated (RNR2), whenever a new real sample is incorporated in the training set. For the sake of presentation, only two cut-off values are considered, namely 1% and 5%, and the weight parameter is treated as the independent variable. The choice of reporting results with these cut-off values is motivated by the fact that they suffice into illustrating the main dynamic related to the setting of this parameter: the higher is the cut-off, the more the training set is composed by only real samples,



(a) **TOB:** 20% of *real* samples.

(b) **TOB:** 70% of *real* samples.

(c) **DTP:** 20% of *real* samples.

(d) **DTP:** 70% of *real* samples.

Figure 5.6: Impact of the weight and cut-off parameters for RNN, RNR, and RNR2, using 10K synthetic samples.

thus making the bootstrapped model eventually collapse to the pure black box one.

In all the performed experiments, the employed distance function is the Euclidean one, but any could be used. Before computing the distance between two samples, a feature normalization process is performed, i.e., the value of every feature is normalized so as to lie in the range [0,1]. When using scale-sensitive distance functions like the Euclidean one, this avoids features that naturally assume higher values (e.g., the messages arrival rate in the case of TOB) to have more weight in determining the distance than other ones (e.g., the batching value in the TOB case study).

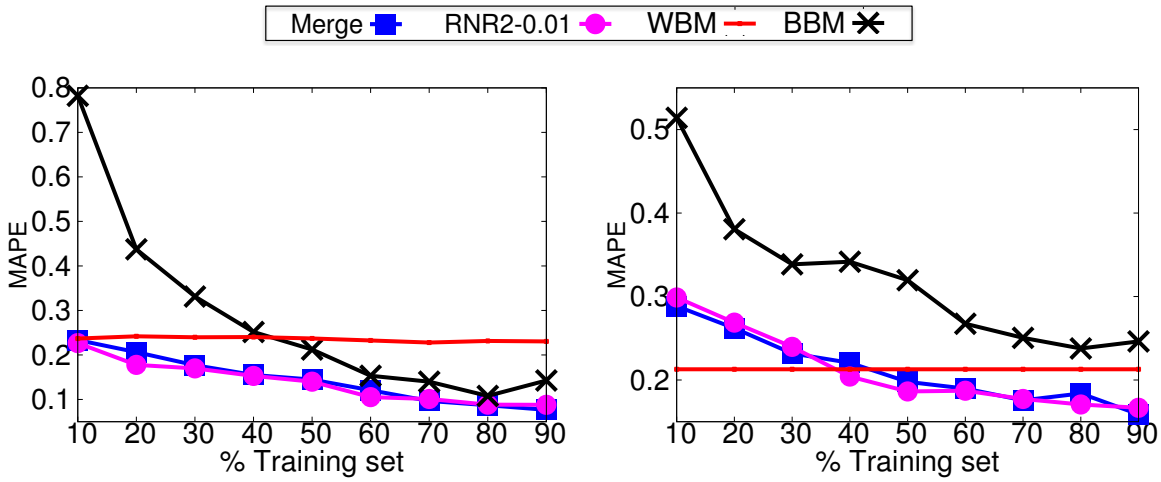
Figure 5.6c and 5.6a, resp. Figure 5.6d and 5.6b, report the MAPE achieved when using 20%, resp. 70%, of the real data set as training set, reporting, as before, the reference values

achieved by the base model and by Cubist (non-bootstrapped). The first result highlighted by these plots is that, also in the replace-based update variants, the weight parameter plays a role of paramount importance. Also the cut-off parameter has a huge impact on the final accuracy of the hybrid model, when implementing RNR and RNR2.

Overall, the cut-off based update policies are more effective into increasing predictive accuracy in the two considered case studies. The improvement with respect to RNN is more evident in the DTP case: this is because RNN entails the possibility of evicting real samples from the training set, whereas RNR and RNR2 do not. As a result, RNN discards some of the information conveyed by real samples, thus losing some of its corrective power.

Note that this effect is tightly related to the characteristics of the target performance function and of the distribution of the corrective samples. For the TOB case, in fact, both real and synthetic samples are drawn uniformly at random from the whole space of possible arrival rate and batching level configurations. Moreover, the 10K synthetic samples are very cluttered in the two-dimensional space in which they lie, thus reducing the probability that RNN evicts a real sample instead of a synthetic one. Conversely, for the DTP case, the samples in the synthetic training set are drawn uniformly at random but the real ones are not as they are, instead, representative of typical configurations and workloads for that kind of platforms. For example, the density of the points characterized by a number of nodes smaller than 25 is higher than the one relevant to points corresponding to more than 100 nodes in the platform; in the same guise, as already said, the replication degree for data items is defined over the set $\{1,2,3,\frac{N}{2},N\}$, being N the number of nodes. This, together with the relative sparseness of the 10K synthetic samples in the seven-dimensional space in which they lie, induce RNN to evict, in some cases, real samples.

Finally, Figure 5.7 compares the accuracy achieved by the two best performing updating heuristics, Merge and RNR2, with that achieved by a pure white and black box approach. Also in this case the size of the initial synthetic training is 10K, and the parameters used by Merge are the ones that resulted in the best performance in the evaluation cases considered so far. The accuracy of the various predictors is evaluated while varying the size of the available training set from 10% to 90%.



(a) TOB (b) DTP

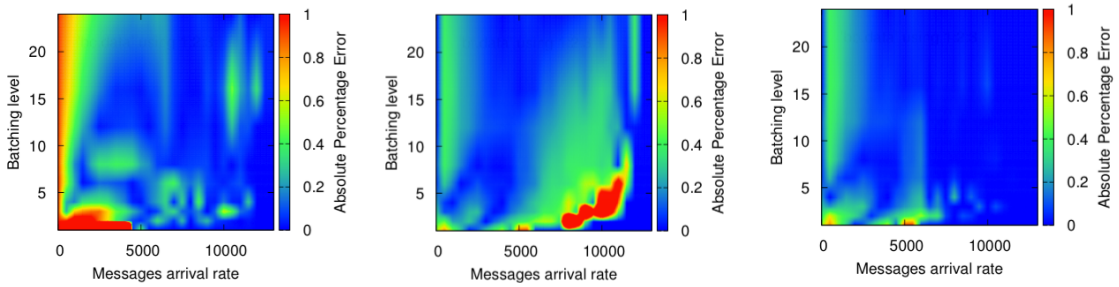
Figure 5.7: Comparison between Merge and Replace-based Bootstrapping

The plot in Figure 5.7 clearly highlights the advantages that the Bootstrapping technique can provide, eventually outperforming both the base model and the reference ML-based predictor. It also shows that, in the considered case studies, and for the considered parameters' values, there is no clear winner between the two updating variants. In fact, the conducted evaluation suggests —maybe surprisingly— that the weighting parameter results to be the one that affects accuracy the most, up to the point that its careful tuning allows the Merge updating policy to perform similarly to the —relatively more complex— RNR2.

5.4.2.3 Bootstrapping in extrapolation

So far, the Bootstrapping technique has been evaluated by drawing the additional training set D_t for the black box learner uniformly at random from a real data set D, and assessing its accuracy over $D \setminus D_t$. This means that the learned performance function has been corrected by benefiting from an unbiased sampling of the whole space over which its accuracy is then assessed. This section serves the purpose of assessing the Bootstrapping technique's robustness against biased sampling strategies: even if provided only with a set R of real samples corresponding to narrow regions of the parameters' space, the bootstrapped learner still inherits the predictive power of the base base predictor when working in extrapolation with respect to R.

A realistic use case for such a scenario would be if the real samples were not to be collected



(a) **TOB:** Cubist with real samples (b) **TOB:** Cubist with synthetic samples (c) **TOB:** Bootstrapping (Merge)

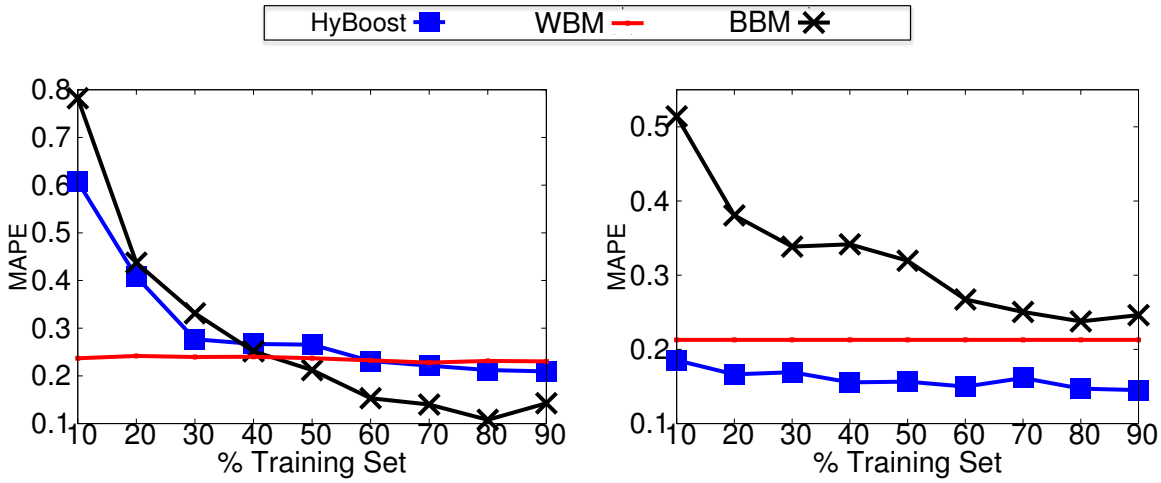
Figure 5.8: Assessing Bootstrapping's effectiveness when working in extrapolation.

during a dedicated and controlled training phase, but, rather, directly from the in-production system. In this case, in fact, there would not be the possibility to steer a training phase that guarantees a reasonable coverage of the whole configurations' space of the target performance function. As a result, the bootstrapped learner might not be corrected at all in some portions of the configurations space, thus completely relying on the underlying base model's predictive capabilities to serve performance predictions queries regarding such regions.

To simulate such a biased sampling scenario, the following experiment has been performed. A dataset D_T has been drawn from a *specific* portion of the whole set of available real samples, instead that uniformly at random from it. This dataset is used as training set for a pure black box learner and as additional training set for a bootstrapped leaner. Then, the predictive capabilities of the two learners have been compared, by evaluating their accuracy over the whole set of available real samples.

The results hereby presented refer to TOB case only, as its two-dimensional nature allows for easily visualizing the outcome of the aforementioned comparison (Figure 5.8). Similar results have, however, been obtained for the DTP case study.

The set of real samples is drawn from the region of the parameters' space corresponding to the right portion of the heat maps in Figure 5.8 (that is $[6500, 13000] \times [1, 24]$). Figure 5.8a shows the accuracy achieved by Cubist when trained only over samples from this region and queried over the whole test set. It is easy to see that it attains a very high accuracy in the parameters' space that has been included in its training set; conversely, it blunders when asked to provide a performance forecast for samples belonging to a previously unseen parameters'



(a) TOB (b) DTP

Figure 5.9: Evaluating the accuracy of HyBoost.

region.

Figure 5.8c, instead, shows the accuracy achieved by a bootstrapped learner trained over a combination of synthetic samples and the same set of real samples used in the previous case. Clearly, the accuracy in the right part of the plot is similar to the one in Figure 5.8b; the left part, instead, which corresponds to the performance queries in extrapolation, portrays a significant enhancement in accuracy. These improved predictive capabilities in extrapolation stem from the availability of a synthetic training set provided by the embedded base model. This claim can be verified by analyzing Figure 5.8a, which reports the accuracy of a Cubist learner trained *only* over synthetic data samples: it is easy to see that the left side of the plot is very similar to the left side of the plot in Figure 5.8b, demonstrating how a bootstrapped learner is able to leverage the knowledge provided by the base model about the performance of the target application in unexplored regions of the parameters' space.

5.4.3 Hybrid Boosting

The conducted evaluation study on the HyBoost technique, described in this section, only focuses on analyzing the effectiveness of this technique depending on the characteristics of the white and black box models for two considered case studies. The only tuning parameter of this technique, in fact, would be the size and the composition of the chain, i.e., the number of

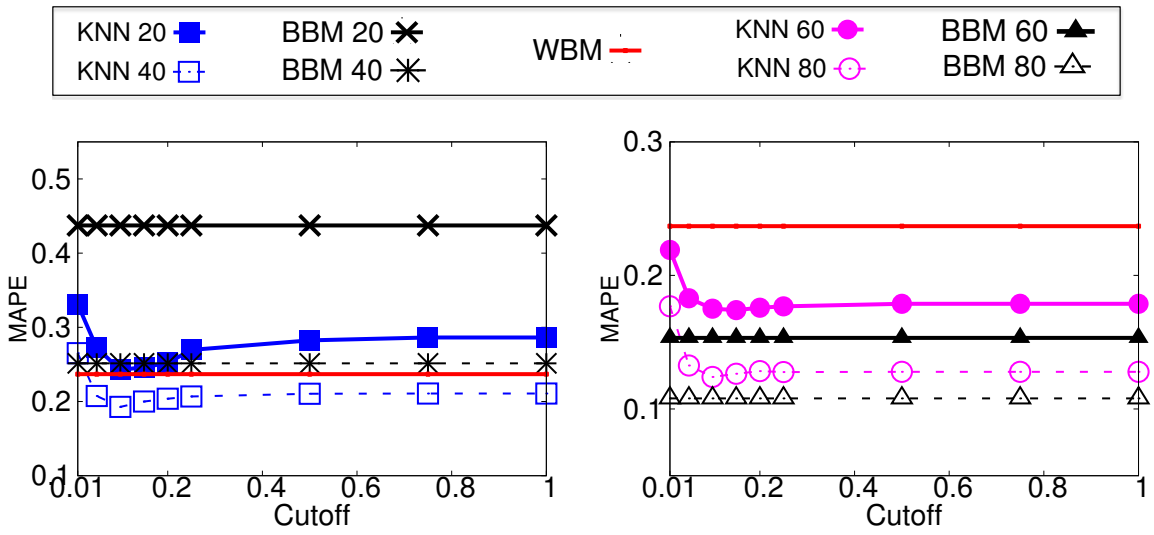
employed corrective black box models, their learning algorithms and corresponding parameterization. As already specified in Section 5.4.1.3, however, the evaluation study presented in this work only focuses on hybrid models using Cubist as only black box learner; therefore, the plots in Figure 5.9 only report the accuracy of the HyBoost-based models while varying the available training set.

As depicted in Figure 5.9a, the HyBoost technique is not effective in the TOB case study. The hybrid predictor is, in fact, able to leverage the availability of a base model to reduce error with respect to the black box learner alone when low training data is available, but it is never able to deliver higher accuracy than the best of the two baseline predictors. Arguably, this depends on the fact that the error distribution of the base model is not easy to be learnt by Cubist, which, when provided with enough training data, achieves higher accuracy by directly learning the target performance function rather than the analytical model's residuals.

The landscape changes dramatically in the DTP case study. As shown in Figure 5.9b, the HyBoost-based learner outperforms in terms of accuracy both the underlying white and the black box models, at any of the considered percentages of available training data. On the other hand, the accuracy only marginally increases as more training data becomes available: this suggests that Cubist is effective into identifying a proper corrective function for the base model with a small amount of training data, but it is unable to sensibly improve over it as more information become available.

5.4.4 Hybrid KNN

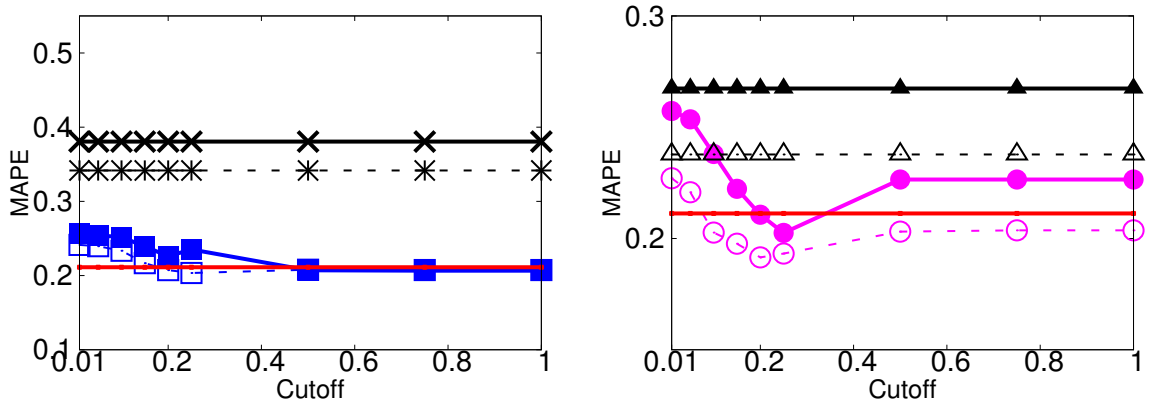
The evaluation of the KNN Hybrid Ensemble technique is focused on analyzing how its accuracy is affected by the tuning of the cutoff value, which determines the maximum distance between two samples to be considered "neighbors". Figure 5.10 and Figure 5.11 show the accuracy achieved by KNN while varying the cut-off parameter c for the case, respectively, of TOB and the target DTP, namely Infinispan; for the sake of the results' presentation and usability, KNN has been configured to use at most $k = 10$ neighbors. Each plot reports results obtained by letting KNN and Cubist observe two percentages of the training set, namely: 20% and 40% in Figures 5.10a and 5.11a; 60% and 80% in Figures 5.10b and 5.11b.



(a) Training set 20% and 40%

(b) Training set 60% and 80%

Figure 5.10: Sensitivity analysis of KNN w.r.t. the *c* parameter (TOB)



(a) Training set 20% and 40%

(b) Training set 60% and 80%

Figure 5.11: Sensitivity analysis of KNN w.r.t. the *c* parameter (DTP)

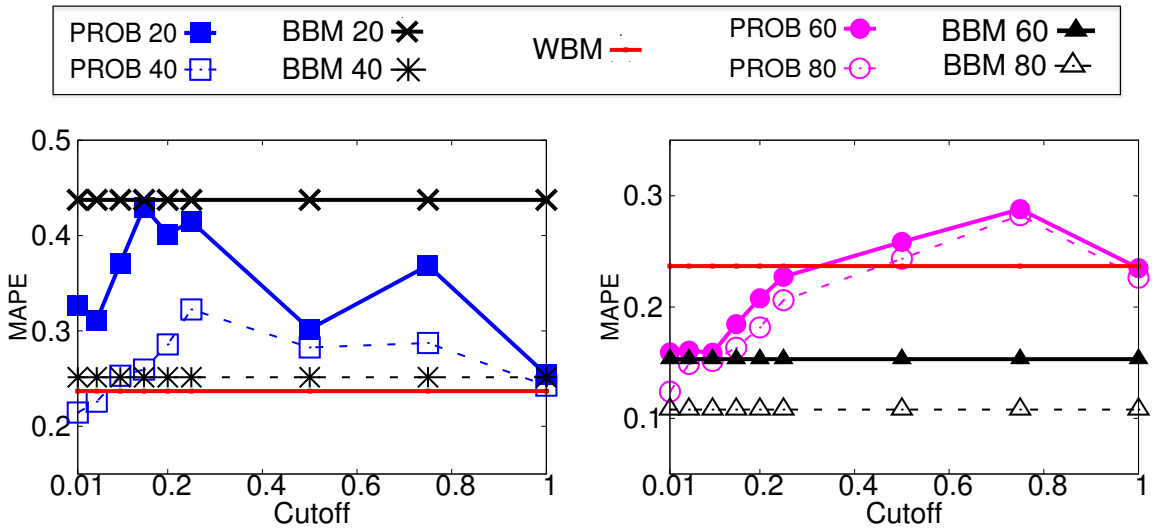
The plots highlight that the optimal settings of the *c* parameter is quite different not only depending on the case study, but also on the available amount of training data. It is, in fact, possibly to identify two different dynamics, depending on the target case study. In the TOB case, the KNN-based predictor is able to outperform both the base model and the black box one with a low amount of additional training data and using a small cut-off value (Figure 5.10a). On the other hand, at higher percentages of training set, it is unable to outperform the black box learner for any cut-off value (Figure 5.10b). The first phenomenon can be explained by noting that, for

this case study, it is relatively easy to determine which is the best model to use depending on the incoming query: as confirmed by the heat-map presented in Section 5.4.1.3 (Figure 5.3a), in fact, the region in which the base white box learner is more inaccurate is well defined and, thus, identifiable by means of a neighbors-based approach. The second phenomenon can be explained by noting that, at high training set percentages, the black box learner is sensibly more accurate than the white box one, and, thus, it is more likely to be ultimately chosen as predictor. In the KNN case, however, only a subset of the available training data is used as training set for the black box learner (80% in this experiment), as part of it constitutes the validation set to estimate the accuracy of the models. Therefore, even if the KNN predictor mostly relies on the black box model to serve queries, the regressor it embeds is trained with less data than its "pure" counterpart, thus delivering a lower accuracy.

In the DTP case study, conversely, it is harder for the KNN learner to determine when it is better to rely on the base model or the black box one. This can be explained both by the high dimensionality of the inputs parameters and by the fact that the base predictor's error heat-map does not present spikes, unlike its TOB counterpart (see Section 5.4.1.3, Figure 5.3b). As a result, when the available training set is small, and by using high cut-off values, the KNN predictor manages to achieve an accuracy that is only to be as good as the best (on average) of the two models, namely the white box one. When the training set increases, the predictor can rely both on more samples to use as neighbors and on a more reliable black box predictor: given that there is no clear winner between the black box model and the base predictor in this case, the KNN predictor manages to achieve an accuracy that is better than theirs. The cut-off value at which the gain is more sensible is higher than in the TOB case: arguably, this depends on the higher dimensionality of the inputs space, which demands higher threshold values to gather a representative neighborhood.

5.4.5 Probing

This section describes the results of the experimental evaluation aimed at assessing how the accuracy of the Probing technique is affected by training set's size and setting of the cut-off parameter c, which determines the maximum tolerable error to regard a white box model's



(a) 20% and 40% of training set

(b) 60% and 80% of training set

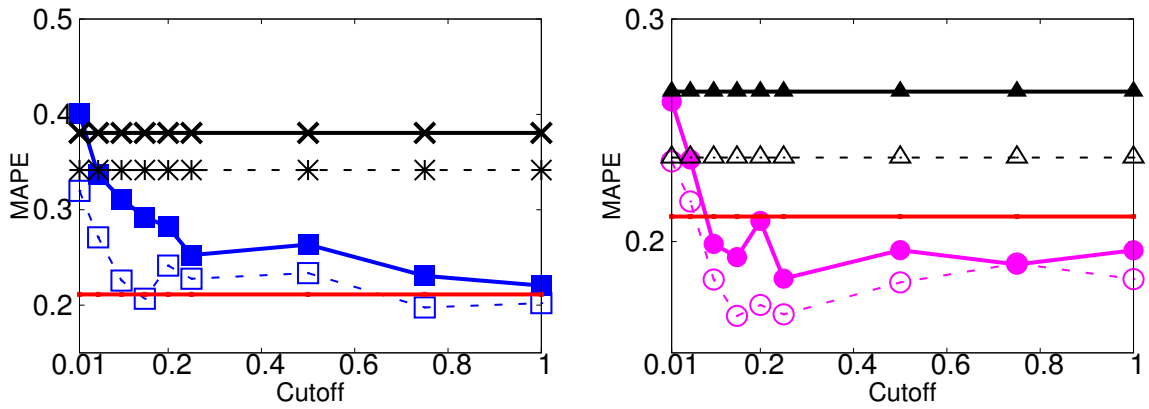
Figure 5.12: Sensitivity analysis of Probing w.r.t. the c parameter (TOB)

prediction as accurate. The classification algorithm employed to train the classifier responsible for estimating the best model for a given query is the Weka implementation of the C.45 Decision Tree (Quinlan, 1993b).

Figure 5.12 and Figure 5.13 report the results of such sensitivity analysis. The first phenomenon that comes evident for both the case studies is that the accuracy does not vary, as a function of c, as smoothly as in the KNN case, which is the other considered *Selection*-based Hybrid Ensemble technique that relies on a cutoff parameter. This is because c directly affects both the training set of the classifier used and of the black box performance predictor. The resulting behavior of these two components affects in an intertwined and complex fashion that ultimately results in the portrayed accuracy trends.

Also, as expectable, for both case studies, the lower the employed cut-off value the more the accuracy is similar to the one attained by the black box model. This happens because the underlying white box model is considered to be accurate and, thus, to be reliable, only if it attains a correspondingly low error. In a dual fashion, as c moves towards higher values, the accuracy delivered by the Probing-based predictor resembles the underlying base model's one.

Regarding the specific case studies, the characteristics of the corresponding base predictors and black box models play again a fundamental role to determine the effectiveness of the



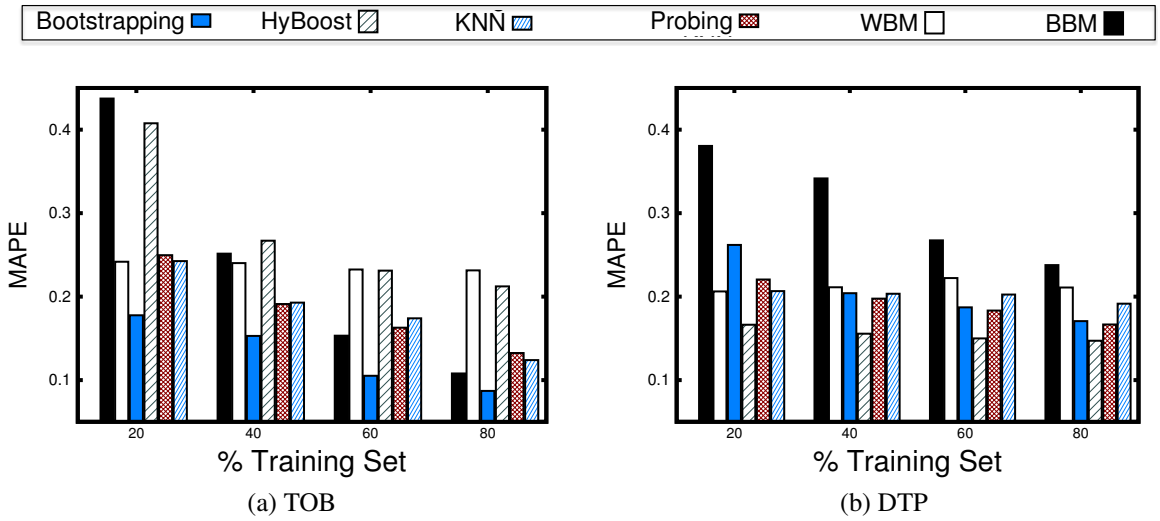
(a) 20% and 40% of training set

(b) 60% and 80% of training set

Figure 5.13: Sensitivity analysis of Probing w.r.t. the c parameter (DTP)

Probing technique. In the TOB case, in fact, Probing only slightly enhances the accuracy with respect to the best between the pure white and black approaches at a medium training set (Figure 5.12a). With lower training set, the classifier cannot distinguish when it is better to rely on the white or the black box. At higher training sets, the accuracy of the black box model alone is generally better than its white box counterpart's (Figure 5.12b). As a result, even if the accuracy of the white box model is below the desired threshold, it is likely that the black box learner alone could still reach a higher accuracy, if trained with the proper data-set. Ultimately, this leads the Probing-based predictor to rely on the white box model even when it should not, and in removing samples from the black box learner's training set, thus reducing its accuracy.

The plots in Figure 5.13, instead, reveal slightly different dynamics for what concerns the DTP case study. On one side, in fact, just like the TOB case, the classifier does not help, or only marginally helps, in increasing accuracy when there is low amount of training data available (Figure 5.13a. Conversely, as training data become more abundant, Probing is able to deliver higher accuracy than the two underlying models alone (Figure 5.13b) this is because, as already highlighted during the KNN discussion, there is no clear winner between the white and the black box model. Therefore, provided that the classifier is able to distinguish when to prefer one over the other, it is possible to take selectively advantage of both with beneficial effects on accuracy.



(a) TOB (b) DTP

Figure 5.14: Comparing the performance of the 4 proposed gray box techniques.

5.4.6 Comparison among the approaches

This section concludes the experimental evaluation and is dedicated to comparing the accuracy achieved by the four proposed hybrid ensemble techniques in the two considered case studies. In particular, the comparison is performed assuming a proper tuning of the internal parameters of the compared ensemble algorithms. Specifically, the reported data are obtained using 10-fold cross validation to determine appropriate values for the internal parameters of the compared ensemble algorithms.

As already hinted in Section 5.4.1.3, identifying the best gray box model and corresponding parameterization given some training data is a problem that falls beyond the scope of the proposed Hybrid Ensemble techniques: it is, indeed, a common trait shared with pure black box modeling techniques. Therefore, it can be tackled by means of standard techniques developed for the selection and tuning of Machine Learning algorithms, such as Bayesian Optimization or grid/random search (Bergstra et al., 2011).

The following evaluation aims at showing how the characteristics of the target performance function and of a hybrid predictor affect accuracy in the most favorable case, i.e., excluding the cases in which a given predictor performs poorly only because of a correspondingly poor setting of its internal parameters.

Figure 5.14 reports the accuracy attained by the four proposed gray box models, as well as

by their underlying pure white and black box ones (noted, respectively, WBM and BBM), while varying the amount of available training data. The first result highlighted by analyzing the plots is that there is always one instance of a gray box predictor that performs better —accuracy-wise— than the pure white and black box models that it combines. This result is particularly important, as it showcases the effectiveness of the proposed approaches in reconciling the white and black box modeling paradigm with the aim of increasing accuracy.

Moreover, Figure 5.14 also shows that each of the proposed hybrid techniques succeeds in outperforming the baselines in at least one case study and for a given amount of training data. Given that, for the DTP case, the baseline model is actually a *Divide et impera*-based model, this plot showcases that the gray box modeling techniques presented in this work can be combined to further increase accuracy. In a dual fashion, Figure 5.14 also shows that none of the considered Hybrid Ensemble models is able to consistently outperform the white and black box baselines.

On one side, these results demonstrate the potentialities of the described modeling techniques; on the other, they highlight the key role played by the technique employed to couple a white and black box model to determining the final achieved accuracy.

In particular, Figure 5.14a shows that Bootstrapping is the best hybrid modeling technique in the TOB case study. This is because the employed ML algorithm is able to encode the original performance function of the white model without loss of accuracy, thus being able to fully taking advantage of the available performance measurements data for "corrective" purposes. On the other hand, Probing, KNN and HyBoost prove to be less effective, as their internal dynamics do not couple well with the characteristics of the distribution error of the base model.

In a dual fashion, Figure 5.14b shows that Bootstrapping is never the best performing gray box modeling technique, because of the loss of accuracy incurred by the ML algorithm when learning the performance function encoded by the base model. Probing, KNN and Hyboost, conversely, perform better, with the latter being the solution that better spouse the characteristics of the underlying base model.

Overall, the final take-away message conveyed by Figure 5.14, is that no one-size-fits-all Hybrid Ensemble model exists, among the proposed ones, that outperforms all the others in

every scenario. Given that all the proposed hybrid techniques rely on a black box component, this conclusion is closely related a very well known result in the ML field, namely the *no free lunch theorem* for Supervised Learning, formulated by Wolpert (Wolpert, 1996). This theorem, roughly speaking, states that no learning algorithm is universally superior to every other; similarly, the performed evaluation shows, experimentally, that also the effectiveness of a gray box modeling technique ultimately depends on the characteristics of the use case, of the target performance function, and of the available data.

5.5 Conclusions

This chapter has described four Hybrid Ensemble modeling techniques that rely on exploiting in synergy white and black box modeling with the purpose of achieving a higher accuracy than a predictor based on a pure white or black box method alone.

In particular, two of the proposed techniques implement the *Patching* principle, according to which black box modeling is employed to correct the predictions produced by a base white box model. Specifically, the first of such techniques, named *Bootstrapping*, consists in generating the training set for a black box algorithm by relying on the predictions of a complementary white box model; information conveyed by samples corresponding to real performance measurements are incrementally integrated into the prediction model. The second *Patching*-base techniques, namely Hybrid Boosting, instead relies on the idea that it may be easier to learn how to correct the prediction of a baseline model, rather than learning directly the target performance function. Therefore, it organizes a white box predictor and a set of black box predictors as a chain, in which the goal of each model is to iteratively correct the prediction error of the previous ones.

The other two proposed technique implement the *Selection* principle, which consists in building several white and black box performance models in parallel and, at query time, in employing only the one that is expected to maximize accuracy depending on the configuration/-workload whose performance is being predicted. The first such technique, named Hybrid KNN, evaluates the accuracy of the available models on a set of samples D_k that are as similar as possible to a new sample x, in order to estimate the most suitable model to use for predicting the performance for x. The second *Selection*-based technique, named Probing, aims at specializing

a single black box model in predicting the target performance function in the specific regions of the inputs space where none of the available white box models proves to be sufficiently accurate. This technique relies on a generic classifier to determine which among the available models is going to be used depending on the query, instead of estimating models' accuracy over a set of neighbors.

The effectiveness of the four proposed techniques have been investigated using two case studies, namely a Total Order Broadcast primitive and the Infinispan DTP. An analytical model is used as base white box predictor for the first case study, and some of its internal parameters are obtained by means of a black box fitting algorithm; on the other hand, the base predictor for the DTP case is one of the *Divide et impera* models proposed in Chapter 4. This showcases how the proposed Hybrid Ensemble techniques can be implemented in conjunction with other gray box modeling techniques.

An extensive experimental evaluation has shown that these techniques can achieve higher accuracy than the white, black and gray box models used to compose the ensemble of predictors. At the same time, it has also revealed an important result, namely that none of such technique outperforms all the others in all the cases: a thorough analysis of the experimental results has highlighted how the effectiveness of any of the proposed approaches depends, in fact, on the characteristics of the target performance function and of the baseline predictors that compose the ensemble.

6 Conclusions and Future Work

Performance modeling of applications and computer systems is a fundamental building block to implement important tasks like capacity planning, automatic resource provisioning and anomaly detection. White and black box modeling are the two most prominent techniques applied to tackle this important issue: the former exploits *a priori* knowledge about the internals of the target application/system to predict its performance; the latter infers a statistical performance model by means of a so-called *training phase*.

Unfortunately, the increasing complexity of modern applications is challenging the effectiveness of the two most prominent approaches to performance modeling, namely white box and black box modeling: on one side, white box solutions rely on assumptions and approximations which may not hold in realistic scenarios; on the other one, black box models are typically accurate only in regions of the configuration space that have been experimented with during the training phase.

This dissertation has proposed several contributions in the field of gray box performance modeling, a methodology that aims aims at reconciling the white and the black box techniques into a hybrid paradigm, with the purpose of achieving the best of the two worlds: reduced training time, increased robustness in extrapolation, and ability to correct initial inaccuracies by incorporating new factual knowledge when available.

In particular, this dissertation has proposed and investigated two box modeling techniques, which complement white and black box modeling according to different principles.

- *Divide et impera:* this approach allows for the joint usage of white and black box modeling techniques, each capturing the performance of distinct, yet possibly inter-related, modules of the target system. The key advantage of this technique is that it allows for decomposing the problem of predicting the performance of a complex system into a set

of simpler sub-models, which can exploit the modeling methodology (i.e., white box or black box) that better fits the characteristics of the considered system's module.

- *Hybrid Ensemble:* this approach is based on the idea of combining the output of a set of white and black box models with the purpose of generating a single model with a higher predictive accuracy than any of its constituent parts. In particular, the role of the black box components is to complement the predictions provided by one or more base white box models by correcting their possible inaccuracies. This dissertation has investigated two principles implementing this approach: *i*) Patching, which progressively learns how to correct the predictions generated by the white box models. Such a goal is pursued by accumulating knowledge on the errors of such models in various regions of their parameter space; and *ii*) Selection, which employs the performance model (either white or black) that is expected to maximize accuracy depending on the configuration/workload whose performance is being predicted.

  The *Hybrid Ensemble* hybrid methodology has been instantiated in four approaches, namely Bootstrapping, Hybrid Boosting, Hybrid KNN and Probing: the first two embody the Patching principle; the other two implement the Selection principle.

The viability and effectiveness of *Divide et impera* approach have been evaluated by applying it to model the performance of Distributed Transactional Platforms. This kind of applications, in fact, represents the archetype of complex application, as it embodies several among the functional (e.g., distribution and replication) and business (e.g., virtualization) requirements of modern systems.

In this dissertation the *Divide et impera* approach is employed in the context of DTP modeling as follows. Analytical models are employed to capture the performance of transactional consistency protocols, whose algorithmic dynamics are fully specified and are thus amenable to be described via white box approaches. The white box models proposed in this dissertation target a wide region of the design space of DTP systems, encompassing different protocols employed to regulate concurrency and replications. Black box modeling is used to predict response

time of operations that require distributed synchronization, which would be hard to predict accurately in environments, such as the Cloud, in which little or no knowledge is provided on the underlying physical (e.g., networking) infrastructure.

Hybrid Ensemble methods, instead, have been applied not only to the problem of DTP performance modeling, but also to predict performance of a Total Order Broadcast primitive, which is a fundamental building block of many distributed transactional systems.

A thorough experimental evaluation on the two case studies has shown that the proposed gray box modeling techniques are capable of delivering higher predictive accuracy than pure white and black box counterparts, while requiring less training data. On the other hand, it has also revealed the sensitivity of the proposed *Hybrid Ensemble* approaches to different aspects of the target performance prediction problem, such as the error distribution of the models that compose the ensemble. This has led to the conclusion that no one-size-fits-all *Hybrid Ensemble* technique exists that performs the best across all possible cases and that, therefore, selecting the right hybrid modeling technique is an important step to achieve high predictive accuracy.

The investigation on gray box modeling can be extended in a number of directions. The experimental evaluation has already shown that the *Divide et impera* and the Hybrid Ensemble approaches can be leveraged in synergy: a first research avenue would be, therefore, to explore the possibility to further combine gray box ensemble models, so as to build a hybrid, more accurate *meta*-ensemble.

A second issue that deserves further investigation is devising ad-hoc heuristics or algorithms to efficiently determine which among the proposed gray box modeling techniques fits better the characteristics of the target performance function, of the available white box models and training data at hand. Indeed, an important finding of this work is that none among the proposed techniques consistently outperforms all the others in all scenarios.

Finally, an interesting extension to this work would be implementing efficient active learning algorithms in which white box models are not only used as starting point to be complemented by black box learners, but are also exploited to drive the training phase of the black box models, with the purpose of increasing the representativeness of the collected samples.

Bibliography

Aerospike. (2014). *Aerospike*. http://www.aerospike.com.

Agrawal, R., Carey, M. J., & McVoy, L. W. (1987). The performance of alternative strategies for dealing with deadlocks in database management systems. *Software Engineering, IEEE Transactions on*(12), 1348–1363.

Alonso, R., Barbara, D., & Garcia-Molina, H. (1990, September). Data caching issues in an information retrieval system. *ACM Trans. Database Syst.*, *15*(3), 359–384. Available from http://doi.acm.org/10.1145/88636.87848

Amazon. (2013). *Amazon S3*. http://aws.amazon.com/s3/.

Amazon. (2015a). *Amazon Autoscaling*. http://aws.amazon.com/it/autoscaling/.

Amazon. (2015b). *SimpleDB*. http://aws.amazon.com/it/simpledb/.

Apache Software Foundation. (2015). *CouchDB*. http://www.couchdb.apache.org.

Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys*, *4*, 40–79. Available from https://hal.archives-ouvertes.fr/hal-00407906 (Published in Statistics Surveys (2010) 4, 40-79)

Astrom, K. J., & Eykhoff, P. (1971, March). System identification-a survey. *Automatica*, *7*(2), 123–162.

Att, K. L., & Leung, K. K. (1997). An update algorithm for replicated signaling databases in wireless and advanced intelligent networks. *IEEE Transactions on Computers*.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*.

Bailis, P., Fekete, A., Hellerstein, J. M., Ghodsi, A., & Stoica, I. (2014). Scalable atomic visibility with ramp transactions. In *Proceedings of the 2014 acm sigmod international conference on management of data* (pp. 27–38). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2588555.2588562

Baker, J., et al. (2011). Megastore: Providing scalable, highly available storage for interactive services. In *Proc. of cidr*.

Ban, B. (2012). *JGroups - A Toolkit for Reliable Multicast Communication.* http://www.jgroups.org.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al. (2003, October). Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, *37*(5), 164–177. Available from http://doi.acm.org/10.1145/1165389.945462

Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., & O'Neil, P. (1995). A critique of ansi sql isolation levels. In *Proceedings of the 1995 acm sigmod international conference on management of data* (pp. 1–10). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/223784.223785

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011, December). Algorithms for Hyper-Parameter Optimization. In *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)* (Vol. 24). Granada, Spain: Neural Information Processing Systems Foundation.

Bernstein, P. A. e. a. (1986). *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc.

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag New York, Inc.

Breiman, L. (1996, August). Bagging predictors. *Mach. Learn.*, *24*(2), 123–140. Available from http://dx.doi.org/10.1023/A:1018054314350

Budhiraja, N., Marzullo, K., Schneider, F. B., & Toueg, S. (1993). Distributed systems (2nd ed.). In S. Mullender (Ed.), (pp. 199–216). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. Available from http://dl.acm.org/citation.cfm?id=302430.302438

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009, June). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, *25*(6), 599–616. Available from http://dx.doi.org/10.1016/j.future.2008.12.001

Cachin, C., Guerraoui, R., & Rodrigues, L. (2011). *Introduction to reliable and secure distributed programming (2. ed.)*. Springer.

Carey, M. J., & Livny, M. (1988). Distributed concurrency control performance: A study of algorithms, distribution, and replication. In *Proceedings of the 14th international conference on very large data bases* (pp. 13–25). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from http://dl.acm.org/citation.cfm?id=645915.671801

Carey, M. J., & Livny, M. (1991, December). Conflict detection tradeoffs for replicated data. *ACM Trans. Database Syst.*, *16*(4), 703–746. Available from http://doi.acm.org/10.1145/115302.115289

Caruana, R., et al. (2004). Ensemble selection from libraries of models. In *Proc. of icml*.

Charron-Bost, B., Pedone, F., & Schiper, A. (Eds.). (2010). *Replication: Theory and practice*. Berlin, Heidelberg: Springer-Verlag.

Chen, J., Soundararajan, G., & Amza, C. (2006). Autonomic provisioning of backend databases in dynamic content web servers. In *Proceedings of the 2006 ieee international conference on autonomic computing* (pp. 231–242). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/ICAC.2006.1662403

Chen, J., Soundararajan, G., Ghanbari, S., & Amza, C. (2013, April). Model ensemble tools for self-management in data centers. , 36-43.

Ciciani, B., Dias, D. M., & Yu, P. S. (1990, June). Analysis of replication in distributed database systems. *IEEE Trans. on Knowl. and Data Eng.*, *2*(2), 247–261. Available from http://dx.doi.org/10.1109/69.54723

Ciciani, B., Dias, D. M., & Yu, P. S. (1992, October). Analysis of concurrency-coherency control protocols for distributed transaction processing systems with regional locality. *IEEE Trans. Softw. Eng.*, *18*(10), 899–914. Available from http://dx.doi.org/10.1109/32.163606

Ciciani, B., Didona, D., Di Sanzo, P., Palmieri, R., Peluso, S., Quaglia, F., et al. (2012). Automated workload characterization in cloud-based transactional data grids. In *Proceedings of the 2012 ieee 26th international parallel and distributed processing symposium workshops and phd forum* (pp. 1525–1533). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/IPDPSW.2012.192

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st acm symposium on cloud computing* (pp. 143–154). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1807128.1807152

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*.

Cost, S., & Salzberg, S. (1993, January). A weighted nearest neighbor algorithm for learning with symbolic features. *Mach. Learn.*, *10*(1), 57–78.

Couceiro, M., Didona, D., Rodrigues, L., & Romano, P. (2015). Self-tuning in distributed transactional memory. In R. Guerraoui & P. Romano (Eds.), *Transactional memory. foundations, algorithms, tools, and applications* (Vol. 8913, p. 418-448). Springer International Publishing.

Couceiro, M., Romano, P., Carvalho, N., & Rodrigues, L. (2009). D2stm: Dependable distributed software transactional memory. In *Proceedings of the 2009 15th ieee pacific rim international symposium on dependable computing* (pp. 307–313). Washington, DC,

USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/PRDC.2009.55

Couceiro, M., Romano, P., & Rodrigues, L. (2010). A machine learning approach to performance prediction of total order broadcast protocols. In *Self-adaptive and self-organizing systems (saso), 2010 4th ieee international conference on* (p. 184-193).

Couceiro, M., Romano, P., & Rodrigues, L. (2011). Polycert: Polymorphic self-optimizing replication for in-memory transactional grids. In *Proceedings of the 12th acm/ifip/usenix international conference on middleware* (pp. 309–328). Berlin, Heidelberg: Springer-Verlag. Available from http://dx.doi.org/10.1007/978-3-642-25821-3_16

Couceiro, M., Ruivo, P., Romano, P., & Rodrigues, L. (2013). Chasing the optimum in replicated in-memory transactional platforms via protocol adaptation. In *Proceedings of the 2013 43rd annual ieee/ifip international conference on dependable systems and networks (dsn)* (pp. 1–12). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/DSN.2013.6575311

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory*.

Cruz, F., Maia, F., Matos, M., Oliveira, R., Paulo, J. a., Pereira, J., et al. (2013). Met: Workload aware elasticity for nosql. In *Proceedings of the 8th acm european conference on computer systems* (pp. 183–196). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2465351.2465370

Curino, C., Jones, E., Zhang, Y., & Madden, S. (2010, September). Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, *3*(1-2), 48–57. Available from http://dx.doi.org/10.14778/1920841.1920853

Das, S., Agrawal, D., & El Abbadi, A. (2010). G-store: A scalable data store for transactional multi key access in the cloud. In *Proceedings of the 1st acm symposium on cloud computing* (pp. 163–174). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1807128.1807157

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., et al. (2007). Dynamo: Amazon's highly available key-value store. In *Proceedings of twenty-first acm sigops symposium on operating systems principles* (pp. 205–220). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1294261.1294281

Dejun, J., Pierre, G., & Chi, C.-H. (2009). Ec2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the 2009 international conference on service-oriented computing* (pp. 197–207). Berlin, Heidelberg: Springer-Verlag. Available from http://dl.acm.org/citation.cfm?id=1926618.1926641

Dejun, J., Pierre, G., & Chi, C.-H. (2010). Autonomous resource provisioning for multi-service web applications. In *Proceedings of the 19th international conference on world wide web* (pp. 471–480). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1772690.1772739

Dejun, J., Pierre, G., & Chi, C.-H. (2011). Resource provisioning of web applications in heterogeneous clouds. In *Proceedings of the 2nd usenix conference on web application development* (pp. 5–5). Berkeley, CA, USA: USENIX Association. Available from http://dl.acm.org/citation.cfm?id=2002168.2002173

Delimitrou, C., & Kozyrakis, C. (2013, March). Paragon: Qos-aware scheduling for heterogeneous datacenters. *SIGARCH Comput. Archit. News*, *41*(1), 77–88. Available from http://doi.acm.org/10.1145/2490301.2451125

Di Sanzo, P., Ciciani, B., Quaglia, F., & Romano, P. (2008, Sept). A performance model of multi-version concurrency control. In *Modeling, analysis and simulation of computers and telecommunication systems, 2008. mascots 2008. ieee international symposium on* (p. 1-10).

Di Sanzo, P., Del Re, F., Rughetti, D., Ciciani, B., & Quaglia, F. (2013, Sept). Regulating

concurrency in software transactional memory: An effective model-based approach. In *Self-adaptive and self-organizing systems (saso), 2013 ieee 7th international conference on* (p. 31-40).

Di Sanzo, P., Molfese, F., Rughetti, D., & Ciciani, B. (2014). Providing transaction class-based qos in in-memory data grids via machine learning. In *Proceedings of the 2014 ieee 3rd symposium on network cloud computing and applications (ncca 2014)* (pp. 46–53). Washington, DC, USA: IEEE Computer Society. Available from `http://dx.doi.org/10.1109/NCCA.2014.16`

Di Sanzo, P., Palmieri, R., Ciciani, B., Quaglia, F., & Romano, P. (2010). Analytical modeling of lock-based concurrency control with arbitrary transaction data access patterns. In *Proceedings of the first joint wosp/sipew international conference on performance engineering* (pp. 69–78). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/1712605.1712619`

Di Sanzo, P., Quaglia, F., Ciciani, B., Pellegrini, A., Didona, D., Romano, P., et al. (2015). A flexible framework for accurate simulation of cloud in-memory data stores. *Simulation Modelling Practice and Theory*(0), -. Available from `http://www.sciencedirect.com/science/article/pii/S1569190X15000945`

Di Sanzo, P., Rughetti, D., Ciciani, B., & Quaglia, F. (2012). Auto-tuning of cloud-based in-memory transactional data grids via machine learning. In *Proceedings of the 2012 second symposium on network cloud computing and applications* (pp. 9–16). Washington, DC, USA: IEEE Computer Society. Available from `http://dx.doi.org/10.1109/NCCA.2012.20`

Didona, D., Carnevale, D., Galeani, S., & Romano, P. (2012). An extremum seeking algorithm for message batching in total order protocols. In *Proceedings of the 2012 ieee sixth international conference on self-adaptive and self-organizing systems* (pp. 89–98). Washington, DC, USA: IEEE Computer Society. Available from `http://dx.doi.org/10.1109/SASO.2012.33`

Didona, D., Felber, P., Harmanci, D., Romano, P., & Schenker, J. (2013a). Identifying the optimal level of parallelism in transactional memory applications. In *International conference on networked systems* (p. 233-247).

Didona, D., Felber, P., Harmanci, D., Romano, P., & Schenker, J. (2013b). Identifying the optimal level of parallelism in transactional memory applications. *Springer Computing*, 1-21.

Didona, D., Quaglia, F., Romano, P., & Torre, E. (2015). Enhancing performance prediction robustness by combining analytical modeling and machine learning. In *Proceedings of the 6th acm/spec international conference on performance engineering* (pp. 145–156). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2668930.2688047

Didona, D., & Romano, P. (2014a). On Bootstrapping Machine Learning Performance Predictors via Analytical Models. *ArXiv e-prints*.

Didona, D., & Romano, P. (2014b). Performance modelling of partially replicated in-memory transactional stores. In *Modelling, analysis simulation of computer and telecommunication systems (mascots), 2014 ieee 22nd international symposium on* (p. 265-274).

Didona, D., & Romano, P. (2014c). Self-tuning transactional data grids: The cloud-tm approach. In *Proceedings of the symposium on network cloud computing and applications, (ncca)* (p. 113-120). IEEE.

Didona, D., & Romano, P. (2015). Hybrid machine learning/analytical models for performance prediction: A tutorial. In *Proceedings of the 6th acm/spec international conference on performance engineering* (pp. 341–344). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2668930.2688823

Didona, D., Romano, P., Peluso, S., & Quaglia, F. (2012). Transactional auto scaler: Elastic scaling of in-memory transactional data grids. In *Proceedings of the 9th international conference on autonomic computing* (pp. 125–134). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2371536.2371559

Didona, D., Romano, P., Peluso, S., & Quaglia, F. (2014, July). Transactional auto scaler: Elastic scaling of replicated in-memory transactional data grids. *ACM Trans. Auton. Adapt. Syst.*, *9*(2), 11:1–11:32.

Diegues, N., & Romano, P. (2015). Bumper: Sheltering distributed transactions from conflicts. *Future Generation Computer Systems*, *51*(0), 20 - 35.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of the first international workshop on multiple classifier systems* (pp. 1–15). London, UK, UK: Springer-Verlag.

Elnikety, S., Dropsho, S., Cecchet, E., & Zwaenepoel, W. (2009). Predicting replicated database scalability from standalone database profiling. In *Proceedings of the 4th acm european conference on computer systems* (pp. 303–316). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/1519065.1519098`

eXistdb. (2014). *eXistdb.* `http://exist-db.org`.

Eyerman, S., Hoste, K., & Eeckhout, L. (2011). Mechanistic-empirical processor performance modeling for constructing cpi stacks on real hardware. In *Proceedings of the ieee international symposium on performance analysis of systems and software* (pp. 216–226). Washington, DC, USA: IEEE Computer Society. Available from `http://dx.doi.org/10.1109/ISPASS.2011.5762738`

Faleiro, J. M., Thomson, A., & Abadi, D. J. (2014). Lazy evaluation of transactions in database systems. In *Proceedings of the 2014 acm sigmod international conference on management of data* (pp. 15–26). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/2588555.2610529`

Fox, G. C., Laszewski, G. von, Diaz, J., Keahey, K., Fortes, J., Figueiredo, R., et al. (2012). *Futuregrid: A reconfigurable testbed for cloud, hpc and grid computing*. Chapman & Hall.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, *95*(2), 337-407.

Friedman, T., & Renesse, R. V. (1997). Packing messages as a tool for boosting the performance of total ordering protocls. In *Proceedings of the 6th ieee international symposium on high performance distributed computing* (pp. 233–). Washington, DC, USA: IEEE Computer Society. Available from http://dl.acm.org/citation.cfm?id=822082.823140

Fujimoto, R. M. (1990, October). Parallel discrete event simulation. *Commun. ACM*, *33*(10), 30–53. Available from http://doi.acm.org/10.1145/84537.84545

Galante, G., & Bona, L. C. E. d. (2012). A survey on cloud computing elasticity. In *Proceedings of the 2012 ieee/acm fifth international conference on utility and cloud computing* (pp. 263–270). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/UCC.2012.30

Gallersdörfer, R., & Nicola, M. (1995). Improving performance in replicated databases through relaxed coherency. In *Proceedings of the 21th international conference on very large data bases* (pp. 445–456). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from http://dl.acm.org/citation.cfm?id=645921.673130

Ganapathi, A., Kuno, H., Dayal, U., Wiener, J. L., Fox, A., Jordan, M., et al. (2009). Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Proceedings of the 2009 ieee international conference on data engineering* (pp. 592–603). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/ICDE.2009.130

Garcia-Molina, H. (1979). *Performance of update algorithms for replicated data in a distributed database.* Unpublished doctoral dissertation, Stanford, CA, USA. (AAI8001920)

Ghahramani, Z. (2004). Unsupervised learning. In *Advanced lectures on machine learning.*

Ghanbari, S., Soundararajan, G., Chen, J., & Amza, C. (2007). Adaptive learning of metric correlations for temperature-aware database provisioning. In *Proceedings of the fourth international conference on autonomic computing* (pp. 26–). Washington, DC, USA:

IEEE Computer Society. Available from http://dx.doi.org/10.1109/ICAC.2007.3

Google. (2015a). *Cloud Datastore.* https://cloud.google.com/datastore/.

Google. (2015b). *Google Cloud Platform Autoscaler.* https://cloud.google.com/compute/docs/autoscaler/.

Gray, J., Helland, P., O'Neil, P., & Shasha, D. (1996, June). The dangers of replication and a solution. *SIGMOD Rec.*, *25*(2), 173–182. Available from http://doi.acm.org/10.1145/235968.233330

Guerraoui, R., & Schiper, A. (1997, April). Software-based replication for fault tolerance. *Computer*, *30*(4), 68–74. Available from http://dx.doi.org/10.1109/2.585156

Hall, M., et al. (2009, November). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, *11*(1), 10–18.

Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on nosql database. In *Pervasive computing and applications (icpca), 2011 6th international conference on* (pp. 363–366).

Harchol-Balter, M. (2013). *Performance modeling and design of computer systems: Queueing theory in action* (1st ed.). New York, NY, USA: Cambridge University Press.

Harizopoulos, S., Abadi, D. J., Madden, S., & Stonebraker, M. (2008). Oltp through the looking glass, and what we found there. In *Proceedings of the 2008 acm sigmod international conference on management of data* (pp. 981–992). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1376616.1376713

Haykin, S. (1998). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Heiss, H.-U., & Wagner, R. (1991). Adaptive load control in transaction processing systems. In *Proceedings of the 17th international conference on very large data bases* (pp. 47–54). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from http://dl.acm.org/citation.cfm?id=645917.672321

Herlihy, M., & Moss, J. E. B. (1993). Transactional memory: architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*.

Herodotou, H., Dong, F., & Babu, S. (2011). No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proc. of the acm symposium on cloud computing (socc)*.

Hong, S., & Kim, H. (2009). An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the 36th annual international symposium on computer architecture* (pp. 152–163). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1555754.1555775

Hwang, S.-Y., Lee, K. K. S., & Chin, Y. H. (1996). Data replication in a distributed system: A performance study. In *Proceedings of the 7th international conference on database and expert systems applications* (pp. 708–717). London, UK, UK: Springer-Verlag. Available from http://dl.acm.org/citation.cfm?id=648309.754404

IBM Corp. (2004). *An architectural blueprint for autonomic computing*. USA: IBM Corp. Available from www-3.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf

IEEE. (2014). *400 Gb/s Ethernet Study Group.* http://www.ieee802.org/3/400GSG/.

Jiménez-Peris, R., Patiño-Martínez, M., Magoutis, K., Bilas, A., & Brondino, I. (2012). Cumulonimbo: A highly-scalable transaction processing platform as a service. *ERCIM News*, *2012*(89). Available from http://dblp.uni-trier.de/db/journals/ercim/ercim2012.html#Jimenez-PerisPMBB12

Juan, D.-C. (2014). *A learning-based framework incorporating domain knowledge for performance modeling*. Unpublished doctoral dissertation, CARNEGIE MELLON UNIVERSITY.

Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., et al. (2008). H-Store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.*, *1*(2), 1496–1499. Available from http://hstore.cs.brown.edu/papers/hstore-demo.pdf

Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., & Lewin, D. (1997). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual acm symposium on theory of computing* (pp. 654–663). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/258533.258660`

Kleinrock, L. (1975). *Queueing systems* (Vol. I: Theory). Wiley Interscience.

Kleinrock, L. (1976). *Queueing systems, vol. 2*. New York, NY: Wiley.

Knottenbelt, W., Zertal, S., & Harrison, P. (2001, Jul). Performance analysis of three implementation strategies for distributed lock management. *Computers and Digital Techniques, IEE Proceedings -*, *148*(45), 176-187.

Kobrix Software. (2014). *HypergraphDB*. `http://www.hypergraphdb.org/index`.

Kobus, T., Kokocinski, M., & Wojciechowski, P. T. (2013). Hybrid replication: State-machine-based and deferred-update replication schemes combined. In *Proc. of the 2013 ieee 33rd international conference on distributed computing systems* (pp. 286–296). Washington, DC, USA: IEEE Computer Society. Available from `http://dx.doi.org/10.1109/ICDCS.2013.30`

Kotselidis, C., Ansari, M., Jarvis, K., Luján, M., Kirkham, C., & Watson, I. (2008). Distm: A software transactional memory framework for clusters. In *Proceedings of the 2008 37th international conference on parallel processing* (pp. 51–58). Washington, DC, USA: IEEE Computer Society. Available from `http://dx.doi.org/10.1109/ICPP.2008.59`

Kristensen, N. R., Madsen, H., & Jørgensen, S. B. (2004). Parameter estimation in stochastic grey-box models. *Automatica*, *40*(2), 225–237.

Kuang, Y., & Mukkamala, R. (1991, Apr). Performance analysis of static locking in replicated distributed database systems. In *Southeastcon '91., ieee proceedings of* (p. 698-701 vol.2).

Lakshman, A., & Malik, P. (2010, April). Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, *44*(2), 35–40. Available from http://doi.acm.org/10.1145/1773912.1773922

Lamport, L. (1998, May). The part-time parliament. *ACM Trans. Comput. Syst.*, *16*(2), 133–169. Available from http://doi.acm.org/10.1145/279227.279229

Lamport, L. (2003). Future directions in distributed computing. In A. Schiper, A. A. Shvartsman, H. Weatherspoon, & B. Y. Zhao (Eds.), (pp. 22–23). Berlin, Heidelberg: Springer-Verlag. Available from http://dl.acm.org/citation.cfm?id=1809315.1809321

Laszewski, G. von, Fox, G. C., Wang, F., Younge, A. J., Kulshrestha, A., Pike, G. G., et al. (2010, 11/2010). Design of the futuregrid experiment management framework. In *Gce2010 at sc10*. New Orleans: IEEE.

Leutenegger, S. T., & Dias, D. (1993). A modeling study of the tpc-c benchmark. In *Sigmod record*.

Li, C., Porto, D., Clement, A., Gehrke, J., Preguiça, N., & Rodrigues, R. (2012). Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the 10th usenix conference on operating systems design and implementation* (pp. 265–278). Berkeley, CA, USA: USENIX Association. Available from http://dl.acm.org/citation.cfm?id=2387880.2387906

Little, J. D. (1961). A proof for the queuing formula: L= λ w. *Operations research*, *9*(3), 383–387.

Little, J. D. (2011). Or forum-little's law as viewed on its 50th anniversary. *Operations Research*, *59*(3), 536–549.

Liu, H., Hussain, F., Tan, C., & Dash, M. (2002). Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, *6*(4), 393-423. Available from http://dx.doi.org/10.1023/A%3A1016304305535

Ljung, L. (Ed.). (1999). *System identification (2nd ed.): Theory for the user*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Marchioni, F., & Surtani, M. (2012). *Infinispan data grid platform*. Packt Publishing.

Martínez-Bazan, N., Muntés-Mulero, V., Gómez-Villamor, S., Nin, J., Sánchez-Martínez, M.-A., & Larriba-Pey, J.-L. (2007). Dex: High-performance exploration on large graphs for information retrieval. In *Proceedings of the sixteenth acm conference on conference on information and knowledge management* (pp. 573–582). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/1321440.1321521`

McDermott, J. P., & Mukkamala, R. (1994). Performance analysis of transaction management algorithms for the sintra replicated-architecture database system. In *Proceedings of the ifip wg11.3 working conference on database security vii* (pp. 215–234). Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co. Available from `http://dl.acm.org/citation.cfm?id=646113.679758`

Meling, H., Montresor, A., Helvik, B. E., & Babaoglu, O. (2008, July). Jgroup-arm: A distributed object group platform with autonomous replication management. *Softw. Pract. Exper.*, *38*(9), 885–923.

Menasce, D. A., & Almeida, V. (2001). *Capacity planning for web services: Metrics, models, and methods* (1st ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Menasce, D. A., Dowdy, L. W., & Almeida, V. A. F. (2004). *Performance by design: Computer capacity planning by example*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Menascé, D. A., & Nakanishi, T. (1982). Performance evaluation of a two-phase commit based protocol for ddbs. In *Proceedings of the 1st acm sigact-sigmod symposium on principles of database systems* (pp. 247–255). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/588111.588152`

Mendes-Moreira, J. a., Soares, C., Jorge, A. M., & Sousa, J. F. D. (2012, December). Ensemble approaches for regression: A survey. *ACM Comput. Surv.*, *45*(1), 10:1–10:40. Available from `http://doi.acm.org/10.1145/2379776.2379786`

Microsoft. (2015a). *Microsoft Azure: How to Scale an Application.* https://azure.microsoft.com/en-gb/documentation/articles/cloud-services-how-to-scale/#automatically-scale-an-application-running-web-roles-worker-roles-or-virtual-machines.

Microsoft. (2015b). *Microsoft Azure SQL Database.* http://azure.microsoft.com/en-us/services/sql-database/.

Miranda, H., Pinto, A., & Rodrigues, L. (2001, Apr). Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Distributed computing systems, 2001. 21st international conference on.* (p. 707-710).

mongoDB inc. (2015). *mongoDB.* http://www.mongodb.org.

Mozafari, B., Curino, C., Jindal, A., & Madden, S. (2013). Performance and resource modeling in highly-concurrent oltp workloads. In *Proceedings of the 2013 acm sigmod international conference on management of data* (pp. 301–312). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2463676.2467800

Mukkamala, R., & Bruell, S. C. (1990). Efficient schemes to evaluate transaction performance in distributed database systems. *The Computer Journal*, *33*(1), 79-89. Available from http://comjnl.oxfordjournals.org/content/33/1/79.abstract

Neo4j. (2015). *Neo4j.* http://www.neo4j.org.

Nicola, M., & Jarke, M. (2000). Performance modeling of distributed and replicated databases. *IEEE Trans. on Knowl. and Data Eng.*.

Objectivity. (2015). *InfiniteGraph.* http://www.objectivity.com.

Ogata, K. (2001). *Modern control engineering* (4th ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Oracle. (2011). *Oracle Coherence.* http://www.oracle.com/technetwork/middleware/coherence/overview/index.html.

Orientechnologies. (2014). *OrientDB*. http://www.orientechnologies.com/orientdb/.

Owen, S., Anil, R., Dunning, T., & Friedman, E. (2011). *Mahout in action*. Greenwich, CT, USA: Manning Publications Co.

Padhye, J., Firoiu, V., Towsley, D., & Kurose, J. (1998, October). Modeling tcp throughput: A simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.*, *28*(4), 303–314. Available from http://doi.acm.org/10.1145/285243.285291

Paiva, J. a., Ruivo, P., Romano, P., & Rodrigues, L. (2014, December). Autoplacer: Scalable self-tuning data placement in distributed key-value stores. *ACM Trans. Auton. Adapt. Syst.*, *9*(4), 19:1–19:30. Available from http://doi.acm.org/10.1145/2641573

Palmieri, R., Sanzo, P. di, Quaglia, F., Romano, P., Peluso, S., & Didona, D. (2011). Integrated monitoring of infrastructures and applications in cloud environments. In *Proc. of the 2011 international conference on parallel processing*.

Palmieri, R., Sanzo, P. di, Quaglia, F., Romano, P., Peluso, S., & Didona, D. (2012). Integrated monitoring of infrastructures and applications in cloud environments. In *Proceedings of the 2011 international conference on parallel processing* (pp. 45–53). Berlin, Heidelberg: Springer-Verlag.

Papoulis, A. (1991). *Probability, random variables and stochastic processes* (3rd ed.). McGraw-Hill.

Pedone, F., Guerraoui, R., & Schiper, A. (2003, July). The database state machine approach. *Distrib. Parallel Databases*, *14*(1), 71–98. Available from http://dx.doi.org/10.1023/A:1022887812188

Peluso, S., Ruivo, P., Romano, P., Quaglia, F., & Rodrigues, L. (2012, June). When scalability meets consistency: Genuine multiversion update-serializable partial data replication. In *Distributed computing systems (icdcs), 2012 ieee 32nd international conference on* (p. 455-465).

Perez-Sorrosal, F., Martinez, M. Patiño, Jimenez-Peris, R., & Kemme, B. (2011, December). Elastic si-cache: Consistent and scalable caching in multi-tier architectures. *The VLDB Journal*, *20*(6), 841–865. Available from http://dx.doi.org/10.1007/s00778-011-0228-8

Petri, C. A. (1966). *Communication with automata*. Unpublished doctoral dissertation, Universität Hamburg.

Pfister, G. F. (2001). An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*, *42*, 617–632.

Plattner, H., & Zeier, A. (2011). *In-memory data management: An inflection point for enterprise applications* (1st ed.). Springer Publishing Company, Incorporated.

Project Voldemort. (2015). *Voldemort.* http://www.project-voldemort.com/voldemort/.

Quinlan, J. R. (1986, March). Induction of decision trees. *Mach. Learn.*, *1*(1), 81–106. Available from http://dx.doi.org/10.1023/A:1022643204877

Quinlan, J. R. (1993a). *C4.5: Programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Quinlan, J. R. (1993b). *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers Inc.

Quinlan, J. R. (1996, March). Improved use of continuous attributes in c4.5. *J. Artif. Int. Res.*, *4*(1), 77–90. Available from http://dl.acm.org/citation.cfm?id=1622737.1622742

Quinlan, J. R. (2012). *Rulequest Cubist.* http://www.rulequest.com/cubist-info.html.

Raghuram, A., Morgan, T. W., Rajaraman, B., & Ronen, Y. (1992, May). Approximation for the mean value performance of locking algorithms for distributed database systems: A partitioned database. *Ann. Oper. Res.*, *36*(1-4), 299–346. Available from http://dx.doi.org/10.1007/BF02094335

Redis. (2014). *Redis*. http://www.basho.com/riak/.

Reiser, M., & Lavenberg, S. S. (1980, April). Mean-value analysis of closed multichain queuing networks. *J. ACM*, *27*(2), 313–322. Available from http://doi.acm.org/10.1145/322186.322195

Ren, J. F., Tokahashi, Y., & Hasegawa, T. (1996, July). Analysis of impact of network delay on multiversion conservative timestamp algorithms in ddbs. *Perform. Eval.*, *26*(1), 21–50. Available from http://dx.doi.org/10.1016/0166-5316(95)00019-4

Robbins, H. (1985). Some aspects of the sequential design of experiments. , 169-177. Available from http://dx.doi.org/10.1007/978-1-4612-5110-1_13

Romano, P., & Leonetti, M. (2012, Jan). Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning. In *Computing, networking and communications (icnc), 2012 international conference on* (p. 786-792).

Romano, P., Rodrigues, L., Carvalho, N., & Cachopo, J. (2010). Cloud-tm: harnessing the cloud with distributed transactional memories. *SIGOPS Operating Systems Review*, *44*.

Rughetti, D., Di Sanzo, P., Ciciani, B., & Quaglia, F. (2014, May). Analytical/ml mixed approach for concurrency regulation in software transactional memory. In *Cluster, cloud and grid computing (ccgrid), 2014 14th ieee/acm international symposium on* (p. 81-91).

Ruivo, P., Couceiro, M., Romano, P., & Rodrigues, L. (2011). Exploiting total order multicast in weakly consistent transactional caches. In *Proceedings of the 2011 ieee 17th pacific rim international symposium on dependable computing* (pp. 99–108). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/PRDC.2011.21

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Pearson Education.

Sander, J., Ester, M., Kriegel, H.-P., & Xu, X. (1998, June). Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Min. Knowl. Discov.*, *2*(2), 169–194. Available from http://dx.doi.org/10.1023/A:1009745219419

Santos, N., & Schiper, A. (2013, July). Optimizing paxos with batching and pipelining. *Theor. Comput. Sci.*, *496*, 170–183. Available from http://dx.doi.org/10.1016/j.tcs.2012.10.002

Schapire, R. E. (1999). A brief introduction to boosting. In *Proceedings of the 16th international joint conference on artificial intelligence - volume 2* (pp. 1401–1406). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from http://dl.acm.org/citation.cfm?id=1624312.1624417

Schroeder, B., Harchol-Balter, M., Iyengar, A., Nahum, E., & Wierman, A. (2006, April). How to determine a good multi-programming level for external scheduling. In *Data engineering, 2006. icde '06. proceedings of the 22nd international conference on* (p. 60-60).

Settles, B. (2009). *Active learning literature survey* (Computer Sciences Technical Report No. 1648). University of Wisconsin–Madison.

Sharma, U., Shenoy, P., & Towsley, D. F. (2012). Provisioning multi-tier cloud applications using statistical bounds on sojourn time. In *Proceedings of the 9th international conference on autonomic computing* (pp. 43–52). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2371536.2371545

Sheth, A. P., Singhal, A., & Liu, M. T. (1985, October). An analysis of the effect of network parameters on the performance of distributed database systems. *IEEE Trans. Softw. Eng.*, *11*(10), 1174–1184. Available from http://dx.doi.org/10.1109/TSE.1985.231865

Shyu, S.-C., & Li, V. O. K. (1990, June). Performance analysis of static locking in distributed database systems. *IEEE Trans. Comput.*, *39*(6), 741–751. Available from http://dx.doi.org/10.1109/12.53595

Simha, R., & Majumdar, A. (1997). An urn model with applications to database performance evaluation. *Computers and Operations Research*, *24*(4), 289 - 300. Available from http://www.sciencedirect.com/science/article/pii/S0305054896000639

Singh, R., Sharma, U., Cecchet, E., & Shenoy, P. (2010). Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proceedings of the 7th international conference on autonomic computing* (pp. 21–30). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1809049.1809053

Singhal, M. (1989, November). Deadlock detection in distributed systems. *Computer*, *22*(11), 37–48. Available from http://dx.doi.org/10.1109/2.43525

Son, S. H., & Haghighi, N. (1990). Performance evaluation of multiversion database systems. In *Proceedings of the sixth international conference on data engineering* (pp. 129–136). Washington, DC, USA: IEEE Computer Society. Available from http://dl.acm.org/citation.cfm?id=645475.654019

Soundararajan, G., Lupei, D., Ghanbari, S., Popescu, A. D., Chen, J., & Amza, C. (2009). Dynamic resource allocation for database servers running on virtual storage. In *Proccedings of the 7th conference on file and storage technologies* (pp. 71–84). Berkeley, CA, USA: USENIX Association. Available from http://dl.acm.org/citation.cfm?id=1525908.1525914

Sovran, Y., Power, R., Aguilera, M. K., & Li, J. (2011). Transactional storage for geo-replicated systems. In *Proceedings of the twenty-third acm symposium on operating systems principles* (pp. 385–400). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/2043556.2043592

Stewart, C., Chakrabarti, A., & Griffith, R. (2013). Zoolander: Efficiently meeting very strict, low-latency slos. In *Proceedings of the 10th international conference on autonomic computing (icac 13)* (pp. 265–277). San Jose, CA: USENIX. Available from https://www.usenix.org/conference/icac13/technical-sessions/presentation/stewart

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001, August). Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, *31*(4), 149–160. Available from http://doi.acm.org/10.1145/964723.383071

Taft, R., Mansour, E., Serafini, M., Duggan, J., Elmore, A. J., Aboulnaga, A., et al. (2014, November). E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proc. VLDB Endow.*, *8*(3), 245–256. Available from http://dx.doi.org/10.14778/2735508.2735514

Tay, Y. C. (2013). *Analytical performance modeling for computer systems, second edition*. Morgan & Claypool Publishers.

Tay, Y. C., Goodman, N., & Suri, R. (1985, December). Locking performance in centralized databases. *ACM Trans. Database Syst.*, *10*(4), 415–462. Available from http://doi.acm.org/10.1145/4879.4880

Tesauro, G., Jong, N. K., Das, R., & Bennani, M. N. (2006). A hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of the 2006 ieee international conference on autonomic computing* (pp. 65–73). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/ICAC.2006.1662383

Tesauro, G., Jong, N. K., Das, R., & Bennani, M. N. (2007). On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*.

Thereska, E., & Ganger, G. R. (2008). Ironmodel: Robust performance models in the wild. , 253–264. Available from http://doi.acm.org/10.1145/1375457.1375486

Thomasian, A. (1993, January). Determining the number of remote sites accessed in distributed transaction processing. *IEEE Trans. Parallel Distrib. Syst.*, *4*(1), 99–103. Available from http://dx.doi.org/10.1109/71.205656

Thomasian, A. (1994). On a more realistic lock contention model and its analysis. In *Proceedings of the tenth international conference on data engineering* (pp. 2–9). Washington, DC, USA: IEEE Computer Society. Available from http://dl.acm.org/citation.cfm?id=645479.655131

Thomasian, A. (1998, March). Concurrency control: Methods, performance, and analysis. *ACM Comput. Surv.*, *30*(1), 70–119. Available from http://doi.acm.org/10.1145/

274440.274443

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of the 19th acm sigkdd international conference on knowledge discovery and data mining* (pp. 847–855).

TPC Council. (2011). *TPC-C Benchmark*. http://www.tpc.org/tpcc.

Trushkowsky, B., Bodík, P., Fox, A., Franklin, M. J., Jordan, M. I., & Patterson, D. A. (2011). The scads director: Scaling a distributed storage system under stringent performance requirements. In *Proceedings of the 9th usenix conference on file and stroage technologies* (pp. 12–12). Berkeley, CA, USA: USENIX Association. Available from http://dl.acm.org/citation.cfm?id=1960475.1960487

Tsoumakos, D., Konstantinou, I., Boumpouka, C., Sioutas, S., & Koziris, N. (2013). Automated, elastic resource provisioning for nosql clusters using tiramola. In (Vol. 0, p. 34-41). Los Alamitos, CA, USA: IEEE Computer Society.

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., & Tantawi, A. (2005). An analytical model for multi-tier internet services and its applications. In *Proceedings of the 2005 acm sigmetrics international conference on measurement and modeling of computer systems* (pp. 291–302). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1064212.1064252

Wada, H., Fekete, A., Zhao, L., Lee, K., & Liu, A. (2011). Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *CIDR 2011, fifth biennial conference on innovative data systems research, asilomar, ca, usa, january 9-12, 2011, online proceedings* (pp. 134–143).

Watkins, C., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, *8*(3-4), 279-292. Available from http://dx.doi.org/10.1023/A%3A1022676722315

Welch, G., & Bishop, G. (1995). *An introduction to the kalman filter* (Tech. Rep. No. 95-041). Department of Computer Science University of North Carolina at Chapel Hill.

Whiteaker, J., et al. (2011, January). Explaining packet delays under virtualization. *SIGCOMM Comput. Commun. Rev.*, *41*(1), 38–44.

Wojciechowski, P. T., Kobus, T., & Kokocinski, M. (2012). Model-driven comparison of state-machine-based and deferred-update replication schemes. In *Proceedings of the 2012 ieee 31st symposium on reliable distributed systems* (pp. 101–110). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/SRDS.2012.44

Wolpert, D. H. (1992, February). Original contribution: Stacked generalization. *Neural Netw.*, *5*(2), 241–259. Available from http://dx.doi.org/10.1016/S0893-6080(05)80023-1

Wolpert, D. H. (1996, October). The lack of a priori distinctions between learning algorithms. *Neural Comput.*, *8*(7), 1341–1390. Available from http://dx.doi.org/10.1162/neco.1996.8.7.1341

Woodside, C. M., Zheng, T., & Litoiu, M. (2008). Performance model estimation and tracking using optimal filters. *IEEE Transactions on Software Engineering*, *34*(3), 391-406.

You, G.-w., Hwang, S.-w., & Jain, N. (2011). Scalable load balancing in cluster storage systems. In *Proceedings of the 12th acm/ifip/usenix international conference on middleware* (pp. 101–122). Berlin, Heidelberg: Springer-Verlag. Available from http://dx.doi.org/10.1007/978-3-642-25821-3_6

Yu, P. S., Dias, D. M., & Lavenberg, S. S. (1993, September). On the analytical modeling of database concurrency control. *J. ACM*, *40*(4), 831–872. Available from http://doi.acm.org/10.1145/153724.153733

Zhang, B., & Hsu, M. (1995). *Performance of concurrency control mechanisms in centralized database systems. (chapter "modeling performance impact of hot spots)"* (V. Kumar, Ed.). Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Zhang, Q., Cherkasova, L., & Smirni, E. (2007). A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of the fourth interna-*

tional conference on autonomic computing (pp. 27–). Washington, DC, USA: IEEE Computer Society. Available from `http://dx.doi.org/10.1109/ICAC.2007.1`

Zhu, B., Li, K., & Patterson, H. (2008). Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th usenix conference on file and storage technologies* (pp. 18:1–18:14). Berkeley, CA, USA: USENIX Association. Available from `http://dl.acm.org/citation.cfm?id=1364813.1364831`