Scientific Report for STSM project

Fabio Perfetti

{perfabio87@gmail.com}

March 2013

Reference Number	COST-STSM-IC1001-12576
Grant Number	COST-STSM-ECOST-STSM-IC1001-060113-026298
COST Action	IC1001
Action Description	ElasticBenchDTM: a benchmarking framework for elastic distributed transactional memory systems
STSM Period	2013-01-06 to 2013-03-15
Participant	Fabio Perfetti University of Rome, "La Sapienza"
Host	Prof. Paolo Romano Distributed System Groups, INESC-ID

1 Background

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 comes with the promise of enormous money saving and efficiency. On the other hand, it requires efforts to programmers, faced with the task of developing distributed applications tailored for dynamic, elastic, fault-prone environments.

Distributed Transactional Memory (DTM) appears to be an attractive paradigm to address this issue: in a DTM platform programmers can exploit the abstraction of transaction, delegating to a distributed middleware the burden of dealing with concurrency, fault-tolerance, transfers of state among nodes after the elastic rescaling of the system. DTM systems have garnered an increasing interest of late, also thanks to the emergence of cloud computing, and several research groups world-wide have started exploring the design of novel distributed consistency algorithms capable of ensuring transactional consistency semantics in a scalable and efficient way [1, 2, 3, 4, 5, 6, 7, 8]. On the other hand, a problem that has so far received less attention in the research community is related to the issue of how to support the dynamic resizing of a DTM platform in order to meet pre-determined QoS agreements while minimizing operational costs. This means, in a typical Cloud computing environment, determining the most cost-effective configuration of the platform, in terms of number of computational nodes and their type (e.g. large vs small instances in Amazon's EC2), which can satisfy the SLAs established between the application developers/users and the underlying IaaS/PaaS providers.

This is far from being a trivial problem, as the design of an automatic elastic scaling mechanism requires tackling a number of complex issues:

- Predicting the performance of a DTM-based application when deployed over a different set of nodes and/or nodes having different computational capabilities.

- Minimizing the overheads associated with the reconfiguration of the system, which, in the case of DTM platforms, can demand expensive state transfer phases aimed at ensuring the system's coherency in presence of nodes joining/leaving the platform.

- Identifying adequate trade-offs between reactivity and robustness of the controller in charge of determining the scale of the platform, in order to strike a balance between the efficiency of the system (which depends on the ability of the controller to timely respond to fluctu-

ations of the workload), and its stability (keeping into account that excessively reactive policies may lead the platform to thrash in case reconfigurations are triggered excessively frequently).

Recently, several solutions have been proposed that address a subset of the above mentioned problems. For instance, the work by Didona et al. [9] and Di Sanzo et al. [10, 11] have leveraged on analytical modelling and/or machine-learning methodologies to develop accurate predictors of the scalability of DTM applications. Several works have proposed to rely either on Virtual Machine (VM) migration supports [12] and/or on specialized solutions keeping into account the transactional nature of the platform [13, 14] to minimize state transfer costs in transactional systems. Finally, the issue of balancing responsiveness and stability when controlling the elastic scaling of replicated systems has been addressed in several works: Ali-Eldin et al. [15], for instance, apply classic control techniques to guide the elastic scaling of a web-farm, whereas Amza et al. [16] address this problem for the case of replicated database systems.

However, to the best of our knowledge, we are not aware of any rigorous approach that addresses the aforementioned problems in a holistic way for the increasingly relevant scenario of DTM platforms.

2 Purpose of the STSM

The purpose of this STSM has been to develop a benchmarking framework specifically designed to assess, in a unified testing environment, the effectiveness/efficiency of elastic scaling techniques for DTM platforms, and, in particular, their ability to tackle the problems discussed in the previous section. The resulting benchmarking framework, which we called ElasticDTMBench, offers the following key features:

- it provides a set of heterogeneous benchmarks for DTM platforms, inspired by popular benchmarks proposed both in the database and in the (non-distributed) transactional memory communities;
- it facilitates the portability of the benchmarking applications across different DTM platforms by abstracting over the implementation of the underlying DTM platform via a simple, generic interface of a transactional key/value store.
- it offers a number of time-varying load generation strategies, specifically designed to stress the elasticity of the DTM platform under test. These include both trace-based load injection mechanisms as well as loads specifiable via analytical functions (e.g. steps, ramps and periodic funcitons).

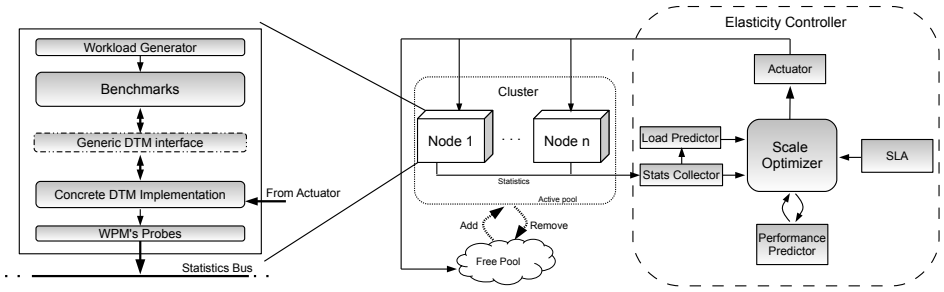

Figure 1: Overview of ElasticDTMBench's architecture.

- it allows to plug-and-play, in a seamless fashion, alternative load and performance prediction techniques. Also in this case, this is achieved via the definition of abstract interfaces aimed at encapsulating the heterogeneity of the actual implementation of these two important modules.

- it ensures portability across different IaaS platforms by means of an abstraction layer that mediates the interactions between the underlying IaaS provider and the controller in charge of automating the elastic scaling process.

ElasticDTMBench integrates and extends a number of open-source projects, and is also distributed as an open-source project that will be made shortly publicly available on the web. By adopting an open-source licensing scheme and a flexible, and easily extensible design, our hope is that ElasticDTMBench can become a reference framework for the researchers active in this challenging emerging area.

3 ElasticDTMBench Overview

This section is devoted to overviewing the architecture of ElasticDTMBench, which is illustrated by the diagram in Figure 1. In the following we describe the main modules composing ElasticDTMBench, discussing the key design choices that were taken while developing them and the rationale that motivated such decisions.

Benchmarking Applications: ElasticDTMBench includes the following 4 benchmarking applications:

- Warehouse: This benchmark is inspired to, and indeed represents an arguable close porting of, the TPC-C [17] benchmark. As in TPC-C, this benchmark emulates the activities of a wholesale supplier and includes three different transaction types: two conflict-prone update

transactions (inspired to the *payment* and *newOrder* transaction profiles of the TPC-C benchmark) having different access pattern characteristics, and a long-running read-only transaction (inspired to the *orderStatus* transaction profile of TPC-C).

- Vacation: This benchmark represents a porting of the homonymous benchmark of the STAMP [18] banchmark suite. The transactions generated by this benchmark mimic the activities of a travel agency, including the browsing and manipulation of a set of trees that keep track of customers and their reservations for various travel items. In addition to the original three types of (update) transactions (reservations, cancellations, and updates), this benchmark has been extended to include an additional read-only transaction profile simulating the search of travel items by end-users before the actual booking.

- WebSession: This benchmark emulates the transactional update of information associated with the session state maintained by an application/web server. These transactions are characterized by very small footprints (i.e., number of objects read/written per transaction) and a reduced contention probability.

- IntSet: This benchmark is inspired to the popular micro-benchmark suite adopted in the evaluation of several (non-distributed) TM systems [19] and consists of several implementations of a set interface (e.g., linked list and skip list). The benchmark is easily tunable to configure different degrees of contention and transactions accessing a small vs large number of objects.

Generic DTM Interface: In order to achieve portability across heterogenous DTM platforms, ElasticDTMBench relies on a simple, yet effective technique (analogous to the approach adopted, for instance, in Carvalho et al. [20]), which is based on the idea of hiding the implementation details of the underlying platform by means of an abstract, generic interface. This approach allows decoupling the benchmarks' implementations from the actual DTM platform they are being deployed on. In fact, the interaction between the benchmarks and the underlying DTM platform is mediated by means of an abstract interface that exposes standard methods for demarcating transactions and manipulating data. The interface assumes a simple key/value model for the representation of data, which, thanks to its simplicity, is expected to be easily matched by most of the existing DTM platforms. An important building block that ElasticDTMBench integrates and extends to this purpose is the open-source project RadarGun [21], developed by Red Hat. RadarGun was originally designed to benchmark heterogeneous key value stores and has been extended, in the scope of this STSM, to support

elastic rescaling of the platform during benchmarking, generation of time-varying workloads and on-line monitoring of the benchmarks' performance.

Workload Generation: given the focus of ElasticDTMBench toward the evaluation of the elasticity properties of DTM platforms, significant effort has been invested in the design and implementation of flexible workload generators capable of generating a wide range of dynamic load curves. More in detail, each node of ElasticDTMBench ships with a local workload generator that uses a pool of threads to inject load (i.e., transactions) in the benchmark. The local load-injectors are configurable to operate according to a number of alternative strategies. Specifically, the load injectors can emulate open/closed user populations [22] submitting requests at either fixed or time-varying arrival rates/think-times; further, the time-varying arrival rates/think-times can be defined using both analytical functions and traces from real systems (such as the traces gather for World Cup 98 [23], or recently made available by Google [24]). Given the distributed nature of the workload generation (i.e., each node in the system can locally inject transactions), ElasticDTMBench relies on a master/slave approach to globally regulate the injection of load in the platform. Specifically, ElasticDTMBench uses a dedicated process, called *master-load-injector*, which orchestrates the evolution of the workload generation according to a user-tunable *load-recipe*. The load-recipe is specified as a set of sequential stages, where each stage has a pre-determined duration and fully defines the workload type to be generated by each node of the platform during that time period. The master-load-injector provides the load-recipe to the set of nodes that are initially configured to run the benchmark, and is responsible for monitoring the global completion of a stage before triggering the advancement to the new one. In addition, it synchronizes the workload generation on any new node that dynamically joins the system, making sure that it is correctly aligned with the currently running stage.

Platform monitoring: ElasticDTMBench relies on the open-source project Workload and Performance Monitor (WPM), developed by the team of Prof. Francesco Quaglia of Rome University in the context of the EU project Cloud-TM [25]. WPM is designed to maximize scalability and efficiency of the monitoring infrastructure for a DTM platform. To this end, it relies on the Lattice bus, which allows for transparently using lightweight data dissemination overlays such as Pub-Sub architectures. Further, WPM is designed to minimize the bandwidth consumption in geographically distributed DTM infrastructures by means of, so-called, log-services, namely processes in charge of aggregating the data streams generated by the DTM nodes belonging to the same LAN/data-center, and to disseminate them via WAN-links using space-efficient encoding techniques. WPM interacts with the DTM platform using an extensible architecture based on the industry-

standard JMX [27] framework to extract statistical information from the local instance of the DTM node.

Elasticity controller: Finally, the Elasticity Controller (EC) is the module in charge of automatizing the elastic scaling of the platform. Since one of the key goals of ElasticDTMBench is to evaluate alternative EC algorithm, particular care has been taken in adopting a modular design for this component, which has, in fact, been broken down into a set of individual sub-components coupled via well-defined interfaces. Specifically, the EC is composed of the following sub-components:

- Load Predictor: this modules is fed with the stream of statistics gathered on-line from the platform and is meant to forecasting *future* workload volume and characteristics in order to allow the implementation of proactive provisioning policies. The current prototype fully specifies the interface of this component, but ships with a dummy implementation. The plan is to incorporate in ElasticDTMBench a simple predictor based on Kalman Filters [26], but this is still a work-in-progress.

- Performance Predictor: this module abstract over an oracle capable, given the current workload characteristics of the system, to predict the performance that the DTM will exhibit when deployed on a specific platform scale (i.e., number of nodes, number of threads active per node and capacity of the node). The current version of ElasticDTMBench integrates the performance predictor presented in Didona et al. [9] and, at the time of writing, work is in progress to integrate a second implementation of this module, which is based on the simulation model developed by Di Sanzo et al. [10].

- SLA specification: In order to define the QoS and cost constraints that should be enforced by the EC, ElasticDTMBench relies on the SLA definition and negotiation framweork developed in the context of the SLA@SOI European project [28]. This framework defines extensible templates encoded in XML, for which we have defined a specific instance allowing to express constraints on the following Key Performance Indicators for each type of transactional class generated by the benchmarking application: i) response time, ii) abort rate, and iii) throughput. Given the open and non-binding nature, of the SLA@SOI template, however, it is straightforward to extend them and incorporate additional types of KPIs or contraints.

- Scale Optimizer: Finally, the Scale Optimizer is the module in charge of determining the platform configuration to be used in order to match the specified SLAs and given the current/predicted workload. The current implementation integrates a simple, reactive policy, which selects

the scale of the platform that minimizes the operational cost (i.e., number of nodes) given the current load and a maximum tolerable abort rate.

- Actuator: As already mentioned in the previous section, ElasticDTMBench is designed to achieve portability across heterogeneous IaaS platforms. To this end, the actuator of the elastic scaling policies determined by the scale optimizer has been based on the open-source project δ-cloud [29], an abstraction layer which exposes a uniform API to automate provisioning of resources from IaaS providers (such as Amazon EC2, OpenNebula and RackSpace).

4 Future collaboration with host institution

As already discussed in the previous section, despite a preliminary prototype of ElasticDTMBench has already been developed during the STSM, there are still several aspects that need to be completed before the project can be publicly released. For this purpose, in the next months, both research teams have decided to continue the mutual collaboration.

The work done during this STSM will be integrated in my thesis project, whose completion is expected by May 2013. We also plan to prepare a joint paper aimed at advertising ElasticDTMBench to the community of researchers active in the area of (elastic) DTM systems.

5 Confirmation by the host institution of the successful execution of the STSM

The Host Paolo Romano from INESC-ID confirms that Fabio Perfetti achieved all the targets that we defined for this collaboration with distinction. ElasticDTMBench is a promising tool that, based on our experience with the development and evaluation of DTM platforms, may greatly facilitate the evaluation of future solutions in this area.

References

[1] T. Kobus, M. Kokocinski and P. T. Wojciechowski. Hybrid Replication: State-Machine-based and Deferred-Update Replication Schemes Combined. In Proc. of *International Conference on Distributed Computing Systems (ICDCS)*. July 2013.

[2] J. Kim, B. Ravindran. Scheduling Transactions in Replicated Distributed Transactional Memory. In Proc. of *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 2013.

[3] M. Herlihy, Y. Sun. Distributed transactional memory for metric-space networks. In Proc. of *International Symposium on Distributed Computing (DISC)*. 2005.

[4] H. Attiya, V. Gramoli and A. Milani. A Provably Starvation-Free Distributed Directory Protocol. In Proc. of *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 2010.

[5] A. Dash and B. Demsky. Automatically Generating Symbolic Prefetches for Distributed Transactional Memories. In Proc. of *International Middleware Conference (Middleware)*. 2010.

[6] C. Kotselidis, M. Ansari, K. Jarvis, M. Luján, C. Kirkham and I. Watson. DiSTM: A Software Transactional Memory Framework for Clusters. In Proc. of *International Conference on Parallel Processing (ICPP)*. 2008.

[7] P. Di Sanzo, D. Rughetti, B. Ciciani and F.Quaglia. Auto-tuning of Cloud-based In-memory Transactional Data Grids via Machine Learning. In Proc. of *International Symposium on Network Cloud Computing and Applications (NCCA)*. 2012.

[8] M. Couceiro, P. Romano and L. Rodrigues. PolyCert: Polymorphic Self-Optimizing Replication for In-Memory Transactional Grids. In Proc. of *International Middleware Conference (Middleware)*. 2011.

[9] D. Didona, P. Romano, S. Peluso and F. Quaglia. Transactional auto scaler: Elastic scaling of in-memory transactional data grids. In Proc. of *Ninth International Conference on Autonomic Computing (ICAC)*. 2012.

[10] P. Di Sanzo, F. Antonacci, B. Ciciani, R. Palmieri, A. Pellegrini, S. Peluso, F. Quaglia, D. Rughetti and R. Vitali. A Framework for High Performance Simulation of Transactional Data Grid Platforms. In Proc. of *International Conference on Simulation Tools and Techniques (SIMUTools)*. 2013.

[11] D. Rughetti, P. Di Sanzo, B. Ciciani and F. Quaglia. Machine Learning-based Self-adjusting Concurrency in Software Transactional Memory Systems. In Proc. of *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2012.

[12] E. Cecchet, R. Singh, U. Sharma, and P. J. Shenoy. Dolly: virtualization-driven database provisioning for the cloud. In Proc. of *International Conference on Virtual Execution Environments (VEE)*. 2011.

[13] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In Proc. of *International Conference on Management of data (COMAD)*. 2011.

[14] S. Das, S. Nishimura, D. Agrawal, and A. E. Abbadi. Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration. In Proc. of *Very Large Data Bases Conference (VLDB)*. 2011.

[15] Ahmed Ali-Eldin, J. Tordsson, E. Elmroth. An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures. In Proc. of *Network Operations and Management Symposium (NOMS)*. 2012.

[16] J. Chen, G. Soundararajan, C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. In Proc. of *International Conference on Autonomic Computing (ICAC)*. 2006.

[17] TPC-C on-line transaction processing benchmark, *http://www.tpc.org/tpcc/default.asp*

[18] C. C. Minh, JaeWoong Chung, C. Kozyrakis, K. Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. In Proc. of *International Symposium on Workload Characterization (IISWC)*. 2008.

[19] M. Herlihy , V. Luchangco , M. Moir. A flexible framework for implementing software transactional memory. In Proc. of *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 2006.

[20] N. Carvalho, P. Romano and L. Rodrigues. A Generic Framework for Replicated Software Transactional Memories (short paper). In Proc. of *International Symposium on Network Computing and Applications (NCA)*. 2011.

[21] Radargun benchmark framework, https://github.com/radargun/radargun/wiki

[22] D. A. Menascé, V. A. F. Almeida: Capacity Planning for Web Services: metrics, models and methods. *Prentice Hall, PTR*

[23] A. Martin, J. Tai. Workload Characterization of the 1998 World Cup Web Site. In *Tech. Rep. HPL-1999-35R1, HP Labs*. 1999.

[24] Google Cluster Data, *http://googleresearch.blogspot.com/2010/01/google-cluster-data.html*

[25] R. Palmieri, P. Di Sanzo, F. Quaglia, P. Romano, S. Peluso and D. Didona. Integrated Monitoring of Infrastructures and Applications. In Proc. of *Cloud Environments Cloud Computing Project and Initiatives (CCPI)*. 2011.

[26] G. Welch and G. Bishop. An introduction to the Kalman filter. In *Tech. Rep. 95-041 University of North Carolina at Chapel Hill, Department of Computer Science*. 1995.

[27] Sun Microsystems: The Java Tutorials – Java Management Extensions (JMX). *http://java.sun.com/docs/books/tutorial/jmx/index.html*, 2008.

[28] SLA@SOI EU project. *http://sla-at-soi.eu/*

[29] δ-Cloud, open source project. *http://deltacloud.apache.org/*