Cloud-TM: Scalable, Self-tuning, Transactional Cloud Data Store

Paolo Romano
Cloud-TM at a glance

Partners:
- INESC ID (PT)
- Algorithmica (IT)
- C.I.N.I. (IT)
- Red Hat (IE)

Project coordinator:
Paolo Romano, INESC ID (PT)

Duration:
From June 2010 to May 2013

Programme:
FP7-ICT-2009-5 – Objective 1.2

Further information:
http://www.cloudtm.eu
Cloud: the bright side...

- Unprecedented scalability levels
- Minimize operating costs & carbon footprint via elastic provisioning
- Leverage economies of scale for both services providers and users
- Lower barriers to entry via usage-based pricing schemes
...and the dark side!

• Lack of programming models effectively hiding the issues of:
  – concurrency
  – distribution
  – fault-tolerance
  – elasticity

• Need for mechanisms to ensure efficiency at any scale and for any workload:
  – no-one-size-fits-all solution
  – manual tuning is costly, error prone and suboptimal
  – how to ensure QoS in highly dynamic environment?
Main project motivation
Key project goals

Develop a data-centric PaaS aimed to minimize:

1. developments costs:
   ➞ introducing abstractions aimed to hide complexity

2. administration costs:
   ➞ aiding/replacing sys admins via self-tuning

3. operational costs:
   ➞ maximizing efficiency via self-tuning
The Cloud-TM Solution

...but first some background...
From Transactional Memory...

- Transactional Memory (TM):
  - replace locks with atomic transactions **in the programming language**
  - hide away synchronization issues from the programmer
    - avoid deadlocks, priority inversions, debugging nightmare
    - simpler to reason about, verify, compose
  - simplify development of parallel applications
• Distributed Transactional Memory (DTM):
  – extends TM abstraction over the boundaries of a single machine:
    • enhance scalability
    • ensure fault-tolerance
  – maximize scalability and efficiency via:
    • efficient data replication protocols
    • speculation and batching of consistency actions
...to the Cloud-TM platform!
Cloud-TM Platform

Data Platform

Programming APIs
- Object Grid Mapper
- Search API
- Distributed Execution Framework

In-memory Distributed Transactional Key-Value Store
- Reconfigurable Transactional Memory
- Reconfigurable Replication Manager
- Interface to Storage Systems

Autonomic Manager

QoS/cost specification

WORKLOAD ANALYZER

WORKLOAD MONITOR

ADAPTATION MANAGER

Persistent Storage Systems
- S3
- Cassandra

IaaS Providers
- openstack
- Amazon Web Services
- Nimbus
Cloud-TM Data Platform

Data Platform Programming APIs

- Object Grid Mapper
- Search API
- Distributed Execution Framework

In-memory Distributed Transactional Key-Value Store

- Reconfigurable Transactional Memory
- Reconfigurable Replication Manager
- Interface to Storage Systems

HIBERNATE OGM & Search

Infinispan JGroups
Autonomic Manager

QoS/cost specification

WORKLOAD & QoS MONITOR
WORKLOAD ANALYZER
ADAPTATION MANAGER

from Data Platform
to Data Platform

IaaS Providers
The Cloud-TM approach

- Innovation in three main areas:
Programming Model

- Strong Transactional Consistency
- Object Orientation
- Data querying
- Programming Abstractions
Programming model

Strong transactional consistency

• Transactional manipulation of *in-memory* objects:
  – atomicity and isolation guarantees
  – primary mechanism for durability ➔ replication

• Goal:
  – *shelter programmers from complexity* of weak consistency models
• Full support for object-oriented data model:
  – transparent mapping of OO model to Key-Value model

• Integration with the Java ecosystem:
  – JAVA Persistence API (JPA)
    • the STANDARD way of persisting JAVA objects (Hibernate OGM)
  – Fenix Framework
    • higher level abstraction API, allows for more agile experimentation

• ...as well as with Ruby!
Programming model
Data Querying

• Support for querying object-oriented domain:
  – automatic indexing of the data maintained by the platform
  – subset of industry standard JP-QL interface:
    • exact/approximate values queries
    • polymorphic queries
    • by range, aggregation functions, by association
Programming model

Programming Abstractions

- minimizing data contention
- maximizing data locality
- scheduling execution of parallel tasks

simpler and better performing cloud applications
Programming abstractions
Minimize data contention

• Let expert programmers exploit appl. semantics
• Avoid aborting txs upon “benign” conflicts
• Killer application: collections

---

T1: insert 9

T2: remove 20

---

T1: ghost read
T1: register

T2: ghost read
T2: register

---

T1: ghost read
head

T2: ghost read
7

T2: ghost read
12

T2: ghost read
20
• Locality hints (LH):
  – let programmers specify which objects’ attributes should be used to determine their placement
  – LH define a multi-dimensional hyperspace
  – Objects with common LHs get co-located

• Object ➔ Point in LH space ➔ Platform Node
  – programmer defined
  – automatized by the platform
Programming abstractions
Maximize data locality

- Object ➔ Point in LH space ➔ Platform Node
  
  programmer defined

Class Person {
    @localityHint
    Country nation;
    String Name, Surname;
}

Class Car{
    @localityHint
    Country nation;
    String Model, Descrip;
}
Programming abstractions
Maximize data locality

• Object ➞ Point in LH space ➞ Platform Node

programmer defined

automatized by the platform

1. based on consistent hashing (random)
2. based on nodes’ access patterns
   (AutoPlacer [ICAC13])
Delayed Actions \cite{SRDS13}:
- postpone data manipulations till commit time

Distributed Execution Framework \cite{INESC13}:
- exploit locality to route transactions to the nodes where the data is stored

Transaction Migration \cite{INESC13}:
- allow a tx running on node n to execute arbitrary code on a node n’ and resume execution on n
…to the Cloud-TM approach

- Innovation in three main areas:
  - Programming model
  - Distributed Data Management
  - Self-optimization
Cloud-TM Platform

Data Platform

- Programming APIs
  - Object Grid Mapper
  - Search API
  - Distributed Execution Framework

- In-memory Distributed Transactional Key-Value Store
  - Reconfigurable Transactional Memory
  - Reconfigurable Replication Manager
  - Interface to Storage Systems

- Persistent Storage Systems
  - S3
  - Cassandra

Autonomic Manager

- QoS/cost specification
- WORKLOAD & QoS MONITOR
- WORKLOAD ANALYZER
- ADAPTATION MANAGER

- IaaS Providers
  - openstack
  - amazon web services
  - NIMBUS

26
Infinispan

- Cloud-TM reference DTM platform:
  - open source project by JBoss/Red Hat
  - in-memory transactional key-value store
Some users of Infinispan

Telecom & Media
- RIM
- comcast
- Telefónica
- NOKIA
- DreamWorks

Financial Services & Insurance
- CBOE
- Putnam Investments
- citi
- Lombard Odier
- Bank of America
- realestate.com.au
- USAA
- State Farm Insurance

Retail
- eBay
- Best Buy

Travel
- Emirates

Govt.
- NASA

Energy
- GE Energy

Thursday, 16 June 2011
Distributed Data Management

Highly scalable transactional consistency protocols

Non-blocking platform’s reconfiguration

Polymorphic replication
Distributed Data Management

Highly scalable transactional consistency protocols

Non-blocking platform’s reconfiguration

Polymorphic replication
Highly scalable replication

• Genuine partial replication schemes:
  – #copies of data items << #nodes in the system
  – involve in the transaction processing only nodes maintaining copies of accessed data:
    • no centralized components/no global broadcasts

• Explored various tradeoffs in the consistency spectrum:
  – Weak consistency: TOM [PRDC11]
  – Strong consistency: GMU & extensions [ICDCS12, Middleware12, SRDS13...]

Workshop on Storage & cloud computing, 21/11/2013, Rennes, France
• Consistency is often sacrificed in cloud data stores to maximize scalability, e.g.:
  – eventual consistency:
    • a posteriori application driven reconciliation
  – read committed/repeatable read isolation levels:
    • applications can observe non-serializable snapshots

• Scalability comes at the cost of complexity
Scalability and consistency?

Are there any sweet spots in the scalability vs consistency trade-off?

• Do we need to give up consistency to achieve scalability?
• **Genuine**
  – fully decentralized/no centralized coordinator

• **Multiversion**
  – read-only txs never aborted nor validated

• *(Extended) Update Serializable:*
  – 1-Copy Serializability for update tx
  – Read-only txs:
    • must observe *some* serializable snapshot
    • can witness non-conflicting update txs in diff. orders
  – compliant with ANSI SQL SERIALIZABLE
Vacation

![Graph showing throughput and aborted transactions for different workloads](image)

- **GMU+GR+DA tx/s**
- **GMU tx/s**
- **GMU+GR+DA abort%**
- **GMU tx/s**

**Throughput (txs/s)**

**Aborted transactions (%)**

*Workshop on Storage & cloud computing, 21/11/2013, Rennes, France*
Figure 61. Evaluating the benefits of Ghost Reads using SkipList

Concurrent removal of elements that are navigated to find the target element do not affect the correctness of the navigation with GMU and actually with any other concurrency control that guarantees that the observed snapshots are legal according to some equivalent sequential schedule [9].

All the experiments have been performed on top of the Cloud™ Cluster using the same virtualization configuration described in Section 6.4, which allows for deploys on top of up to 493 single-core virtual machines, each equipped with 7GB of RAM.

Let us start by analyzing the results with Skiplist, which are reported in Figure 61. The results show an average throughput gain when using ghost reads of 51% across the different scales of the system, with a peak gain of 5156 speedup at 493 nodes. We will consistently see across the experiments, these gains are due to a considerable reduction of aborts. In this case, we report an average reduction of likelihood of update transactions aborts from 48% to 31%.

Next, we show in Figure 61.2 the results when using YCSB. Once again, we can see considerable gains due to the avoidance of many conflicts by exploiting the Ghost Reads mechanism. The plots highlight an average speedup of 51% with the version using Ghost Reads reaching a peak throughput of almost 46k txs/s against 71k txs/s for GMU at 493 nodes.

In the plots, we report also the results using the variant of GMU, in which the prepare messages are delivered via a genuine Total Order multicast (TO). This variant of GMU is particularly effective in conflict-prone scenarios by achieving deadlock avoidance [0]. We present this experiment in Figure 61.3, where we show not only the version using GMU and Ghost Reads, and the version using GMU and regular reads, but also the augmented variants that use TO. On average, the TO variants improve 83% over their counterparts that can be affected by deadlocks and thus spuriously abort due to timeouts. Still, we can see that usage of Ghost over GMU and TO also improves on average 510% times over GMU and TO with regular reads.

Overall, these results highlight that the proposed Ghost Read mechanism is
Distributed Data Management

- Highly scalable transactional consistency protocols
- Non-blocking platform’s reconfiguration
- Polymorphic replication
The search for the holy grail

transactional data

consistency protocols

Single master (primary-backup)

Multi master

Total order-based

2PC-based

Certificate

State machine replication

Non-voting

Voting

BFC

NO ONE SIZE FITS ALL SOLUTION

Workshop on Storage & cloud computing, 21/11/2013, Rennes, France
No one size fits all

- Existing solutions are optimized for specific workload/scale scenarios

- In **dynamic** environments where both:
  1. the workload characteristics, and
  2. the amount of used resources

vary over time, **self-tuning is the only way to achieve optimal efficiency**
The Cloud-TM approach

- **Low resources:**
  - Minimum costs
  - Primary-backup:
    - Low % write: low load on primary

- **Auto-scale up:**
  - New nodes hired for read-only requests
  - Primary-backup:
    - Low % write: primary stands the load

- **Multi-master:**
  - Hi % write: primary overwhelmed
  - Higher scalability

- **Auto-scale down:**
  - Minimum costs
  - Switch back to primary-backup

---

Legend:
- Node processing read-only requests
- Node processing read-write requests

- Low traffic
- Read-dominated
- Low conflict

- Hi traffic
- Read-dominated
- Low conflict

- Hi % write

Low % write transactions

Low traffic

High

Low

Time

Workshop on Storage & cloud computing, 21/11/2013, Rennes, France
MorphR [Middleware11, DSN13]

• Generic framework supporting **dynamic switching** between **arbitrary replication protocols**

• Protocol switching phases encapsulated in abstract FSMs with neatly defined interfaces

• Problem of determining the right replication protocol is delegated to the Cloud-TM Autonomic Manager
Key idea

• Support for:
  – blocking (stop&go) reconfigurations:
    + generic
    - less efficient
  – non-blocking reconfigurations:
    + efficient
    - specific for each pair of protocol
MORPHR in action

Figure 3.19: Dynamic behavior of MorphR self-tuning framework (GeoGraph - CloudTM Cluster)
Distributed Data Management

- Highly scalable transactional consistency protocols
- Non-blocking platform’s reconfiguration
- Polymorphic replication
Non-blocking reconfigurations

• Adding/removing nodes while transactions are in progress is not a trivial issue:
  – data may have to be migrated to different nodes
  • a process that may last even several minutes
  • this include also metadata like locks, VCs, etc…

• while state transfer is in progress we want to minimize impact on performance:
  • ongoing transactions should not be aborted
  • new transactions should progress w/o impairment

...of course transactional consistency need to be guaranteed whatsoever!
Non-blocking state transfer
...to the Cloud-TM approach

• Innovation in three main areas:
Cloud-TM Platform

Data Platform

Programming APIs

Object Grid Mapper | Search API | Distributed Execution Framework

In-memory Distributed Transactional Key-Value Store

Reconfigurable Transactional Memory
Reconfigurable Replication Manager
Interface to Storage Systems

Autonomic Manager

QoS/cost specification

WORKLOAD & QoS MONITOR
WORKLOAD ANALYZER
ADAPTATION MANAGER

In-Cloud

Persistent Storage Systems

IaaS Providers

S3
Cassandra

openstack
amazon web services
NIMBUS
Self-optimization

- Pervasive workload and performance monitoring
- Innovative performance forecasting methodologies
- Multi-objective optimization
- QoS/cost driven self-optimization
Self-optimization

- Pervasive workload and performance monitoring – workload characterization across all platform’s layers
Self-optimization

- Pervasive workload and performance monitoring—lightweight algorithms from stream analysis literature to pinpoint hot spots for data locality and contention

<table>
<thead>
<tr>
<th>Top-K Abort Inducing Keys</th>
<th>Key</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance_101</td>
<td>12345</td>
<td></td>
</tr>
<tr>
<td>Balance_202</td>
<td>654</td>
<td></td>
</tr>
<tr>
<td>Stock_Item_234</td>
<td>543</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Top-K Remotely Acc. Keys</th>
<th>Key</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inventory_132</td>
<td>45321</td>
<td></td>
</tr>
<tr>
<td>Agent_432</td>
<td>12302</td>
<td></td>
</tr>
<tr>
<td>Stock_Item_234</td>
<td>9000</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Top-K Locally Acc. Keys</th>
<th>Key</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance_101</td>
<td>12345</td>
<td></td>
</tr>
<tr>
<td>Balance_202</td>
<td>654</td>
<td></td>
</tr>
<tr>
<td>Stock_Item_234</td>
<td>543</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Self-optimization

- innovative performance forecasting methodologies
  - novel methodologies for the performance modeling of distributed transactional platforms
Chapter 3

Autonomic Manager

The diagram of Figure 3.1 provides a high-level overview of the architecture of the Autonomic Manager (AM), and reports the set of self-tuning mechanisms that it supports.

As illustrated in Figure 3.1, the self-optimization mechanisms supported by the Cloud-TM Autonomic Manager can be classified in two main types:

1. Solutions aimed at identifying the optimal values of a set of key configuration parameters/mechanisms of the Cloud-TM Data Platform, namely the:
   - Scale of the underlying platform, i.e., the number and type of nodes over which the Cloud-TM Data Platform is deployed;
   - Number of replicas of each datum stored in the platform, which we also call replication degree;

2. Multi-objective optimization

   QoS & Cost constraints

   - Performance Forecasting Tools
   - Locality Enhancing Mechanisms

   ADAPTATION MANAGER

   - Scale Optimizer
   - Replication Degree Optimizer
   - Replication Protocol Optimizer
   - Locality-aware Request Dispatching
   - Locality-aware Data Placement

   Automatic re-configuration of the Cloud-TM Data Platform

   Workload & Performance Data From Workload Analyzer

   Workload & Performance Data From Workload Analyzer

Workshop on Storage & Cloud Computing, 21/11/2013, Rennes, France
Self-optimization

• QoS/cost driven self-optimization

Application developer
- specify SLAs
- upload application & define workload (real or synthetic)

QoS negotiator
- if tx conflict prob.<10% and CPU time per tx<500msec
- update SLA & workload characteristics

Cloud-TM platform
- deploy application, profile it & predict scalability trends
- if cost & throughput

Submit SLA Request
- Throughput *
- Response time *
- Maximum Admissible Abort Rate *
- Observation Period *

The diagram illustrates the process of self-optimization, where the application developer specifies SLAs, uploads the application and defines the workload, and the QoS negotiator optimizes based on the application's performance metrics. The Cloud-TM platform then deploys the application, profiles it, and predicts scalability trends to update the SLA and workload characteristics.
The open source way

• Research results integrated in highly visible Red Hat projects:

  Infinispan
  HIBERNATE Search
  HIBERNATE OGM
  RHQ  JGroups

Cloud-TM Platform

Data Platform
- Data Platform Programming APIs
  - Object Grid Mapper
  - Search API
  - Distributed Execution Framework
  - HIBERNATE OGM & Search

In-memory Distributed Transactional Key-Value Store
- Reconfigurable Transactional Memory
- Reconfigurable Replication Manager
- Interface to Storage Systems

Autonomic Manager
- QoS/cost specification
- WORKLOAD & QoS MONITOR
- WORKLOAD ANALYZER
- ADAPTATION MANAGER
- RHQ

S3  Cassandra
Persistent Storage Systems

IaaS Providers

Workshop on Storage & cloud computing, 21/11/2013, Rennes, France
Conclusions

• Distributed programs are hard to develop
  – even harder in dynamic, large-scale clouds

• API should hide complexity, whenever possible
  – not limit expressiveness, whenever needed

• No-one-size-fits-all solution
  – elastic computing urges for self-tuning solutions
The Cloud-TM Approach

- Strong transactional consistency
- Object-Oriented programming & query support
- QoS-oriented resource provisioning
- Scalability & efficiency achieved thanks to:
  - innovative data consistency schemes
  - abstractions to maximize locality & reduce conflicts
  - pervasive, multi-objective self-optimization
Do not miss the Cloud-TM website:

http://www.cloudtm.eu

You will find:
ready-to-go VM images, source code, demos, tutorials, docs...
References


Full list of relevant publications available here:
http://www.cloudtm.eu/home/Publications
Self-optimization

- Pervasive workload and performance monitoring
  - state of the art workload forecasting algorithms
Self-optimization

- Pervasive workload and performance monitoring
  – novel metrics to characterize application scalability

Fig. 4: ACF using heterogeneous benchmarks and platforms.
Geograph – static configuration

Figure 3.18: Comparison of the performance of static protocol configurations (GeoGraph - Cloud-TM Cluster)

Workshop on Storage & cloud computing, 21/11/2013, Rennes, France
Data distribution, replication, persistence

- Interoperability with diverse persistent storages
The problem of data locality

• In partially replicated system, transactions may have to fetch remote data during exec.

• If this happens "too often": the network will eventually saturate — transactions will become longer and more vulnerable to conflicts

...shortly: performance will be poor

Poor Data Placement Can Cripple Performance!
AutoPlacer

• Self-tuning system for adapting data placement in a distributed key/value store

• Key challenges:
  – which data should be moved?
    • big data ➔ large monitoring overheads
  – where to move the data
    • distributed optimization problem
  – how to encode and maintain efficiently the mapping?
    • big data ➔ large directory Key ➔ Node(s)
Hot Spots Detection

• Online algorithm, round based:
  – fully distributed hot-spot detection
    • most frequently accessed remote keys (per node)
  – lightweight probabilistic top-k algorithm
    • sub-linear space (stream analysis)
    • bounded error
  – in each round, nodes send access frequencies of their hot spots to their node owners
AutoPlacer

• Self-tuning system for adapting data placement in a distributed key/value store

• Key challenges:
  – which data should be moved?
    • big data $\rightarrow$ large monitoring overheads
    – where to move the data
      • distributed optimization problem
  – how to encode and maintain efficiently the mapping?
    • big data $\rightarrow$ large directory Key $\rightarrow$ Node(s)
Who gets the data?

• Distributed optimization formulated as an Integer Linear Programming (ILP) problem

$$\min \sum_{j \in \mathcal{N}} \sum_{i \in \mathcal{O}} X_{ij}(cr^r r_{ij} + cr^w w_{ij}) + X_{ij}(cl^r r_{ij} + cl^w w_{ij})$$

subject to:

$$\forall i \in \mathcal{O} : \sum_{j \in \mathcal{N}} X_{ij} = d \land \forall j \in \mathcal{N} : \sum_{i \in \mathcal{O}} X_{ij} \leq S_j$$

• Place d copies of each object so to:
  – Minimize global number of remote accesses
  – Ensure storage capacity constraints

Based on approx. stats from probabilistic Top-K
Relaxed to LP problem and computed in parallel as independent subproblems
AutoPlacer

• Self-tuning system for adapting data placement in a distributed key/value store

• Key challenges:
  – which data should be moved?
    • big data → large monitoring overheads
  – where to move the data
    • distributed optimization problem
  – how to encode and maintain efficiently the mapping?
    • big data → large directory Key → Node(s)
How to store where data is?

State of the art solutions

- directory based:
  PRO: maximum flexibility
  CON: non-scalable, large overhead (remote lookup)

- random hashing:
  PRO: lightweight and scalable
  CON: no control on data placement

encode <data/node> association via decision tree classifiers:

- very compact and highly efficient
- small possibility of misclassifications / misplacement
Task 3: The above information is used in the third task identified as the optimization task to find an appropriate placement for those items. The result of this task is a partial relocation map, i.e., a mapping of where replicas of each hotspot item, which is supervised by the node, must be placed for the current round.

Task 4: Even if the number of hotspots tracked at each round is a small fraction of the entire set of items maintained in the key-value store, over multiple rounds the relocation map can grow in an undesired way and may even be too large to be efficiently distributed to all nodes. This task is devoted to encoding the relocation map in a probabilistic data structure that can be efficiently replicated on all nodes in order to ensure fast lookups, i.e., a Probabilistic Associative Array (PAA). Specifically, each node computes the PAA for the relocated objects it supervises.

Task 5: Once each PAA has been computed, each node disseminates it among all nodes. By assembling the PAAs received from all the nodes in the system, each node can locally build an object lookup table that includes updated information on the placement of data optimized during this round.

Task 6: Finally, at the end of each round, the data items for which new locations have been derived are transferred in order to match the new data placement. As can be inferred from the previous description, the work is divided among all nodes and communication takes place only during tasks 2, 5, and 6, respectively, to exchange statistical information on hotspots, distribute the PAA, and finally relocate the objects. Also, the tasks that require communication are performed in parallel, without the help of any centralized component.

In the next subsections, we provide more information about the two main components of AUTOPLACER, namely, the optimizer (executed by Task 3) and the PAA (built in Task 4 and used subsequently to perform data lookups locally).
TPC-C

Workshop on Storage & cloud computing, 21/11/2013, Rennes, France
Figure 3.27: Evolution of throughput and read-only transaction’s response time using AUTOPLACER (GeoGraph - FutureGrid Cluster)