A Framework for High Performance Simulation of Transactional Data Grid Platforms

Pierangelo Di Sanzo, Francesco Antonacci, Bruno Ciciani, Roberto Palmieri, Alessandro Pellegrini, Sebastiano Peluso, Francesco Quaglia, Diego Rughetti, Roberto Vitali

High Performance and Dependable Computing Systems Group
Sapienza, University of Rome

SIMUTools 2013

Target: In-Memory Data Platforms

- In the last few years a new generation of in-memory transactional data platforms (NoSQL data grids) has proliferated
  - VMware vFabric GemFire
  - Oracle Coherence
  - Red Hat's Infinispan
  - Apache Cassandra
- They well meet elasticity requirements imposed by the pay-per-use cost model of cloud computing:
  - Rely on a simplified key-value data model
  - Employ efficient in-memory replication mechanisms to achieve data durability
  - Natively offer facilities for dynamically resizing the amount of hosts within the platform

All that glitters isn't gold!

Deploying a distributed transaction key-value store platform poses a number of performance/reliability/availability issues:

- How many nodes in the platform?
- Which concurrency control algorithm?
- How many replicas of data?
- Which data placement scheme?

and on top of that:

- Given a platform setting, does it also well fit in different scenarios (e.g. when the workload changes)?

All that glitters isn't gold!

Deploying a distributed transaction key-value store platform poses a number of performance/reliability/availability issues:

- How many nodes in the platform?
- Which concurrency control algorithm?
- How many replicas of data?
- Which data placement scheme?

and on top of that:

- Given a platform setting, does it also well fit in different scenarios (e.g. when the workload changes)?

Experience suggests that, e.g., an oversized platform (too many nodes) causes a performance drop (and is more expensive) ☹☹☹

And what about dynamic reconfiguration?

Traditional solutions to dimensioning entail:

- Analytical models
- Machine learning
- Petri nets

And what about dynamic reconfiguration?

Traditional solutions to dimensioning entail:

- Analytical models
- Machine learning
- Petri nets

The performance can exhibit a strong non-linear behavior when the number of nodes grows

And what about dynamic reconfiguration?

Traditional solutions to dimensioning entail:

- Analytical models
- Machine learning
- Petri nets

The performance can exhibit a strong non-linear behavior when the number of nodes grows

Timely what-if analysis could enable for runtime reconfiguration

Goals

- We propose a solution based on high-performance simulation
- A discrete-event simulation library allows easy development of data grid models to support what-if analysis when varying:
  - Number of cache servers
  - Degree of replication of data objects
  - Placement of data-copies across the platform
- The library natively supports:
  - 2PC
  - Repeatable read semantics (based on lazy locking)
  - Primary data ownership
  - Multi-master schemes
- Implementing new strategies is an easy task for the modeler
- The library is run on top of ROOT-Sim

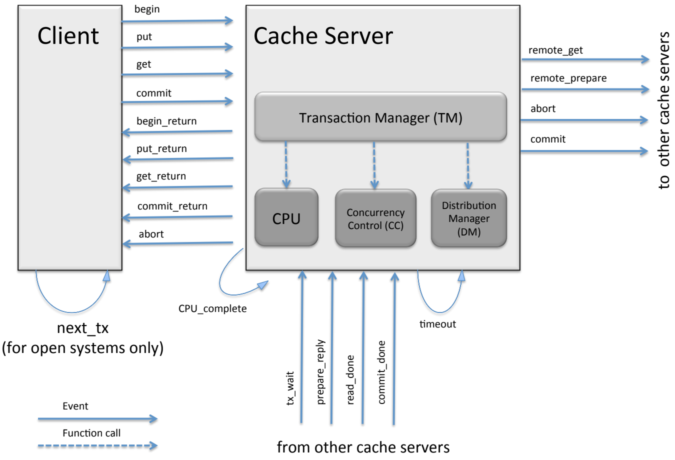
CloudTM

This project has been developed in the context of CloudTM FP-7 Project

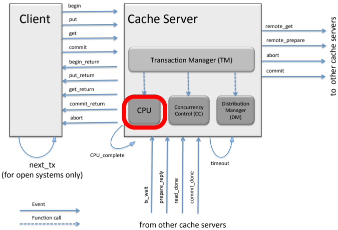
http://www.cloudtm.eu

Goal: Self-tuning of In-Memory Data Grids

Simulation Framework

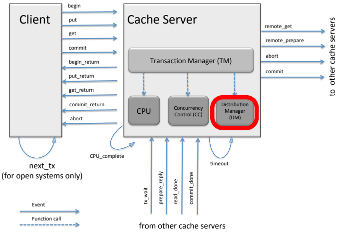


CPU



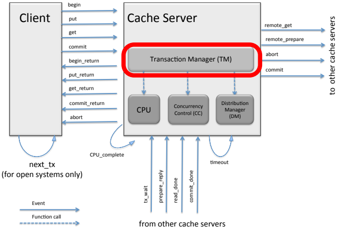
- Modeled as a G/M/K queue
- Allows capturing scenarios with multiple cores
- Expected to be adequate wrt more complex models, because core dynamics are associated with logical contention
- Different cpu models can be easily integrated

Distribution Manager



- Keeps track of the of data placement on the nodes of system
- Tells TM where to direct requests for reading/writing
- Notifies TM which is the primary owner of a copy of the data object to be accessed

Transaction Manager



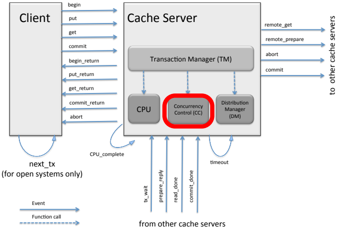
- Acts as a frontend for event processing
- Interacts with the CPU module to compute completion time and update CPU load
- Several events are sent to TM, and trigger specific actions.

Transaction Manager (2)

Transaction Manager processes these events from clients:

- `begin`: Used to nofity that a new transactional interaction has been issued by some client
- `get`: Used to nofity that a read operation on some data object has been issued by the client within a transaction
- `put`: Used to notify that a write operation on some data object has been issued by the client within a transaction
- `commit`: Used to indicate that the client ended issuing transactional operations

Concurrency Control



- Invoked by the TM front end
- Depending on the rules of the concurrency algorithm, CC can reply:
  - *continue*: the transaction's execution can proceed
  - *abort*: the transaction must be aborted
  - *wait*: the transaction is temporarily blocked
- The simulation modeler can easily implement other concurrency control algorithms

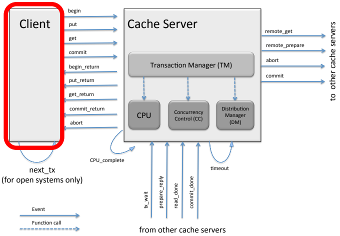
Concurrency Control: An Example

```
record TxInfo {
    TxId
} //end record

CC-logic(input: task T, pointer CC-Table) {
if (CC-table == NULL)
    allocate and initialize [wait-for,active-tx] table;
    // keys are data object identifiers or TxId values
    // entries are lists of TxInfo records or TxId values
    set CC-table point to the allocated table
case T.type
    prepare:
       link T.TxInfo.TxId to CC-Table.active-tx
       AllPrepareKeys = T.TxWriteSet
       link T.TxInfo to CC-Table.wait-for[AllPrepareKeys]
```

Concurrency Control: An Example (2)

```
        if T.TxInfo not top standing for some key
           generate event TX_WAIT[T.TxInfo]
           generate event TIMEOUT[T.TxInfo]
        else generate event PREPARE_DONE[T.TxInfo]
    timeout or commit:
        unlink T.TxInfo.TxId from CC-Table.active-tx
        unlink T.TxInfo from CC-Table[AllOccurrences]
        if (T.type == commit) generate COMMIT_DONE[T.TxInfo]
        else generate PREPARE_FAIL[T.TxInfo]
        for all TxInfo top standing in CC-Table[AnyPresenceRow]
            generate event PREPARE_DONE[TxInfo]

return CC-Table
} //end CC
```

Client

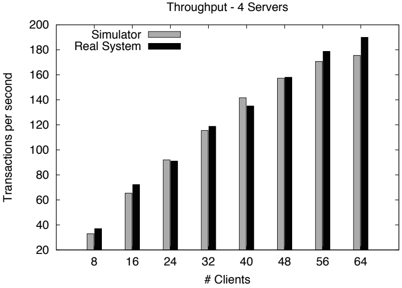


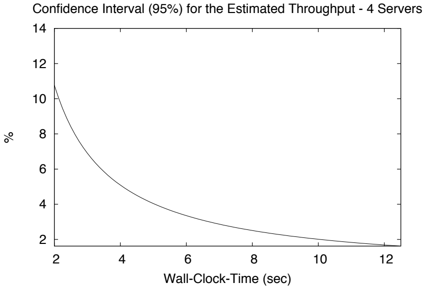
The modeler can specify various settings:

- The system model (open vs closed)
- Number of concurrent clients, and threads per client
- Transaction generation rate/trace
- A number of different transaction profiles, and for each one:
  - Number of transactions to be executed
  - type (put vs. get) and operations per transaction
  - data access distribution
  - inter-operation think time
- transaction back-off time (when aborted)

Validation

TPC-C on RedHat Infinispan, deployed on Amazon EC2.



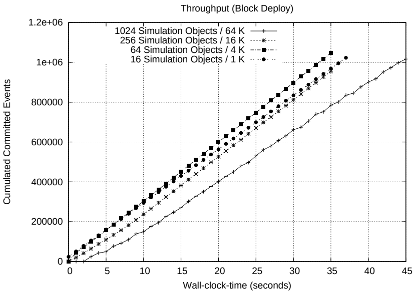


Enabling for timely what-if analysis

- 12 seconds to predict the behaviour of a system is too much
- The framework has been deployed on top of ROOT-Sim

- Up to 1024 simulation objects, 1/8 being cache servers.
- Iso-scaling in terms of both model complexity and underlying computing power
- Run on a couple of HP Proliant servers:
  - 64-bits NUMA machines
  - four 2GHz AMD Opteron 6128 processors and 64GB of RAM
  - Each processor has 8 CPU-cores (for a total of 32 CPU-cores)

Enabling for timely what-if analysis (2)



Thanks for your attention

Questions?

Presenter:

pellegrini@dis.uniroma1.it

http://www.dis.uniroma1.it/∼pellegrini

Research Group:

http://www.dis.uniroma1.it/∼hpdcs

Framework:

https://github.com/cloudtm/cloudtm-autonomic-manager/tree/master/src/dags

ROOT-Sim:

http://www.dis.uniroma1.it/∼ROOT-Sim