CUMULO NIMBO

FP7-257993

Cloud Futures Workshop
Berkeley, California
May 8, 2012

CumuloNimbo: Parallel-Distributed Transactional Processing

Ricardo Jiménez-Peris, Marta Patiño, Iván Brondino – Univ. Politecnica de Madrid

José Pereira, Rui Oliveira, R. Vilaça, F. Cruz – Minho Univ.

Bettina Kemme, Yousuf Ahamad – McGill Univ.

SEVENTH FRAMEWORK PROGRAMME

CUMULO NIMBO

FP7-257993

Goals

- CumuloNimbo aims at solving the lack of scalability of transactional applications that represent a large fraction of existing applications.
- CumuloNimbo aims at conceiving, architecting and developing a transactional, coherent, elastic and ultra scalable Platform as a Service.
- Goals:
  - Ultra scalable and dependable -- able to scale from a few users to many millions of users while at the same time providing continuous availability;
  - Support transparent migration of multi-tier applications (e.g. Java EE applications, relational DB applications, etc.) to the cloud with automatic scalability and elasticity.
  - Avoid re-programming of applications and non-transparent scalability techniques such as sharding.
  - Support transactions for new data stores such as cloud data stores, graph databases, etc.

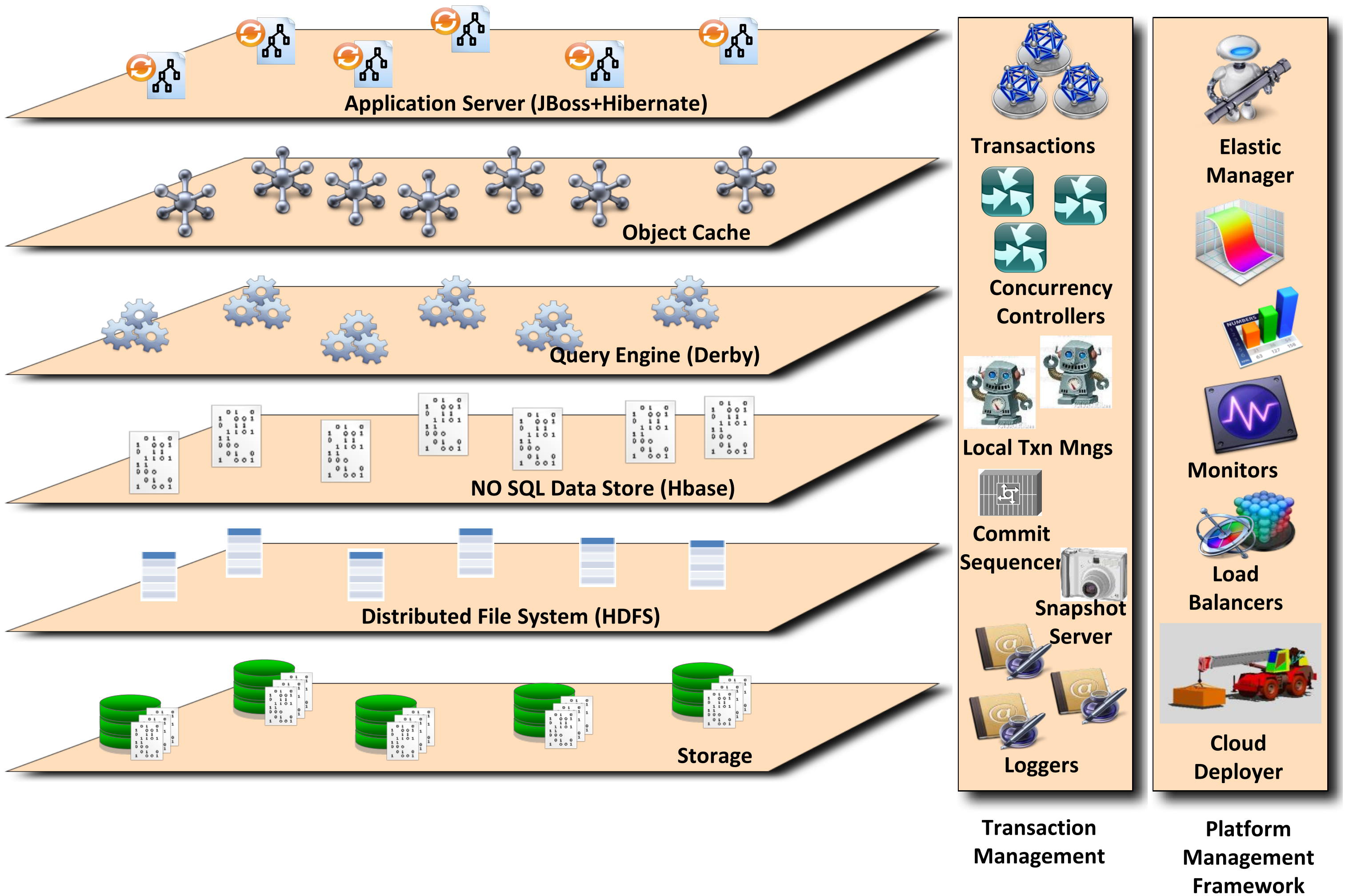
SEVENTH FRAMEWORK PROGRAMME

Challenges

- Main Challenges:
  - Update ultra-scalability (million update transactions per second and as many read-only transactions as needed).
  - Strong transactional consistency.
  - Non-intrusive elasticity.
  - Inexpensive high availability.
  - Low latency.
- CumuloNimbo goes beyond the State of the Art by scaling transparently transactional applications to very large rates without sharding, the current practice in Today's cloud.

Global Architecture

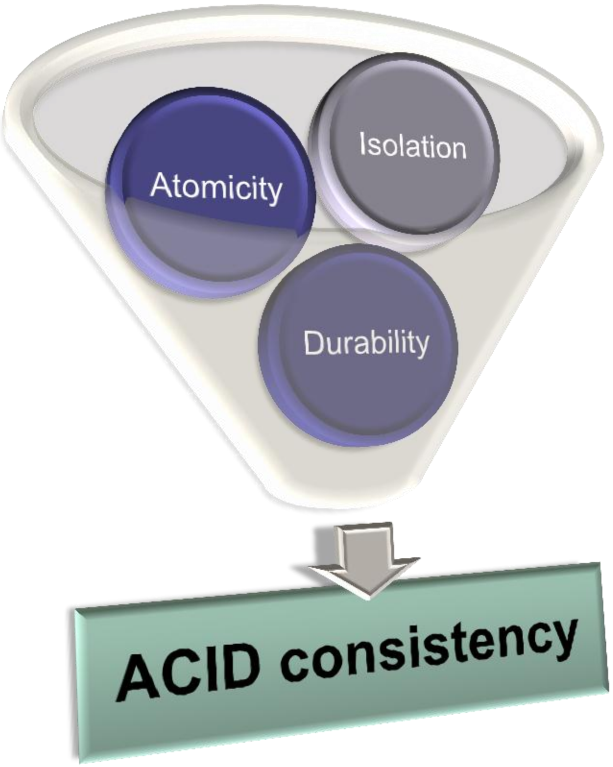
FP7-257993



Ultra-Scalable Transactional Processing

- Guarantees transactional coherence across all tiers: application server, object cache and database.
- No constraints on applications, transactional processing and data, no required a priori knowledge.
- Fully transparent:
  - Syntactically: no changes required in the application.
  - Semantically: equivalent behavior to a centralized system.
- Can be integrated with any other infrastructure requiring transactional support (e.g. graph databases).

Ultra-Scalable Transactional Processing: Approach

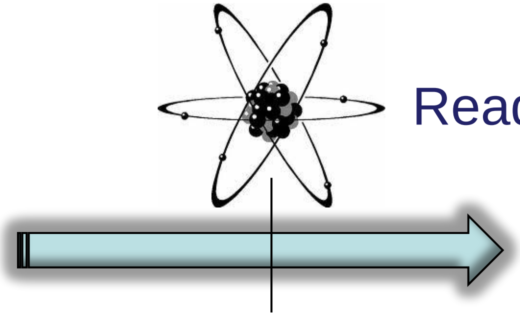


- Decomposition of transactional processing.
    - No DB or transactional manager as a single component.
- Atomicity, consistency, isolation and durability are attained separately.
    - Each component scaled independently but in a composable manner.
    - The first bottleneck is in a component able to do million update transactions per second.
- Transactions are committed in parallel.
- Based on snapshot isolation:
    - Avoids read/write conflicts providing an isolation very close to serializability.
    - Serializability can be implemented on top of it, if needed.

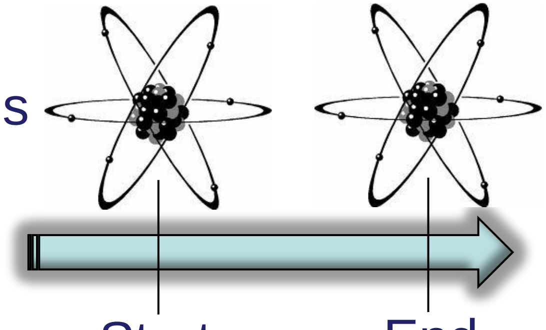
SEVENTH FRAMEWORK PROGRAMME

Ultra-Scalable Transactional Processing: Snapshot Isolation

- Serializability provides a fully atomic view of a transaction, reads and writes happen atomically at a single point in time.

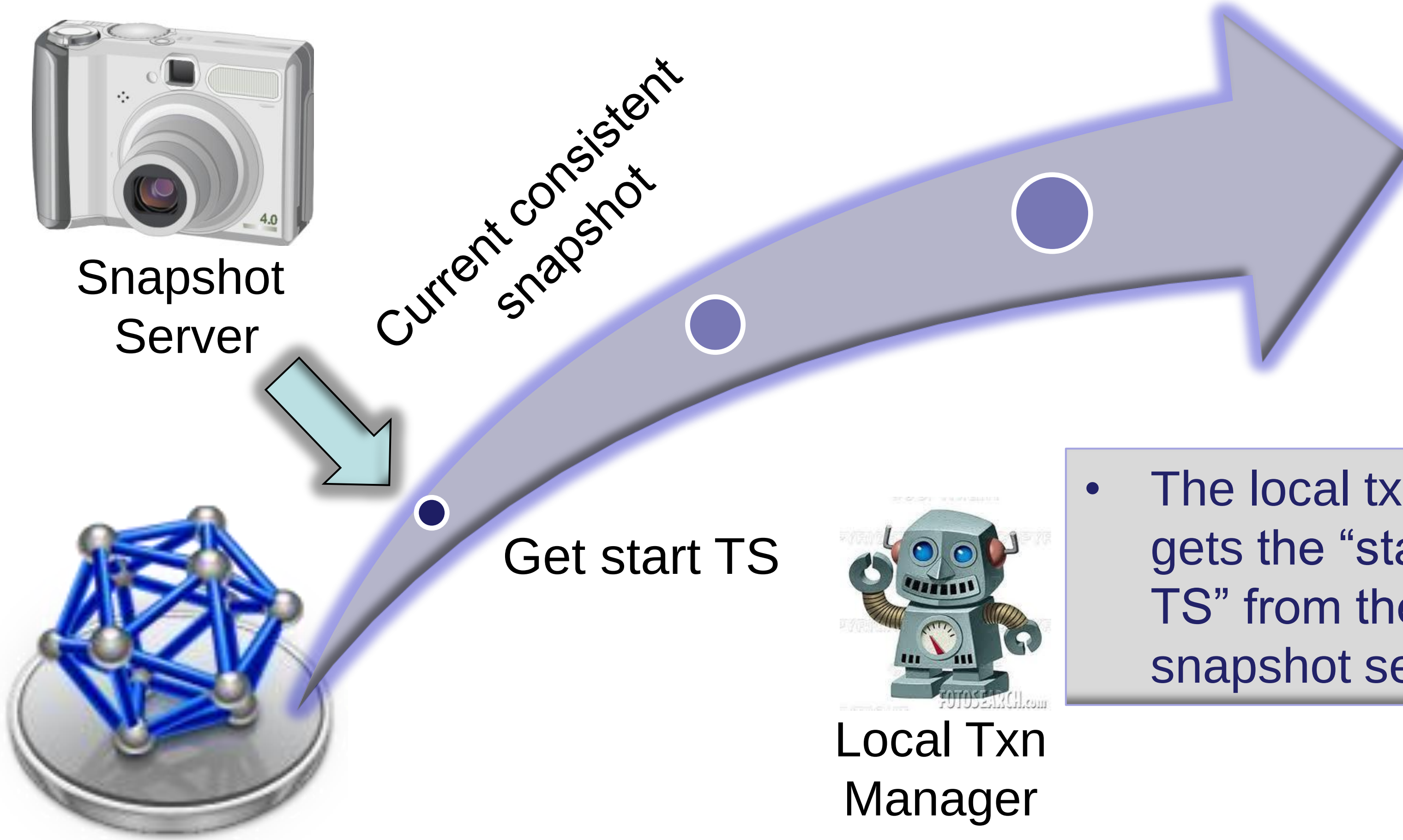


- Snapshot isolation splits atomicity in two points one at the beginning of the transaction where all reads happen and one at the end of the transaction where all writes happen.



Ultra-Scalable Transactional Processing: Components and Txn Life Cycle

FP7-257993



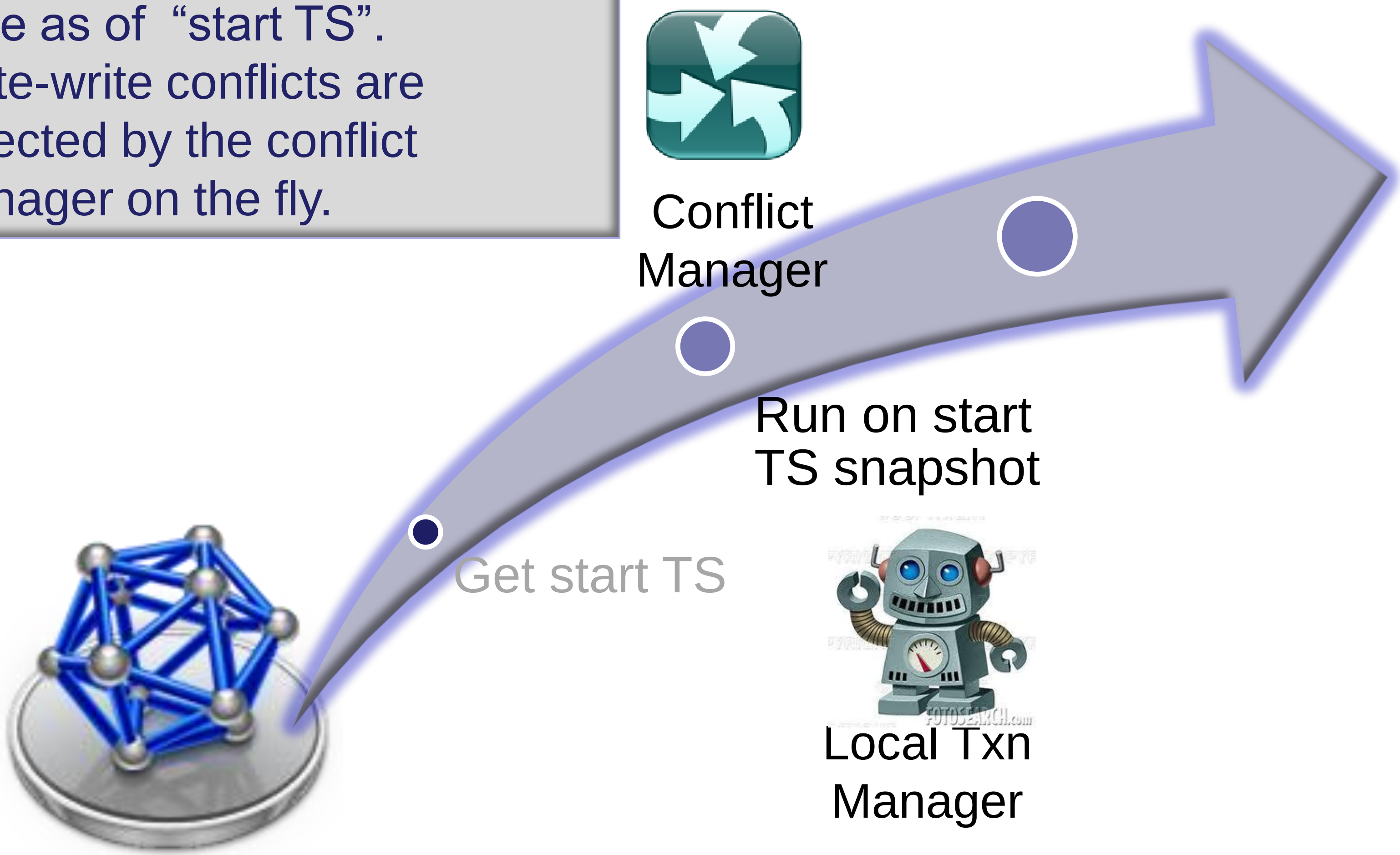
- The local txn mng gets the “start TS” from the snapshot server.

SEVENTH FRAMEWORK PROGRAMME

Ultra-Scalable Transactional Processing: Components and Txn Life Cycle

FP7-257993

- The transaction will read the state as of "start TS".
- Write-write conflicts are detected by the conflict manager on the fly.

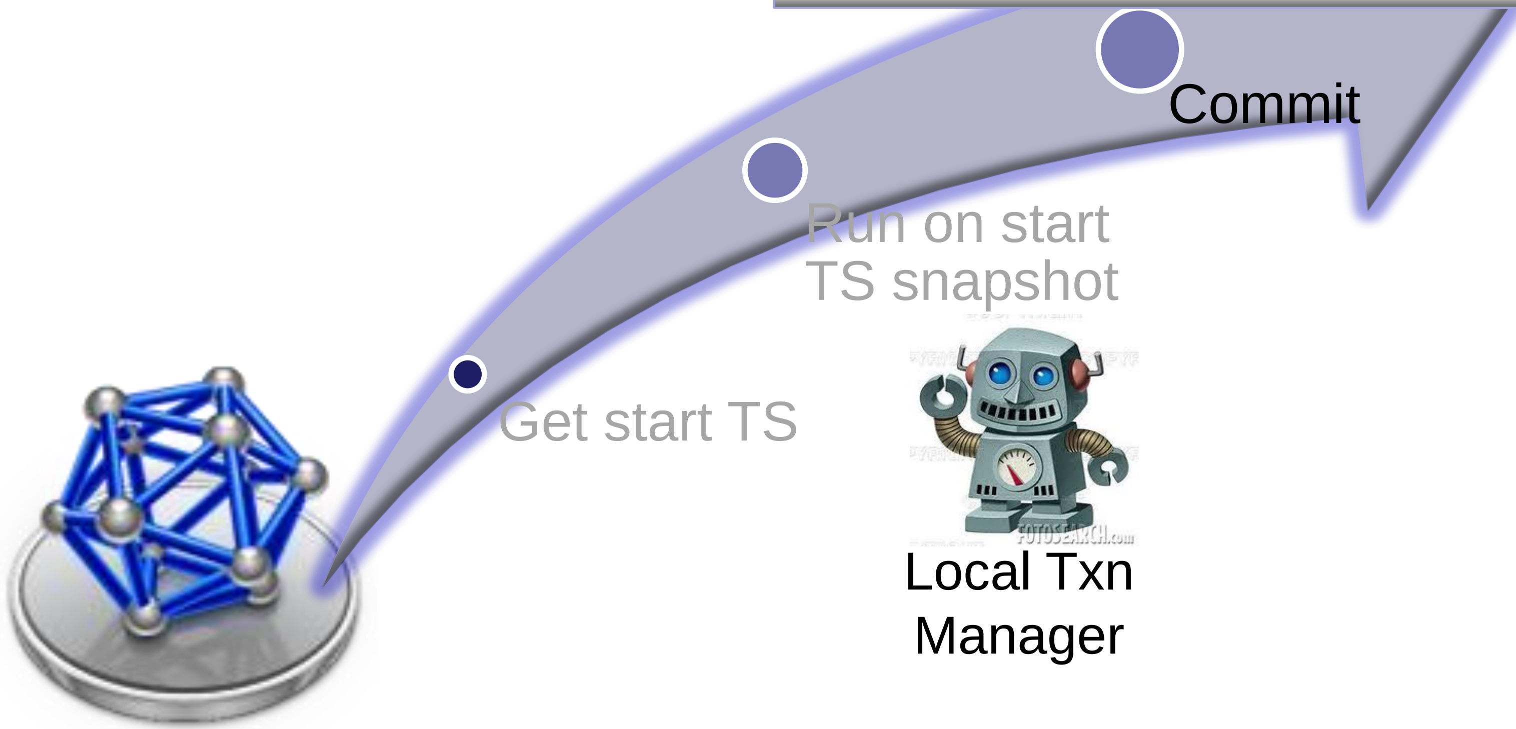


SEVENTH FRAMEWORK PROGRAMME

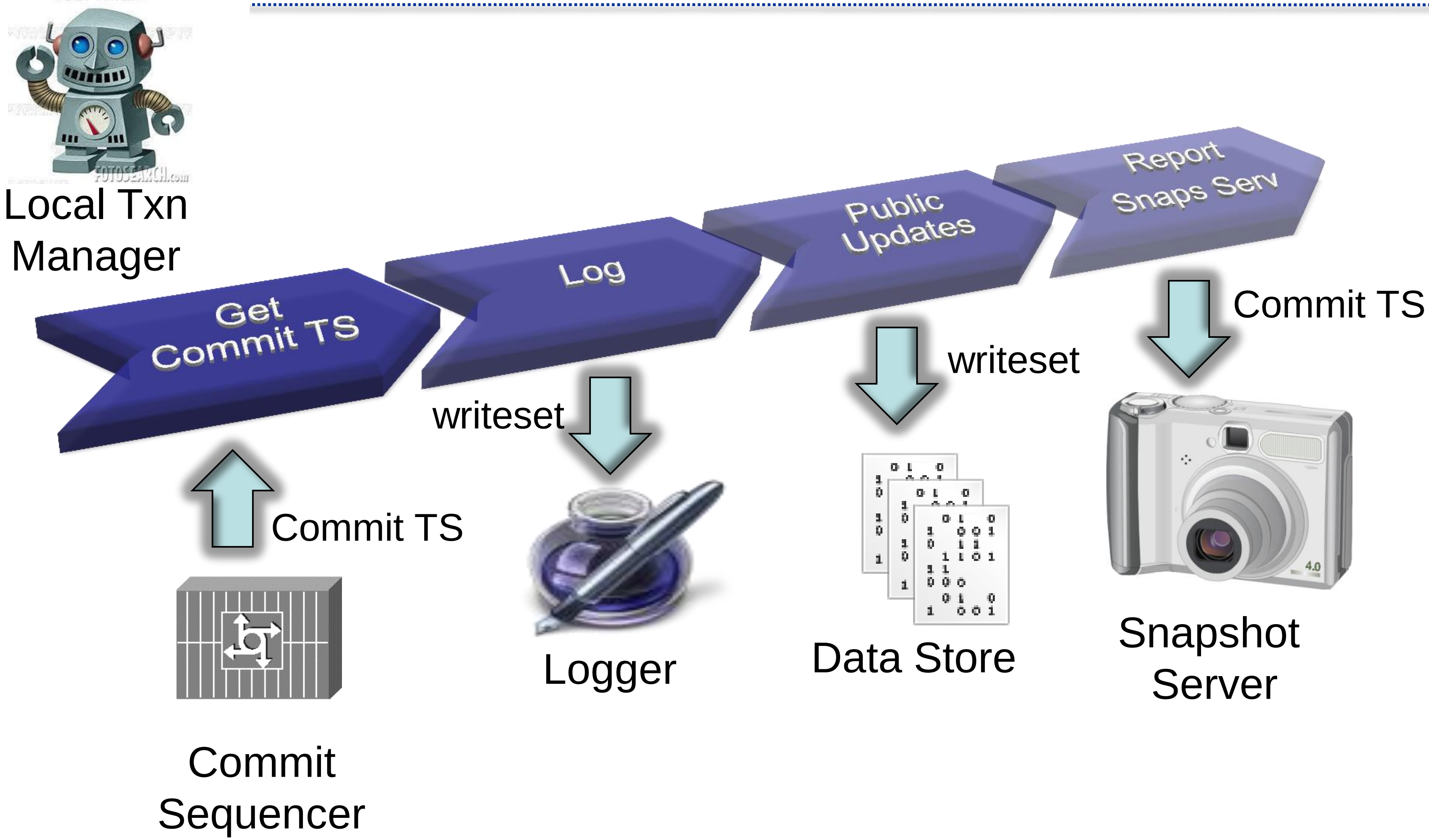
FP7-257993

Ultra-Scalable Transactional Processing: Components and Txn Life Cycle

- The local transaction manager orchestrates the commit.



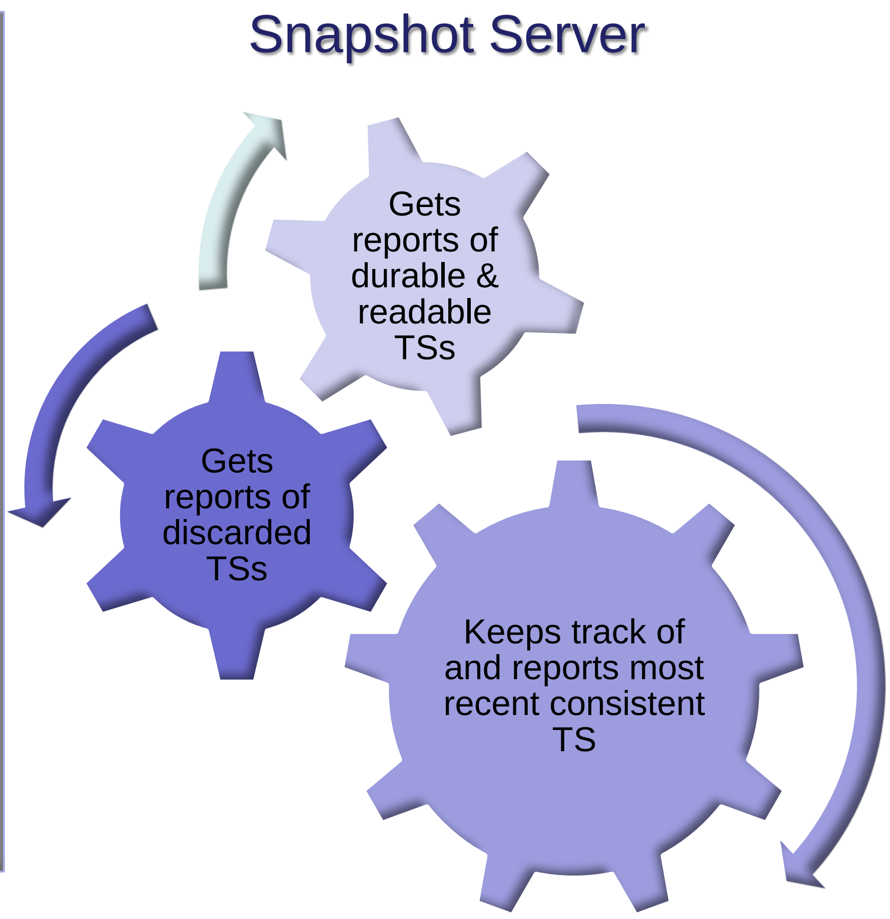
Ultra-Scalable Transactional Processing: Components and Txn Life Cycle



Ultra-Scalable Transactional Processing: Snapshot Server

FP7-257993

- The Snapshot server keeps track of the most recent snapshot that is consistent:
  - Its TS should such that there is no previous commit TS that is not yet durable and readable or it has been discarded.
  - That is, it keeps the longest prefix of used/discarded TSs such that there are no gaps.
- In this way transactions can commit in parallel and consistency preserved.



SEVENTH FRAMEWORK PROGRAMME

Ultra-Scalable Transactional Processing: Loggers

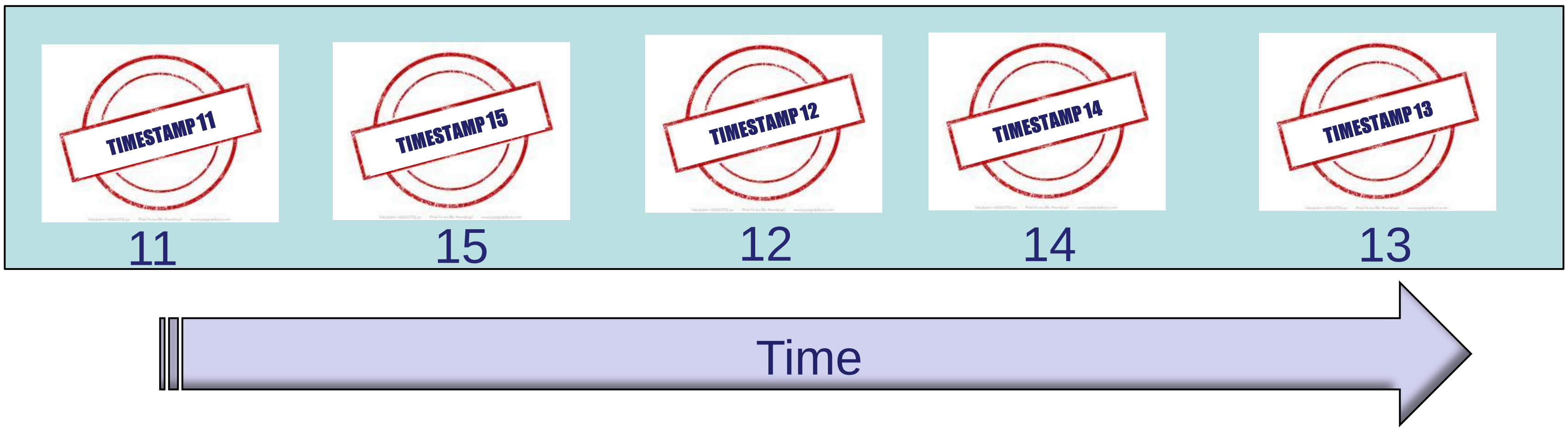
- Each logger takes care of a fraction of the log records.
- Loggers log in parallel and are uncoordinated.
- Loggers can be replicated.
- If this is the case the durability can be configured as:
  - To be in the memory of a majority of logger replicas (replicated memory durability).
  - To be in a persistent storage of a logger replica (1-safe durability).
  - To be in a persistent storage of a majority of logger replicas (n-safe durability).
- The client gets the commit reply after the writeset is durable (with respect the configured durability).

Ultra-Scalable Transactional Processing: Snapshot Server

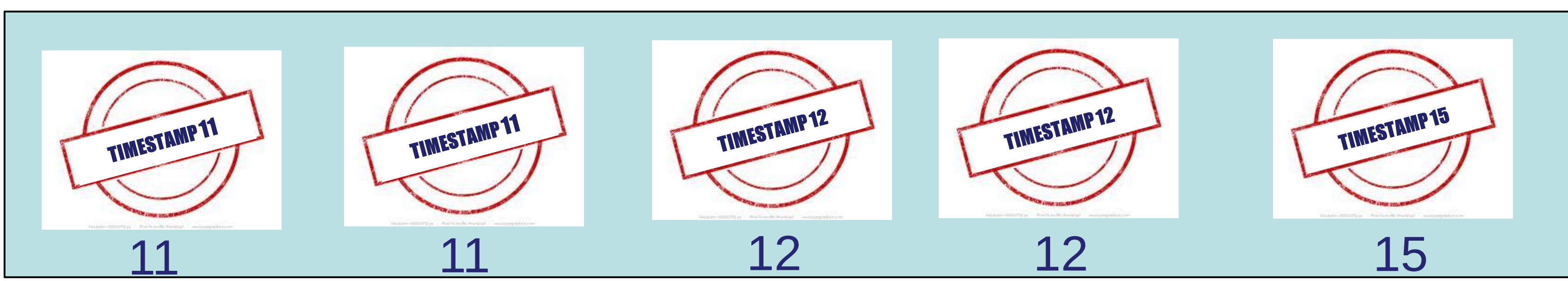
CUMULO NIMBO

FP7-257993

Sequence of timestamps received by the Snapshot Server



Evolution of the current snapshot at the Snapshot Server



SEVENTH FRAMEWORK PROGRAMME

CUMULO NIMBO

FP7-257993

Ultra-Scalable Transactional Processing: Increasing Efficiency

- The described approach so far is the original reactive approach.
- It results in multiple messages per update transaction.
- The adopted approach is proactive:
  - The local transaction managers report periodically about the number of committed update transactions per second.
  - The commit sequencer distributes batches of commit timestamps to the local transaction managers.
  - The snapshot server gets periodically batches of timestamps (both used and discarded) from local transaction managers.
  - The snapshot server reports periodically to local transaction managers the most current consistent snapshot.

SEVENTH FRAMEWORK PROGRAMME

CUMULO NIMBO

FP7-257993

Scalable Application Server and Distributed Object Cache

- We exploit JBoss and Hibernate as application server technology.
- We rely on their reflection capabilities (interceptors and hooks respectively) to intercept:
  - Transactional processing → Becomes ultra-scalable.
  - Second level cache → Becomes a distributed elastic cache.
- No changes required in the application server/persistency manager.
- The cache is multi-version aware guaranteeing full cache transparency.
- Approach applicable to any transactional application server either source code or with sufficient reflection capabilities.
- Support very large caches at both object and DB level enabling in-memory databases/application servers.

SEVENTH FRAMEWORK PROGRAMME

Scalable SQL processing: Query Engine

- SQL processing is performed at the SQL engine tier.
- A SQL engine instance:
  - Transforms SQL code into a query plan.
  - The query plan is optimized according the collected statistics (e.g. cardinality of keys).
  - Orchestrate the query plan execution on top of the distributed data store.
  - Returns the result of the SQL execution to the client.
  - Maintains updated the statistics in the data store.
- The SQL engine has been implemented by modifying Apache Derby, changing its transactional processing by CumuloNimbo's.

CUMULO NIMBO

FP7-257993

Scalable SQL Processing: Data Store

- To scale the data store, we leverage a key-value data store, Apache HBase.
- Relational tables are mapped to HBase tables.
- Secondary indexes are mapped to additional HBase tables that translate secondary keys into primary keys.
- Traffic between the query engine and HBase instances is minimized by:
  - Exploiting HBase filters to implement scan operators.
    - Reduces the cost of scans.
  - Leveraging HBase co-processors to compute local statistics on each region necessary.
    - Reduces the cost of statistics necessary for query optimization.

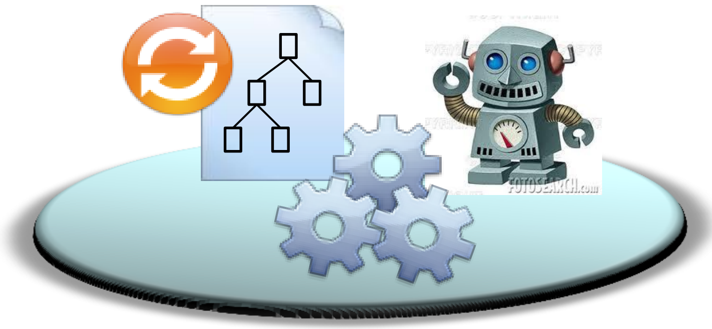
SEVENTH FRAMEWORK PROGRAMME

Efficient Deployment
Collocation of Instances across Tiers

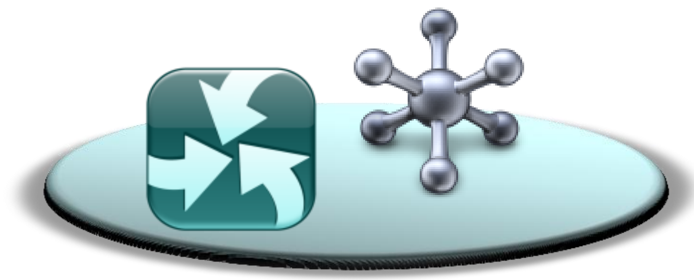
- One of the main goals is throughput efficiency, i.e., to attain a particular required throughput with the minimal number of resources.
- Both elasticity and dynamic load balancing contribute towards this goal.
- But another aspect is related on how to deploy the multiple instances of the multiple tiers to minimize the distribution overhead.
- Collocation of tiers has been considered and actually performed to diminish the number of distributed hops required to process a transaction.

Efficient Deployment
Collocation of Instances across Tiers

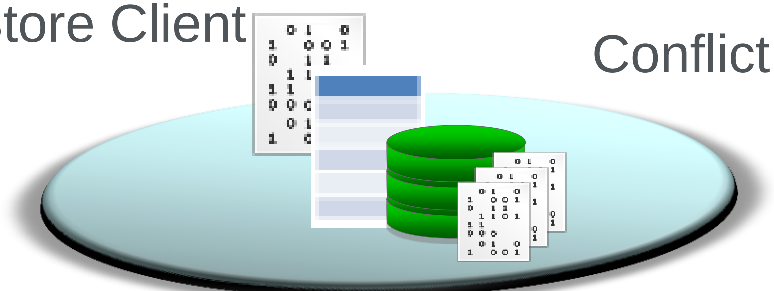
FP7-257993



Application Server instance
+ ORM instance
+ Local Txn Mng instance
+ Query engine instance
+ Key-Value Data Store Client



Distributed Cache instance
+
Conflict Manager instance



Key-Value Data Store
+ Parallel Distributed FS +
Storage Manager

SEVENTH FRAMEWORK PROGRAMME

Ongoing work: Elasticity

- Elasticity is controlled at each layer with customized elastic rules.
  - For instance, the object cache can provision nodes either due to lack of memory or CPU saturation.
- Elasticity is combined with dynamic load balancing to guarantee that provisioning is only triggered when needed.
- Non-intrusive reconfiguration:
  - Focusing on maintaining throughput close to the peak one during reconfiguration.

Ongoing work: Fault Tolerance

- Replication is used for high availability and not for scaling.
  - Low cost data fault tolerance
    - Pushed down to the storage layer (distributed file system)
    - Outside the transaction response time path.
  - Fault tolerance for other components with a simple approach
    - Configuration and vital data stored on a replicated data store (Zookeeper).
    - Single replicated server keeps track of configuration metadata for all tiers and instances.
  - Fault tolerance of critical components:
    - Specialized replication that maximizes throughput and minimizes latency.
    - Commit server, snapshot server, loggers.

CUMULO NIMBO

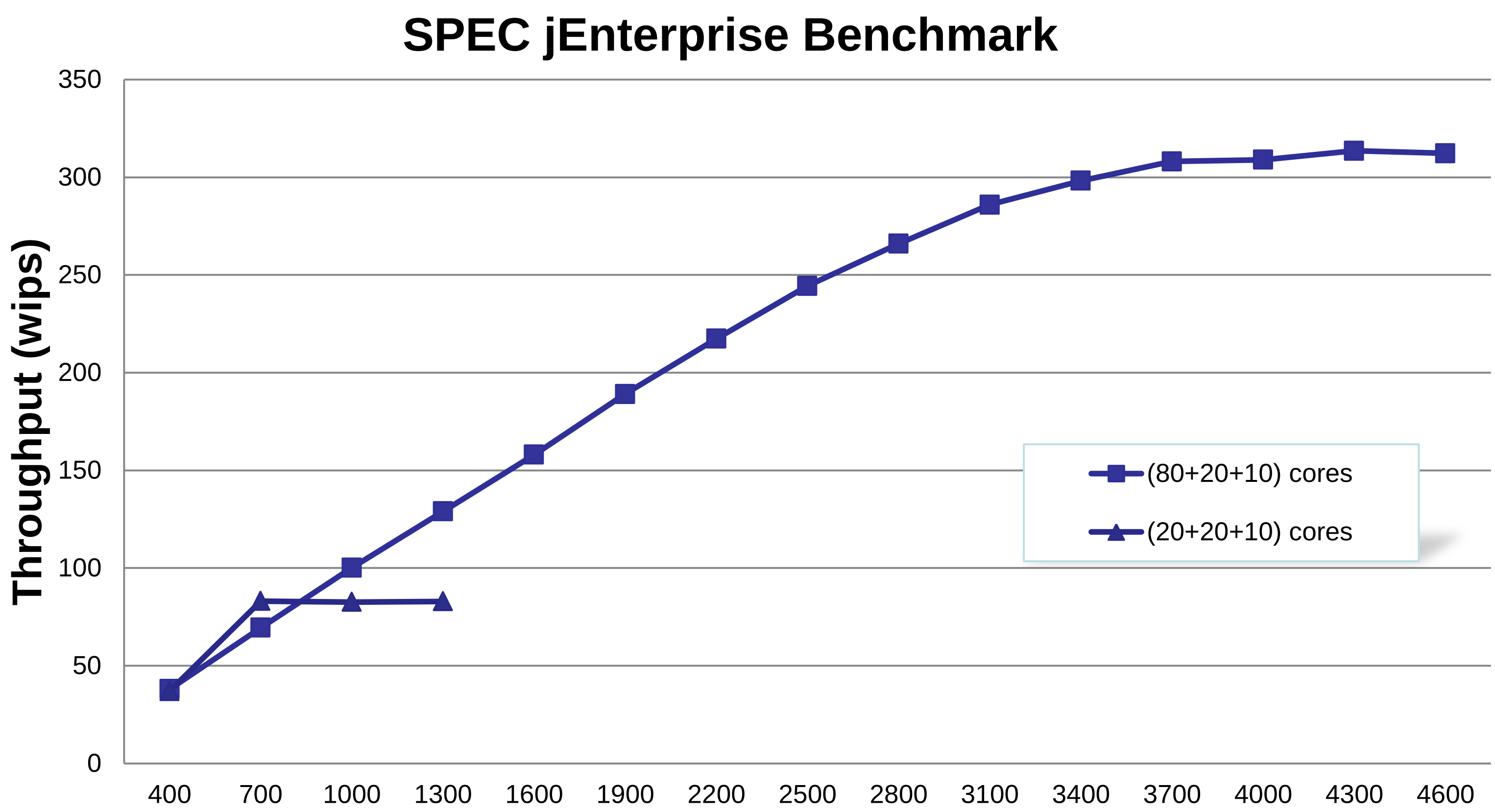
FP7-257993

Evaluation Setup

- HBase+HDFS deployed on 5+1 dual-core nodes (12 cores).
- Distributed cache deployed on 5 dual-core nodes (10 cores).
- Transaction manager core components deployed on 2 dual-core nodes (4 cores).
- JBoss+Hibernate+Derby+HBase client deployed on 5 and 20 quad-core nodes (20 and 80 cores).
- Configuration manager deployed on a dual-core node (2 cores).
- Total cores: 28+20 to 80 (48 to 108 cores)

SEVENTH FRAMEWORK PROGRAMME

Scalability Results



Linear scalability with 100+ cores
Currently exercising 300+ cores

Contact Information

- Ricardo Jiménez-Peris
  - Technical Coordinator.
  - Univ. Politécnica de Madrid.
  - rjimenez@fi.upm.es

- http://cumulonimbo.eu