Clouder: A Flexible Large Scale Decentralized Object Store

Ricardo Vilaça

December 14, 2012

Context

- Traditional RDBMS were not designed for massive scale.
- Storage of digital data has reached unprecedented levels.
- Trade consistency for availability.

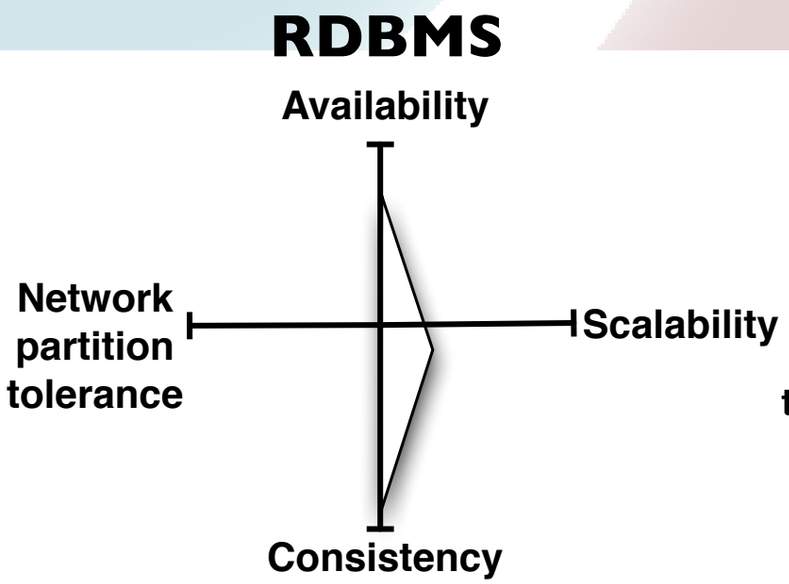
- Massive-scale distributed computing is a challenge at our doorstep.
- Centralized storage and processing making extensive and flexible data partitioning unavoidable.
- Emergence of Cloud Computing paradigm/business model.

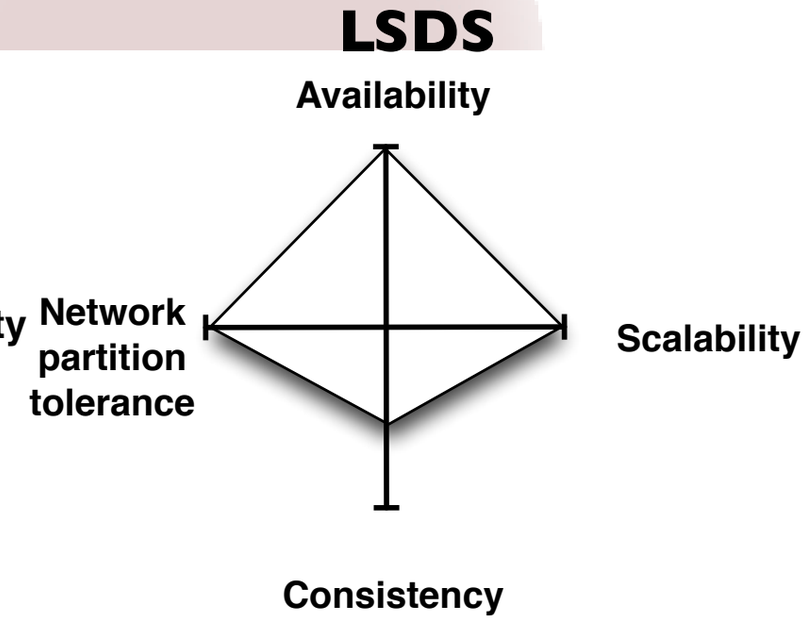
Large scale data stores

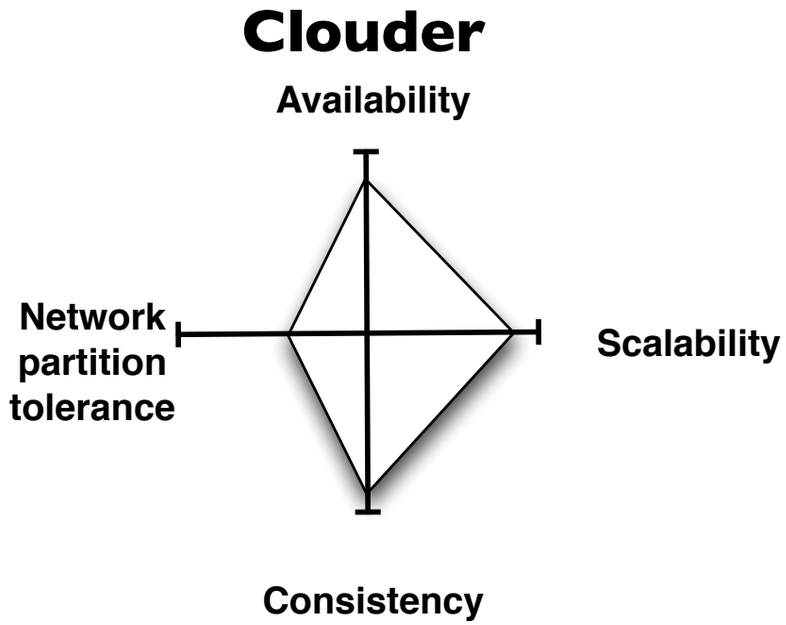
- Elastic computing.
- Highly decentralized, scalable, and dependable systems.
- Google's Bigtable, Amazon's Dynamo, Yahoo's PNUTS, Facebook's Cassandra.
- A simple data store interface, that allows applications to insert, query, and remove individual elements.
- Forfeit complex relational and processing facilities.
- Specific narrow tradeoff between consistency, availability, performance, scale, and cost.

Tradeoffs

- In most enterprises there isn't a large in-house research development team to redesign applications.
- Hard to provide a smooth migration path for existing applications, mostly SQL based.
- Hurdle to the adoption of Cloud computing by a wider potential market.







Problem and goals

- New generation large scale data stores and traditional RDBMS are in disjoint design spaces, and there is a huge gap between them.
- This thesis aims at reducing this gap by:
  - seeking mechanisms to provide additional consistency guarantees;
  - higher-level data processing primitives in large scale data stores:
    - extending data stores with additional operations, such as general multi-item operations;
    - coupling data stores with other existing processing facilities.

HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

DataDroplets

MAPi

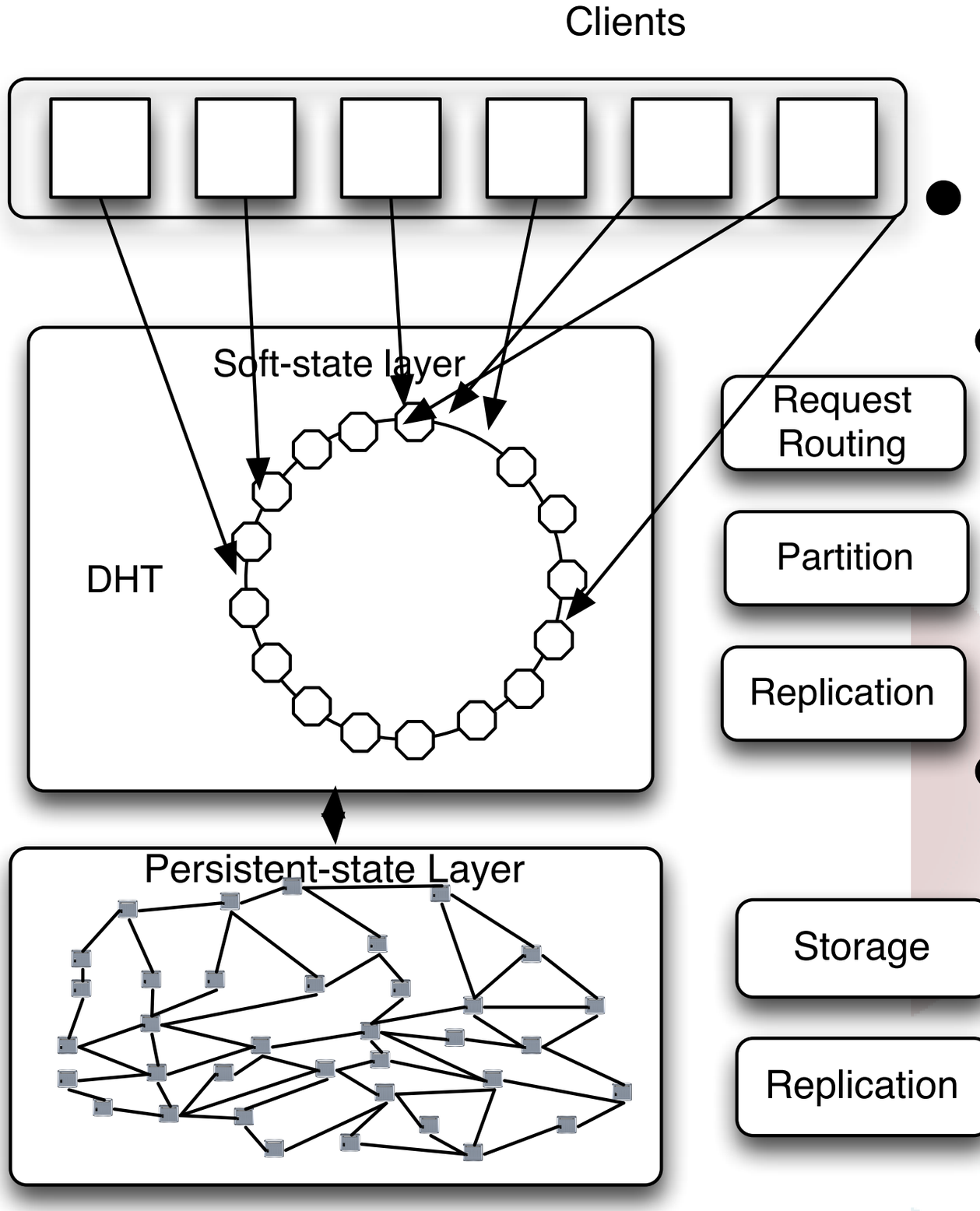
INESCTEC
TECNOLOGIA E CIÊNCIA
LABORATÓRIO ASSOCIADO

Universidade do Minho

Clouder

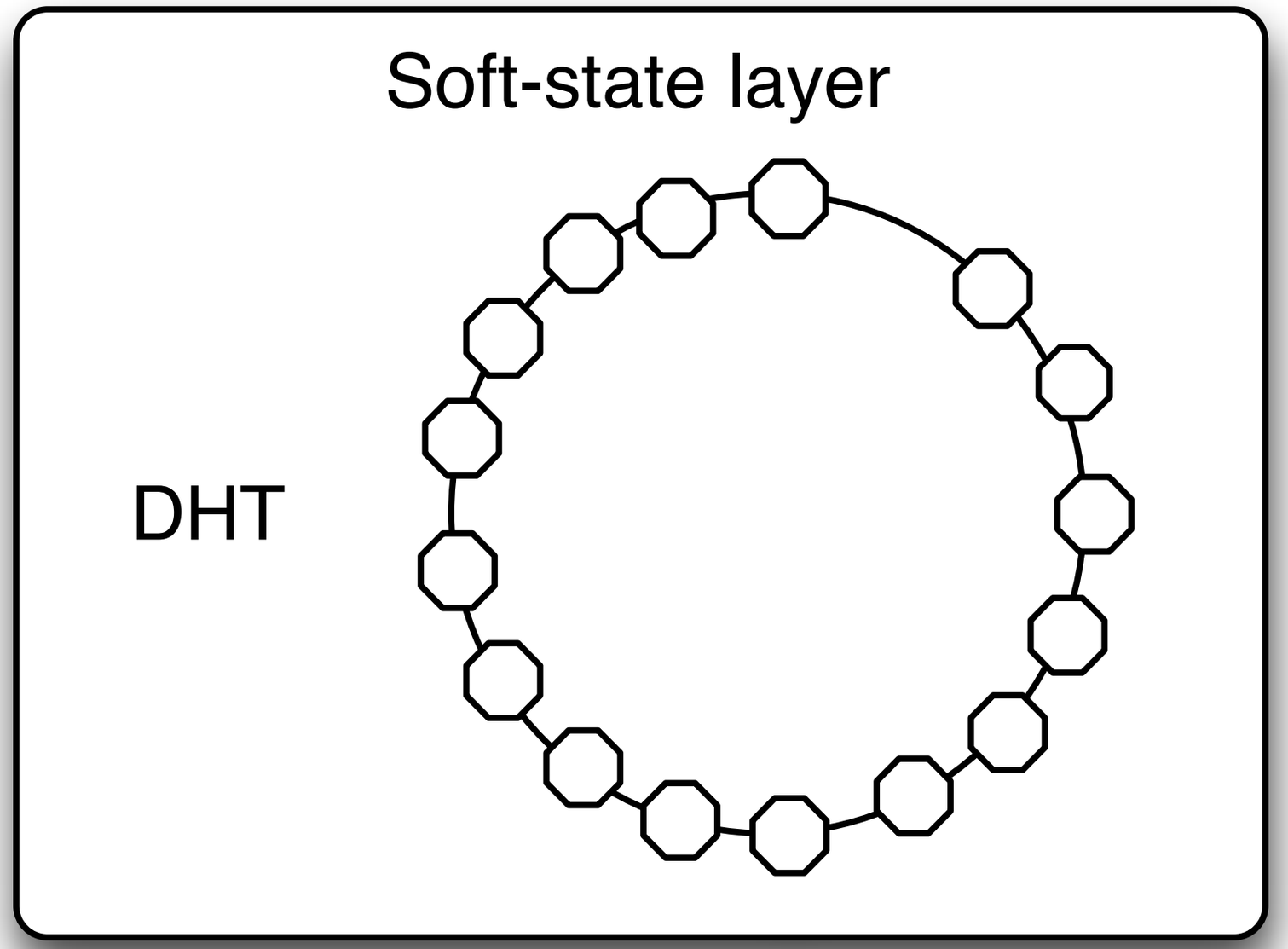
- Assumptions
  - Controlled environment: same owner, reasonable stable membership, specific set of applications.
  - Nodes can be commodity machines mainly dedicated to enterprise or academic business tasks.
- Goals
  - General purpose consistent, conflict-free, data storage, and in-place processing capabilities.
  - Allow to leverage large computing infrastructures.
  - Dependability in face of massive distributed data.

Clouder architecture



- Hybrid
- Structured approach suited to a smaller scale and to offer an adequate interface to the client.
- Unstructured approach is fit to manage the massive scale of the system.

Clouder architecture



Request Routing

Partition

Replication

DataDroplets

- Elastic data store providing atomic access to tuples and flexible data replication.
- Environment
  - Hundreds of nodes with a reasonably stable membership.
- Functionalities
  - Storage cache managed as allowed by the adopted consistency criteria.
  - Data partitioning.
  - Request routing and processing.

DataDroplets data model and API

Key	Value	Tags
1	a	Blue, Oval
3	b	Red, Rectangular
4	c	Oval, Red, Rectangular

$$\text{String} \rightharpoonup (K \rightharpoonup (V \times \mathcal{P}\ \text{String}))$$

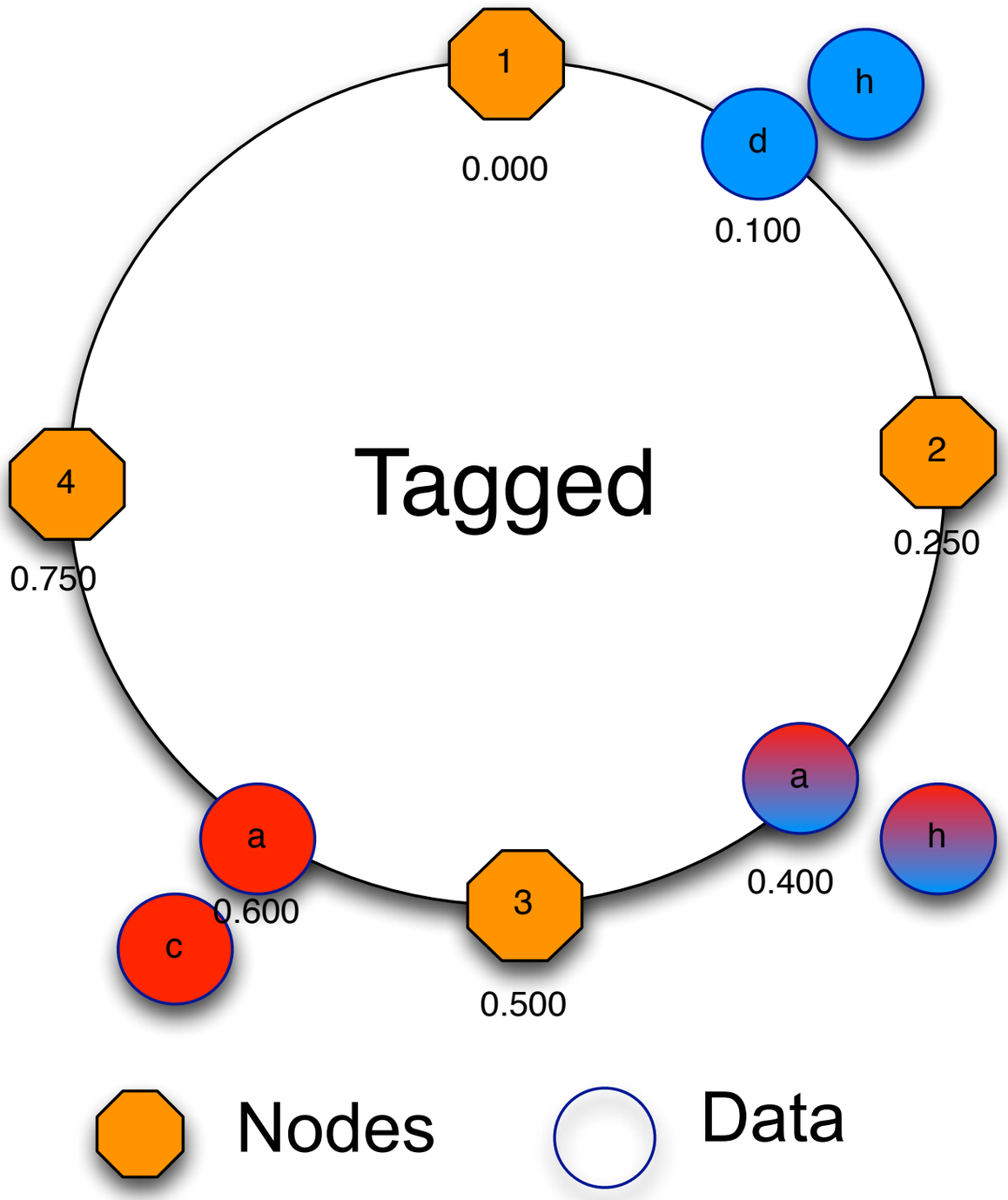
- New Operations:
  - Complex queries containing tags, wildcards, or ranges of tags.

State of the art data placement

- Existing data placement strategies only efficiently support single tuple or range queries.
- Most applications have however general multi-tuple queries.
  - Its performance is highly affected by the data placement strategy.
  - Correlation
    - The probability of a pair of tuples being requested together in a query is not uniform but often highly skewed.
    - Stable over time for most applications.
    - Challenge of achieving such placement in a decentralized fashion.

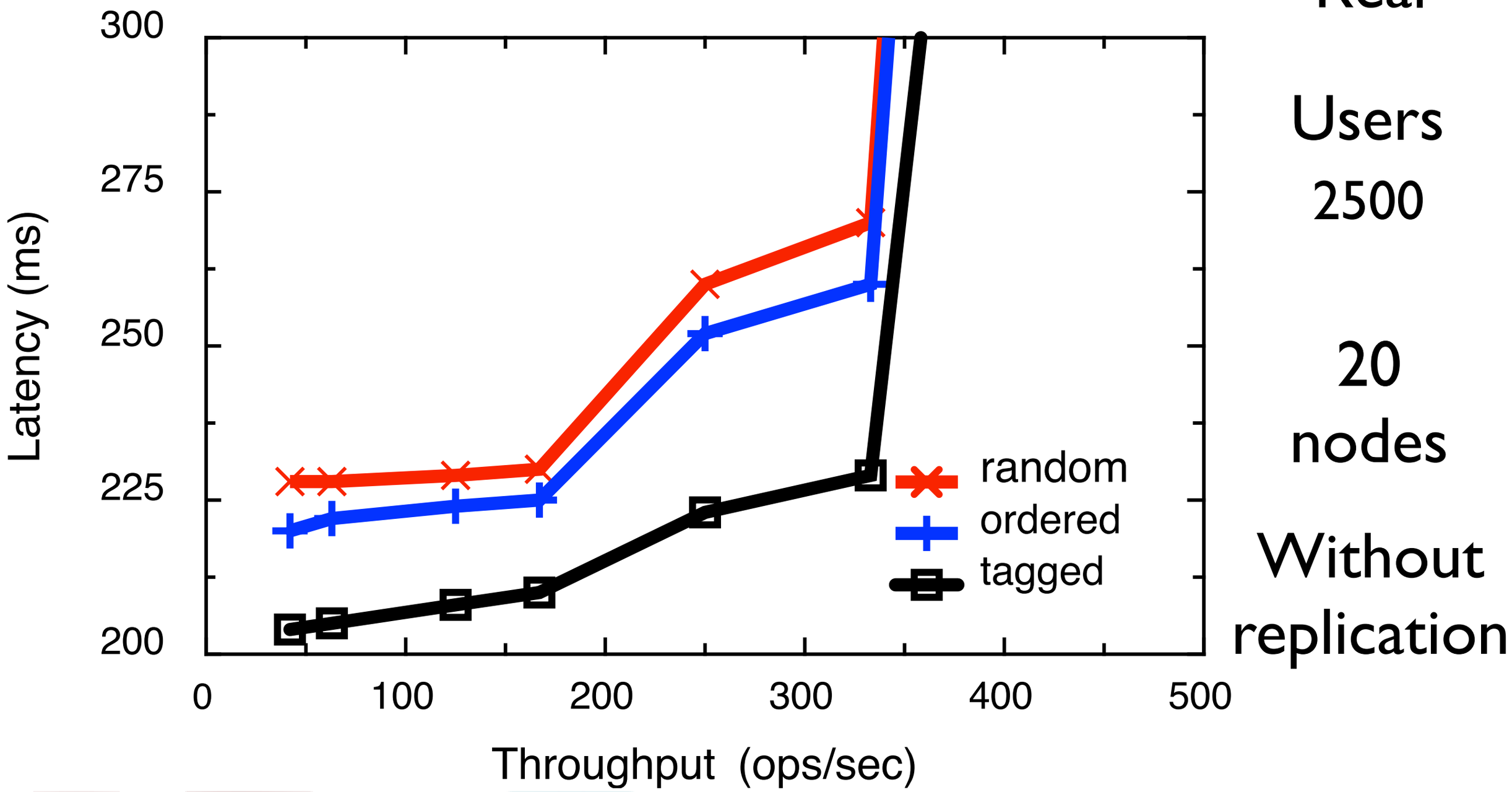
Our approach: Tagged

- Set of tags defined per tuple.
- Uses a dimension reducing and locality-preserving indexing scheme (SFC).



Data placement

Twitter alike workload



HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

SQL on large scale data stores

MAPi

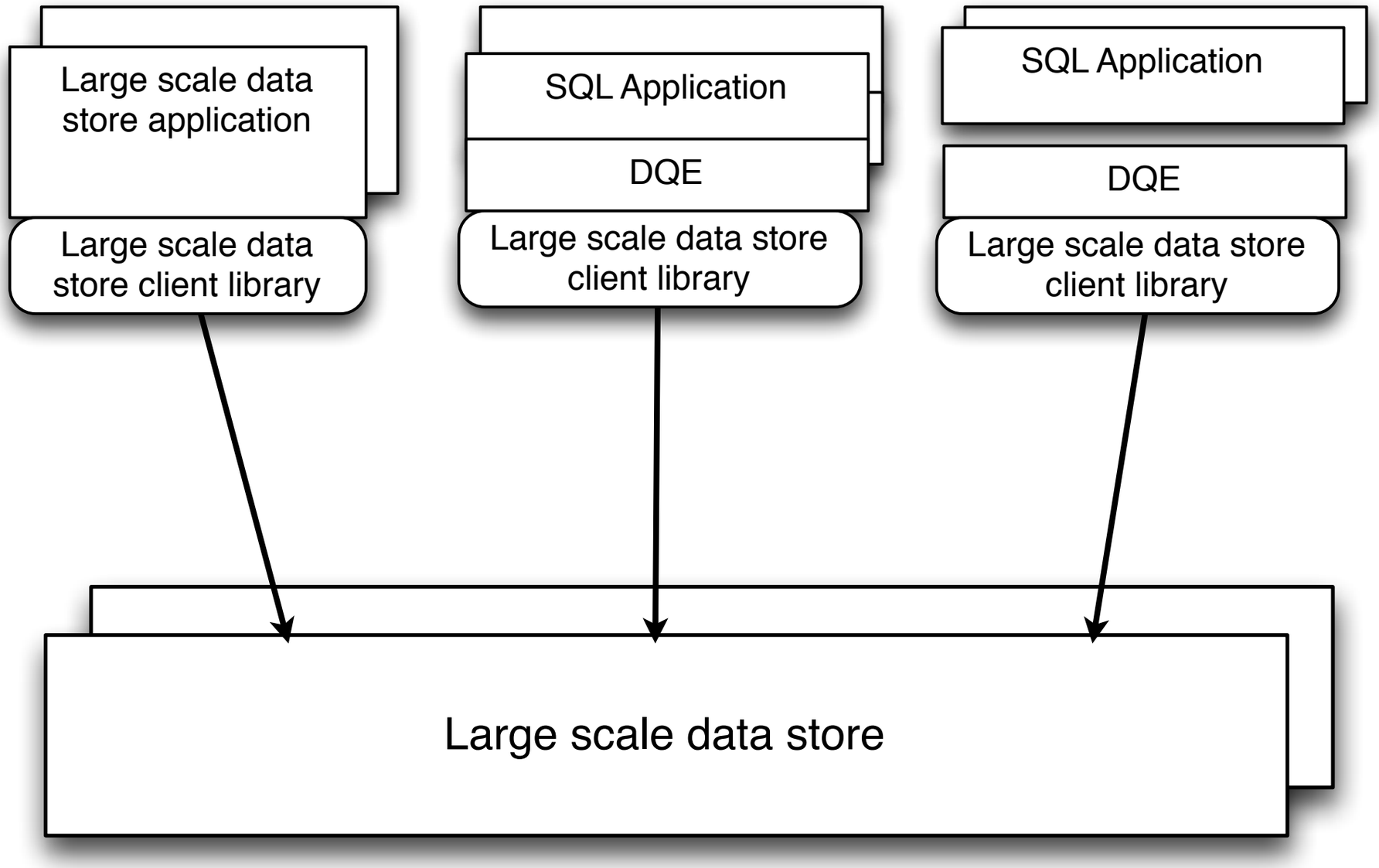
INESCTEC
TECNOLOGIA E CIÊNCIA
LABORATÓRIO ASSOCIADO

Universidade do Minho

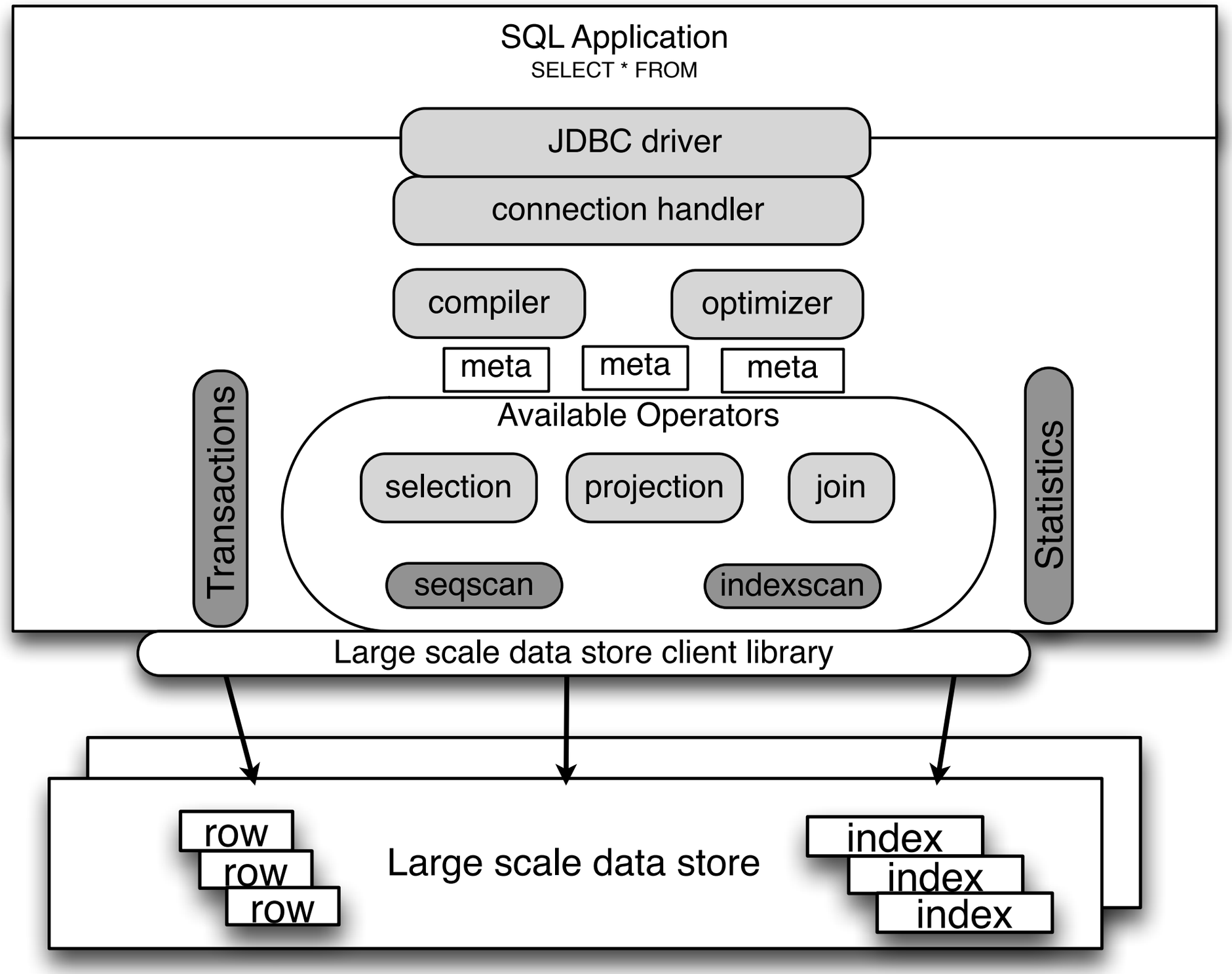
Distributed query engine

- Attaining scalable SQL processing on top of a large scale data store.
- Does not introduce coordination bottlenecks.
- Use the query engine functionality of RDBMS removing components that limit scalability.
  - Separation of concerns: execution, transactions, and storage.
- Resolve impedance mismatches of the relational model and the data model of large scale data stores.

Distributed query engine architecture



Distributed query engine architecture



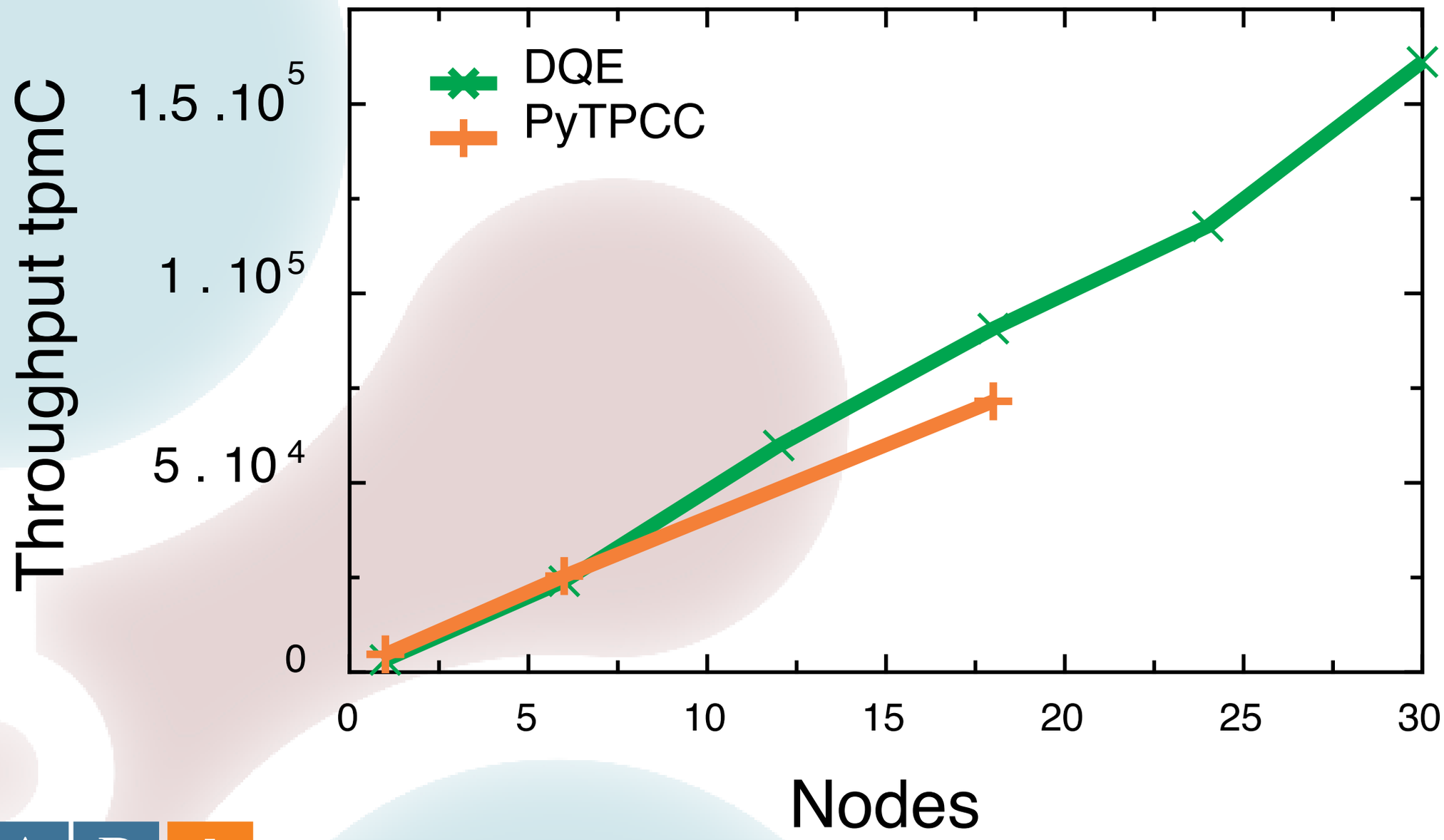
DQE overhead

- YCSB workload.

Workload	1 client 100 tps		50 clients 100 tps	
Operation	HBase	DQE	HBase	DQE
Insert	0.58	0.93	1.04	1.98
Update	0.51	1.3	2.66	3.1
Read	0.53	0.79	1.63	1.7
Scan	1.43	2.9	4.64	6.1

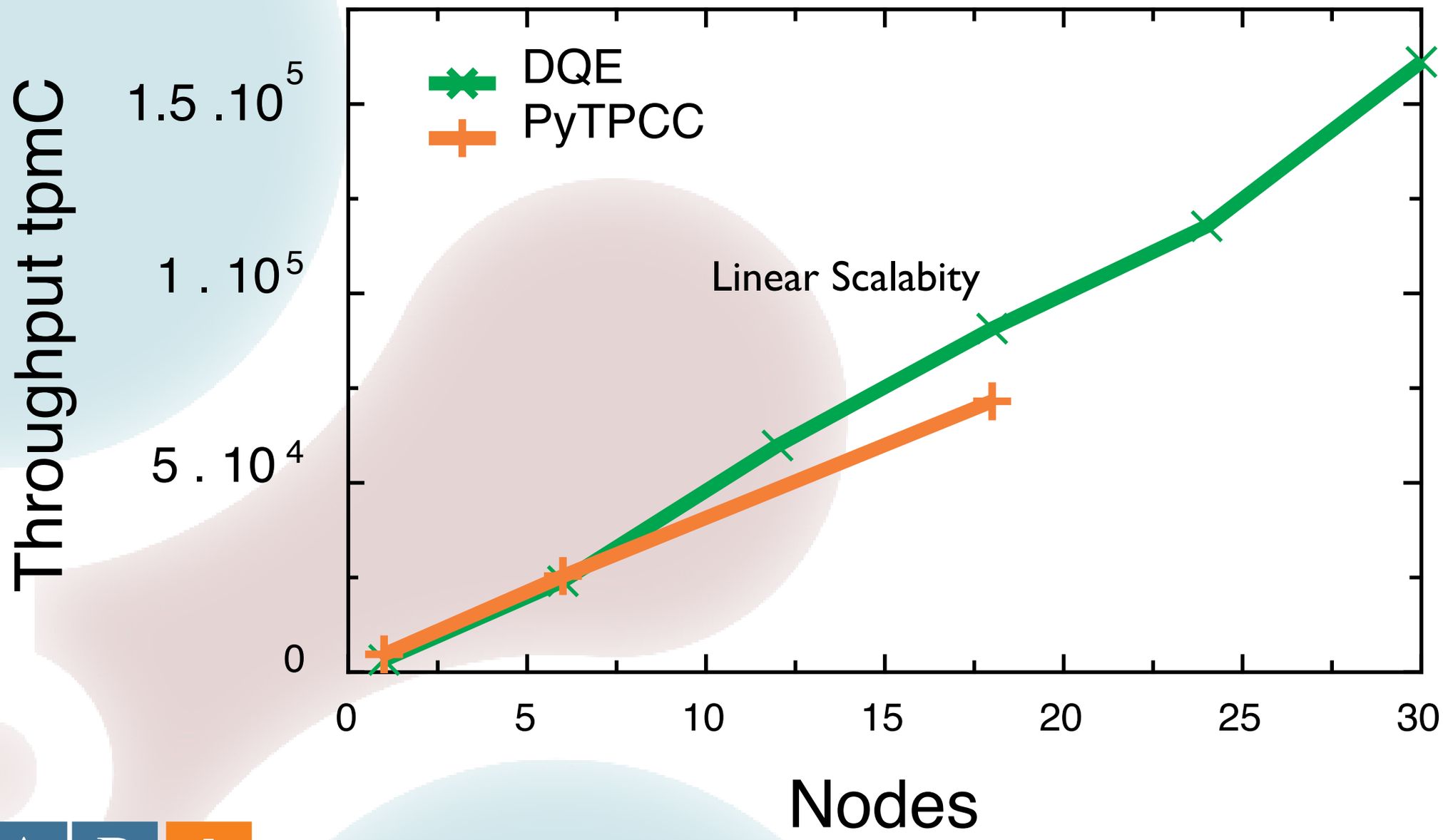
DQE non-transactional scalability

- 1 to 30 HBase RS (1 to 10 DQE).
- TPC-C workload.



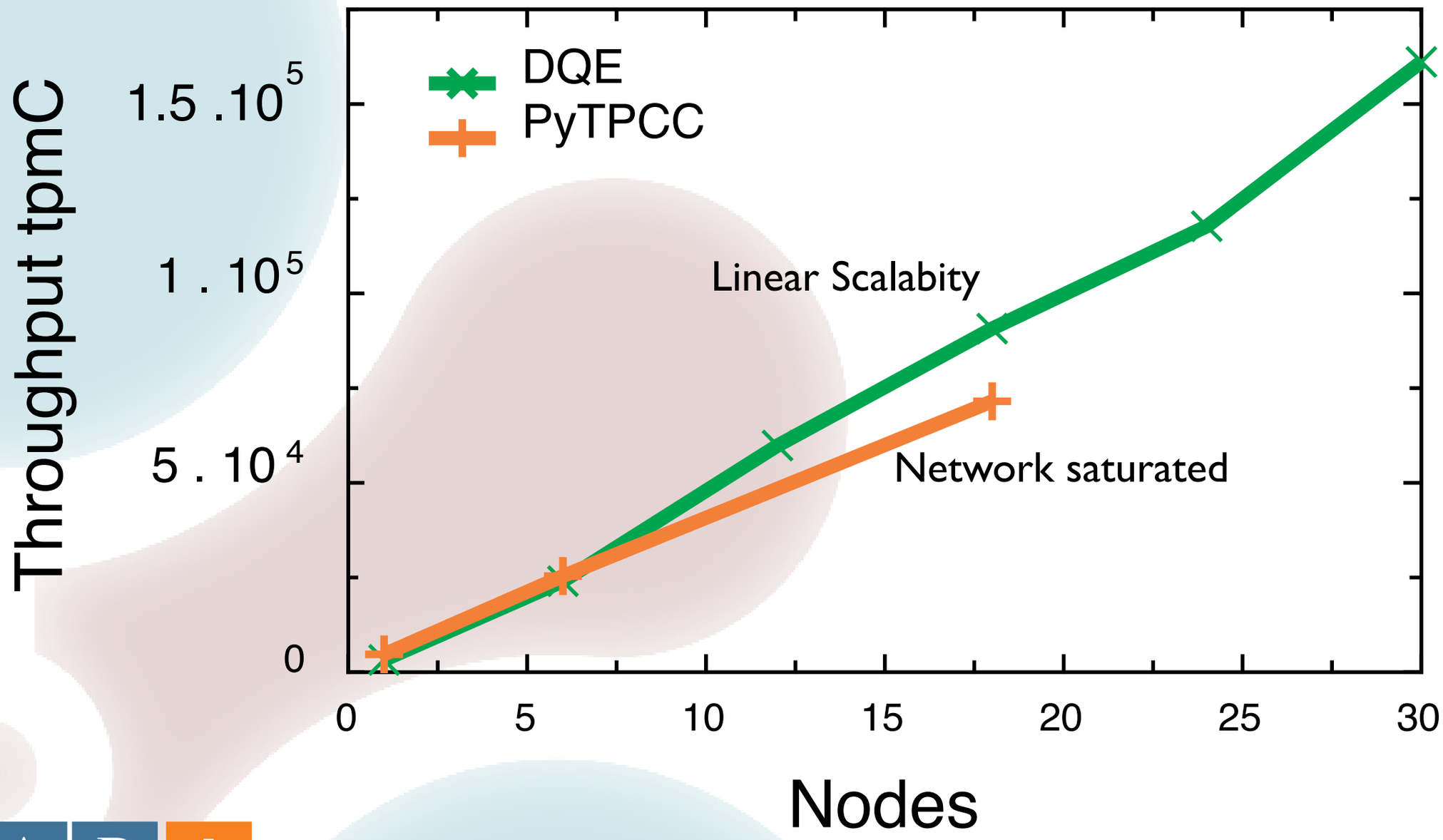
DQE non-transactional scalability

- 1 to 30 HBase RS (1 to 10 DQE).
- TPC-C workload.



DQE non-transactional scalability

- 1 to 30 HBase RS (1 to 10 DQE).
- TPC-C workload.

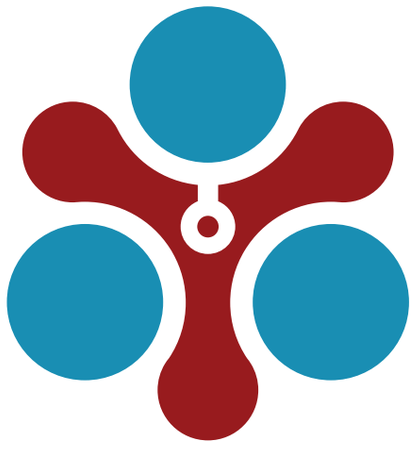


Main contributions

- A new architecture for large scale data stores.
- New large scale data store with additional consistency guarantees and higher level data processing primitives.
- A multi-tuple data placement strategy that allows to efficiently store and retrieve large sets of related data at once.
  - Provides better results in overall query performance.
  - Usefulness of having multiple simultaneous placement strategies in a multi-tenant system.
- Design modifications to existing relational SQL query engines allowing them to be distributed, and efficiently run SQL on scalable data stores while preserving scalability.

Main results

- A prototype of a new large scale data store, following the proposed architecture and implementing the novel data placement strategy.
  - Extensive simulation and real results, under a workload representative of applications currently exploiting the scalability of large scale data store.
- A simple but realistic benchmark for large scale stores based on Twitter and currently known statistical data about its usage.
- A prototype of a distributed SQL query engine running on a large scale data store.
  - Experimental results, using standard industrial database benchmarks.



HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

Clouder: A Flexible Large Scale Decentralized Object Store

Ricardo Vilaça
Supervised by Rui Oliveira

December 14, 2012

MAPi

INESCTEC
TECNOLOGIA E CIÊNCIA
LABORATÓRIO ASSOCIADO

Universidade do Minho

Publications

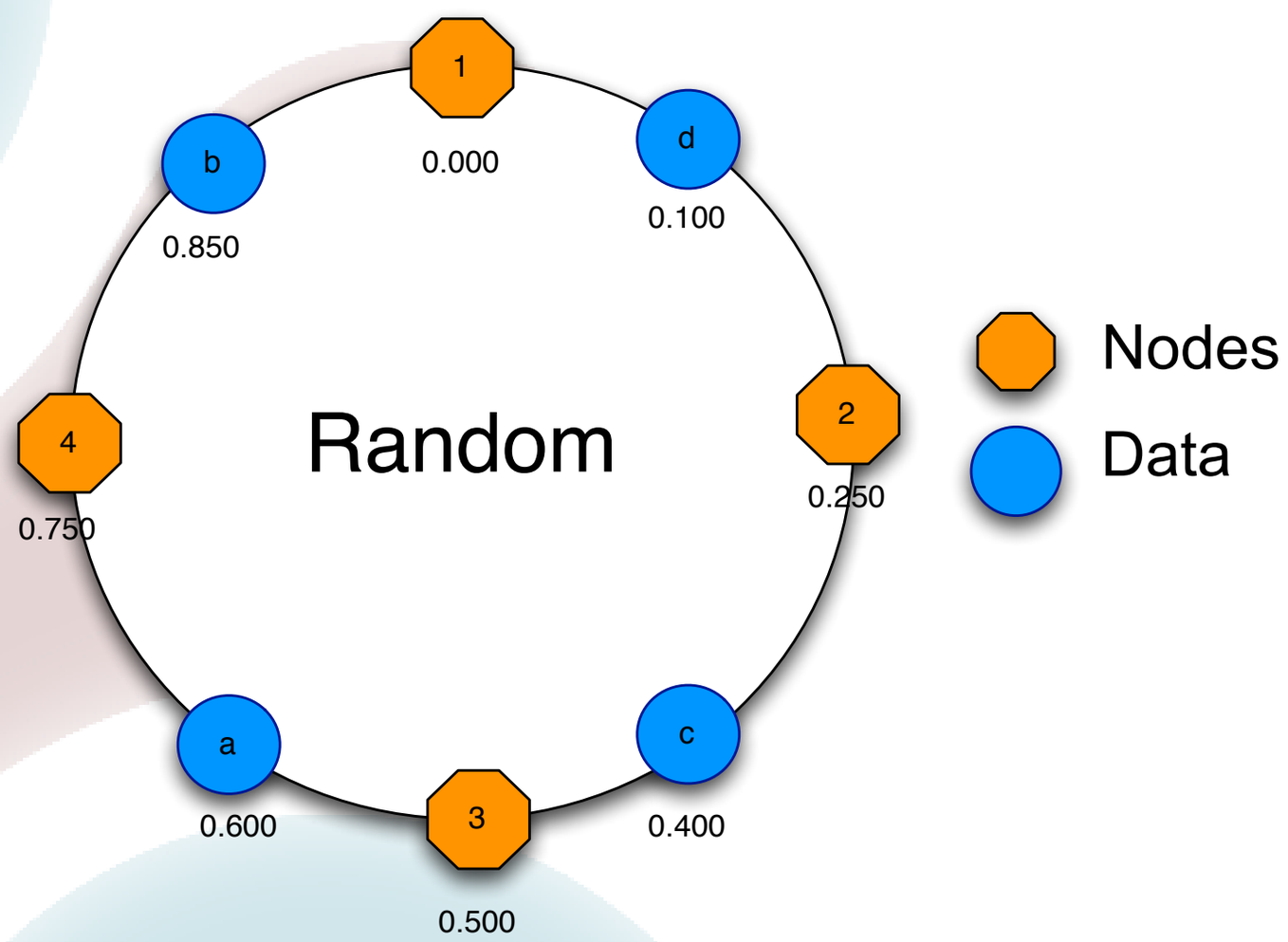
- Involved in a National research project, Stratus (PTDC/EIA-CCO/115570/2009), and two European research projects: CumuloNimbo (FP7-257993) and GORDA (FP6-IST2-004758)
- Ricardo Vilaca, Rui Carlos Oliveira and José Pereira. A correlation-aware data placement strategy for key-value stores. In DAIS 2011
- Miguel Matos, Ricardo Vilaça, José Pereira and Rui Oliveira. An epidemic approach to dependable key-value substrates. In DCDV 2011
- Ricardo Vilaça, Francisco Cruz and Rui Oliveira. On the expressiveness and trade-offs of large scale tuple stores. In DOA 2010
- Ricardo Vilaça and Rui Oliveira. Clouder: a flexible large scale decentralized object store: architecture overview. In WDDDM 2009

Data placement

- Builds on the Chord structured ring overlay network.
- Nodes in the overlay have unique identifiers uniformly picked from the [0,1] interval and ordered along the ring.
- Each node is responsible for the storage of buckets of a distributed hash table (DHT) also mapped into the same [0,1] interval.
- Several data placement strategies defined on a per collection basis.
- Automatic load redistribution on membership changes.
- As some workloads may impair the uniform data distribution the system implements dynamic load-balancing.

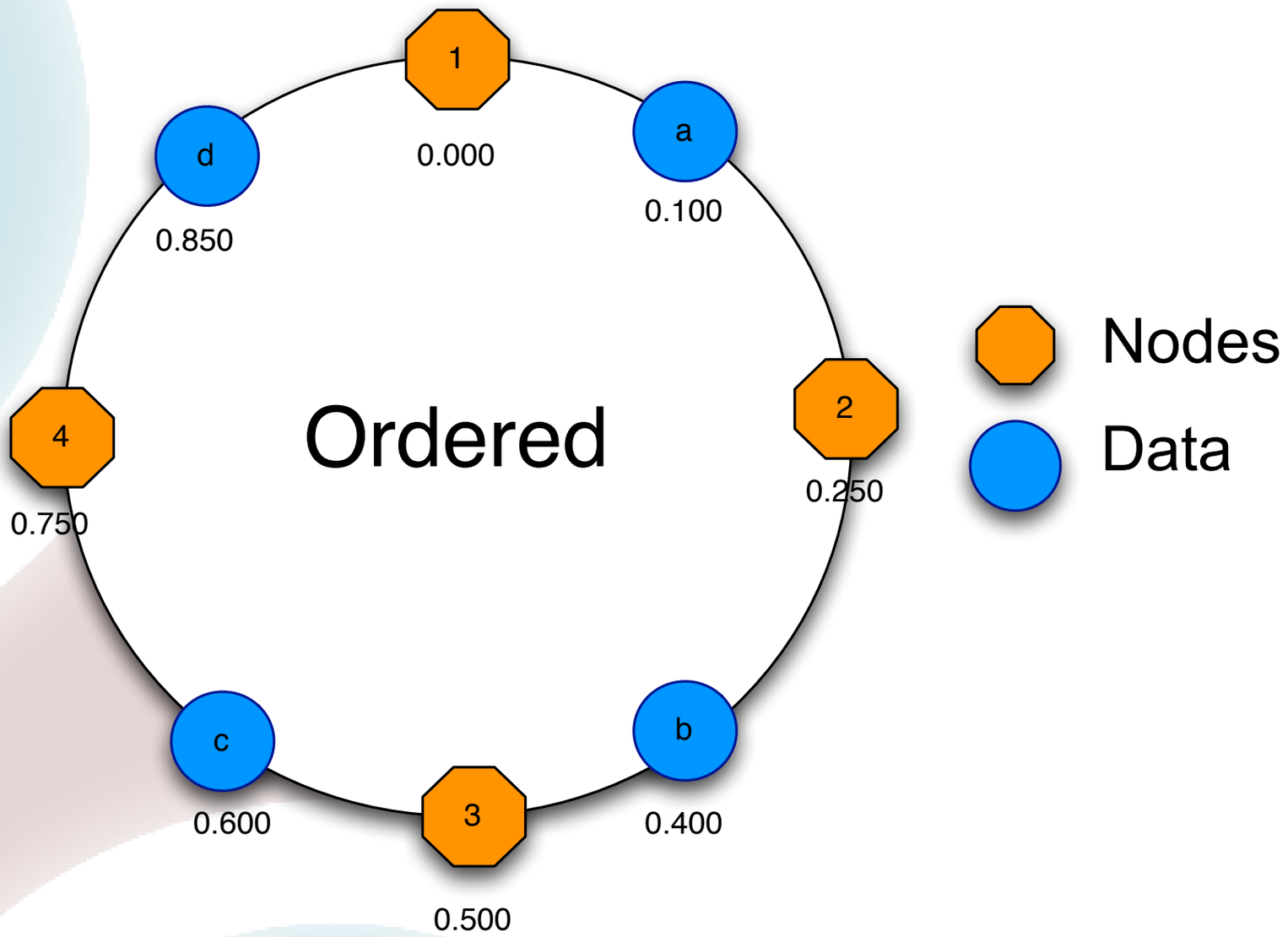
Random

- The random strategy is based on a consistent hash.
- Pseudo-randomly hash the tuple's key.
- Uniformly maps tuples identifiers to the identifier space, providing automatic load balancing.



Ordered

- The ordered strategy places tuples according to the partial order of the tuple'



DataDroplets Simulated evaluation Setting

- 2 Dual-Core AMD Opteron processors running at 2.53GHz and 2GB of RAM.
- Network delay model with latency uniformly distributed between 1 ms and 2 ms to simulate a LAN network.
- Hybrid simulation for CPU profiled with real execution.
- Populated with 10000 concurrent users and the same number of active users were simulated.

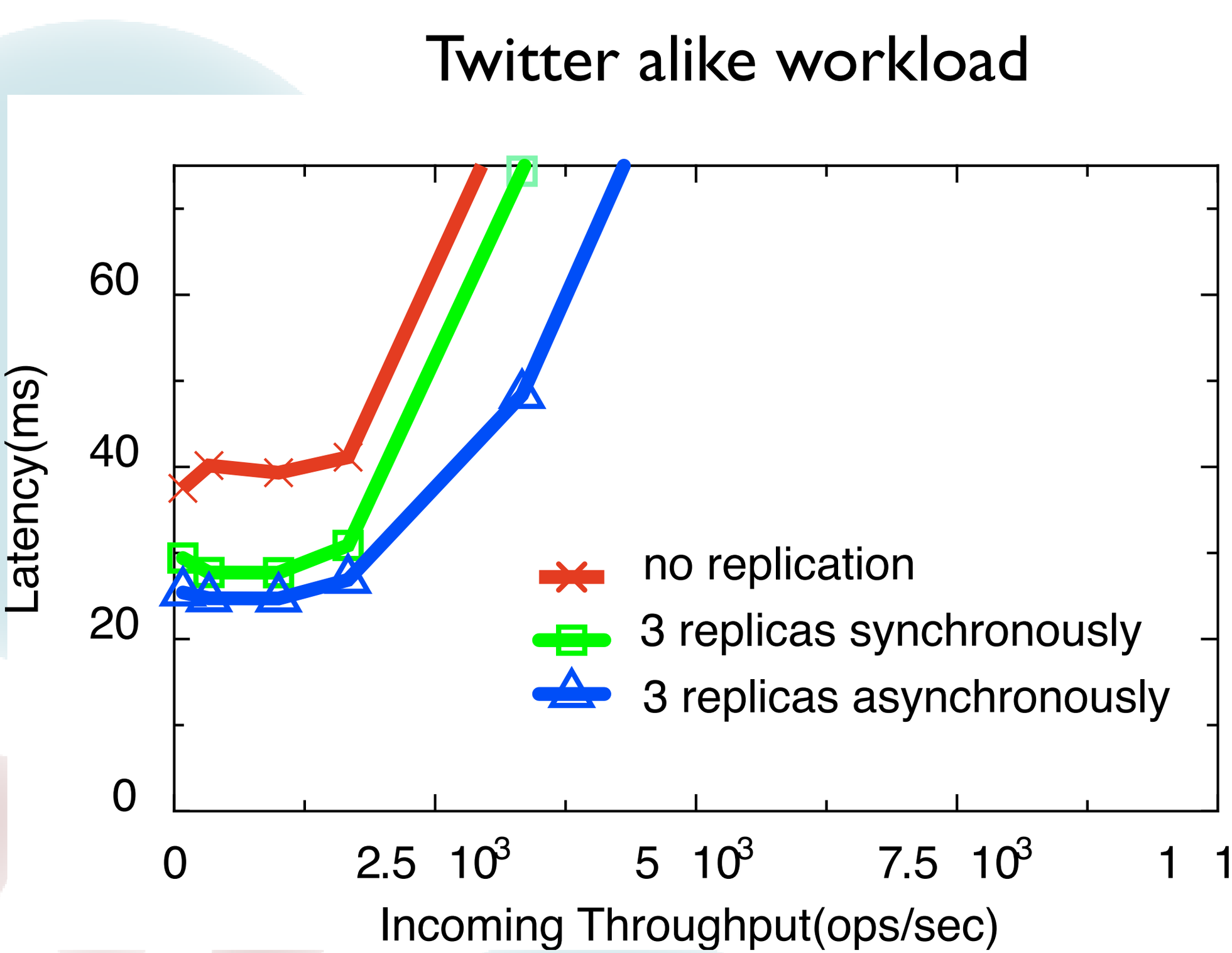
DataDroplets real evaluation setting

- Real
  - 24 AMD Opteron Processor cores running at 2.1GHz, 128GB of RAM.
  - 20 instances of Java Virtual Machine (1.6.0) running ProtoPeer.
  - Apache MINA 1.1.3 for communication.
  - All data persistently stored using Berkeley DB Java edition 4.0.5.
  - Populated with 2500 concurrent users and the same number of active users were

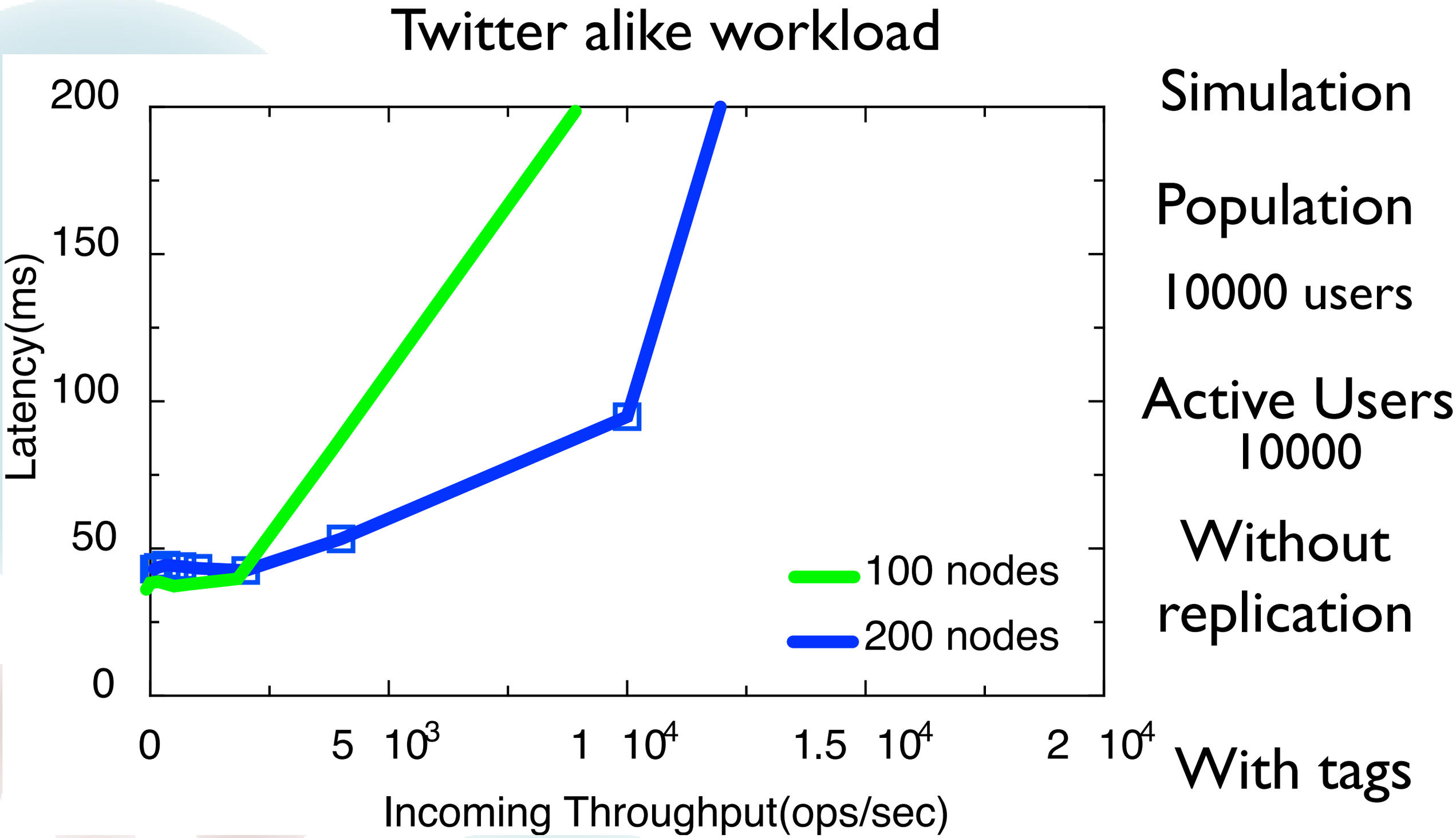
Twitter alike workload

- Workload mimics a Twitter alike application.
- The workload needs three collections to store the needed information: users, tweets and users_timeline.
- Operations
  - List<Tweet>**statuses_user_timeline**(String userID,int start,int count)
  - List<Tweet>**statuses_friends_timeline**(String userID,int start,int count)
  - List<Tweet> **search_contains_hashtag**(String topic)
  - List<Tweet> **statuses_mentions**(String owner)
  - **statuses_update**(Tweet tweet)
  - **friendships_create**(String userID,String toStartUserID)
  - **friendships_destroy**(String userID, String toStopUserID)
- Open-Source

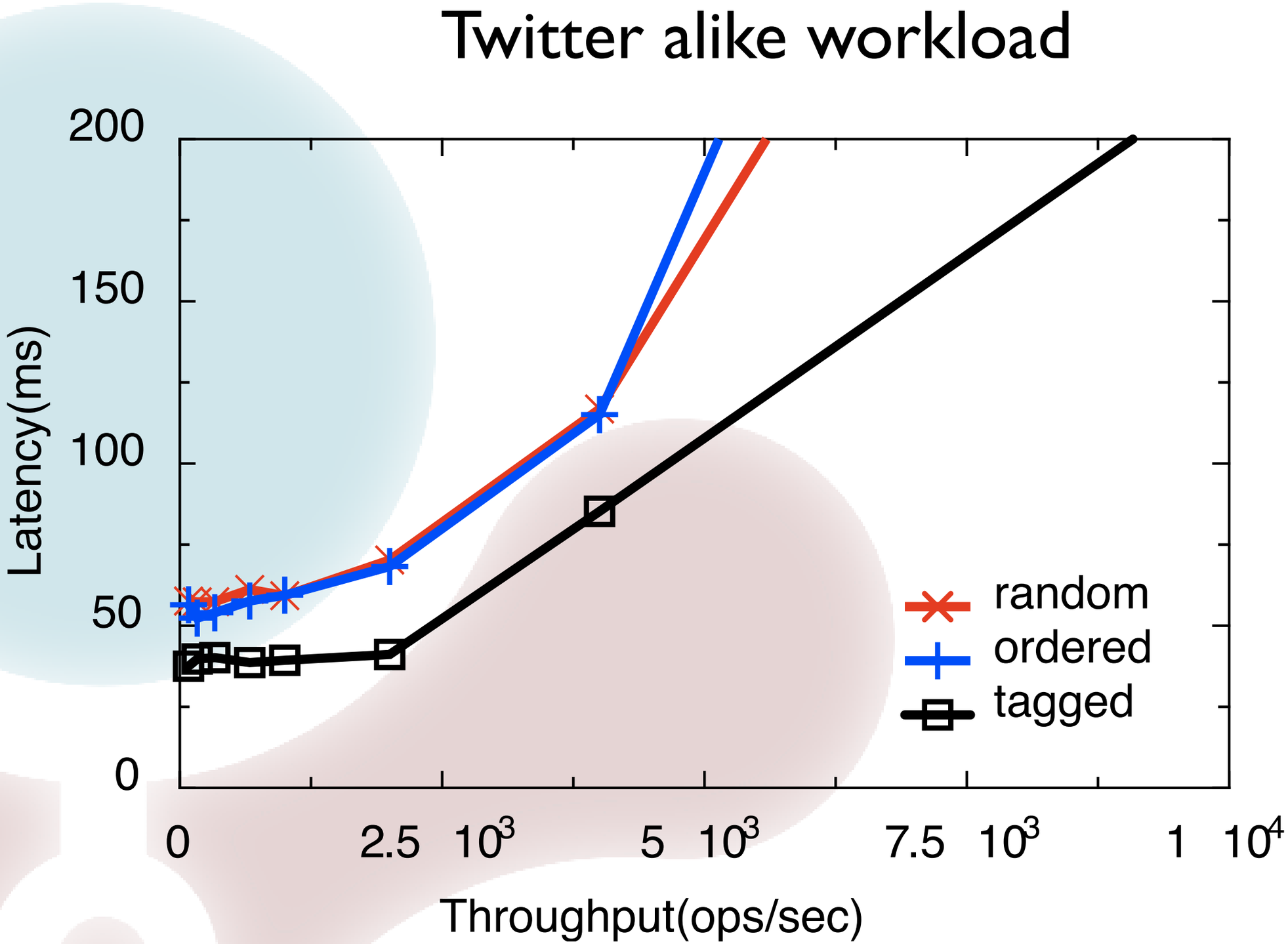
DataDroplets Replication



DataDroplets Scalability



Data placement simulated results



Population

10000 users

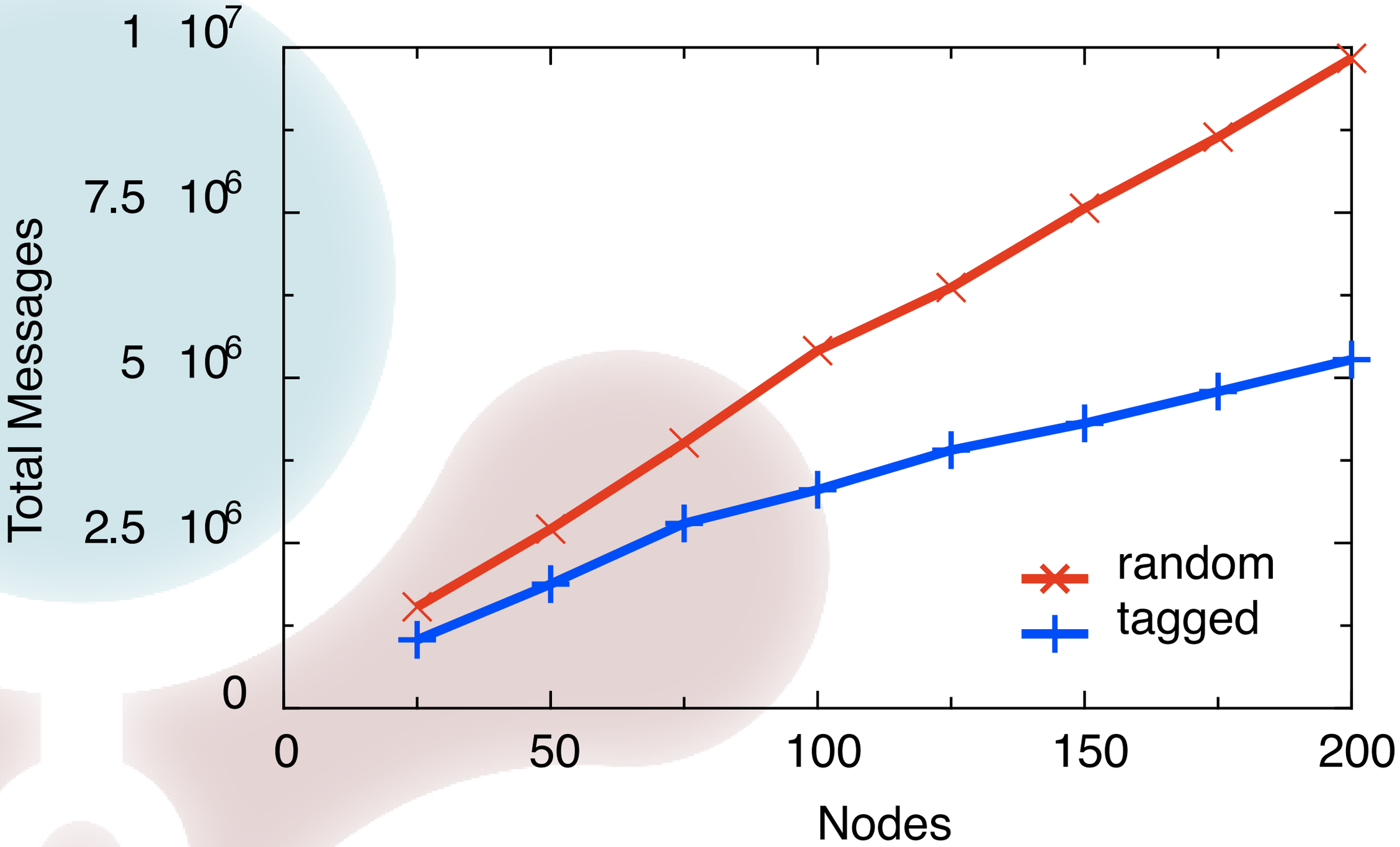
Active Users

10000

100 nodes

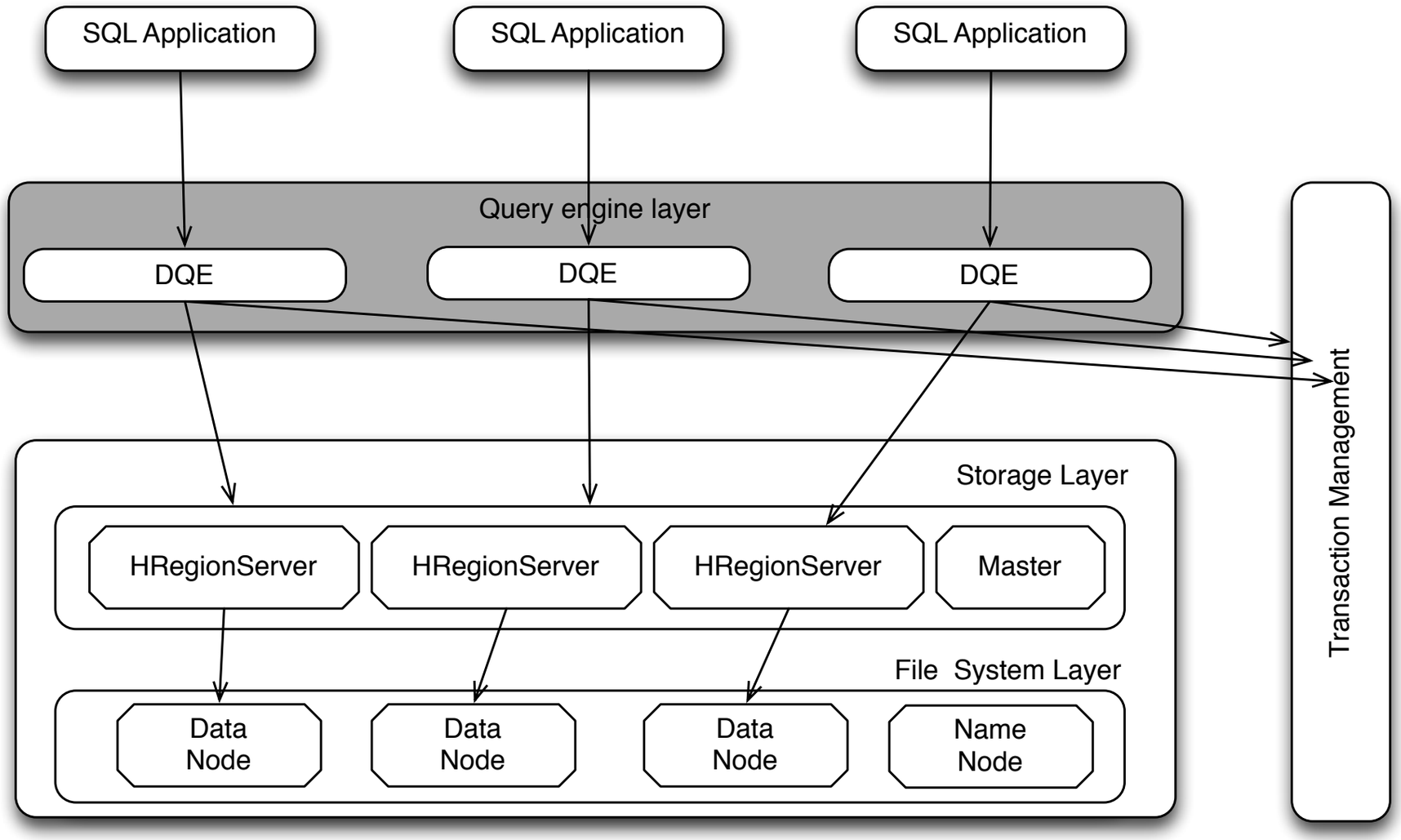
Without replication

Number of exchanged messages



Distributed query engine implementation

- CumuloNimbo FP7 project.
- HBase and Apache Derby



DQE non-transactional evaluation setting

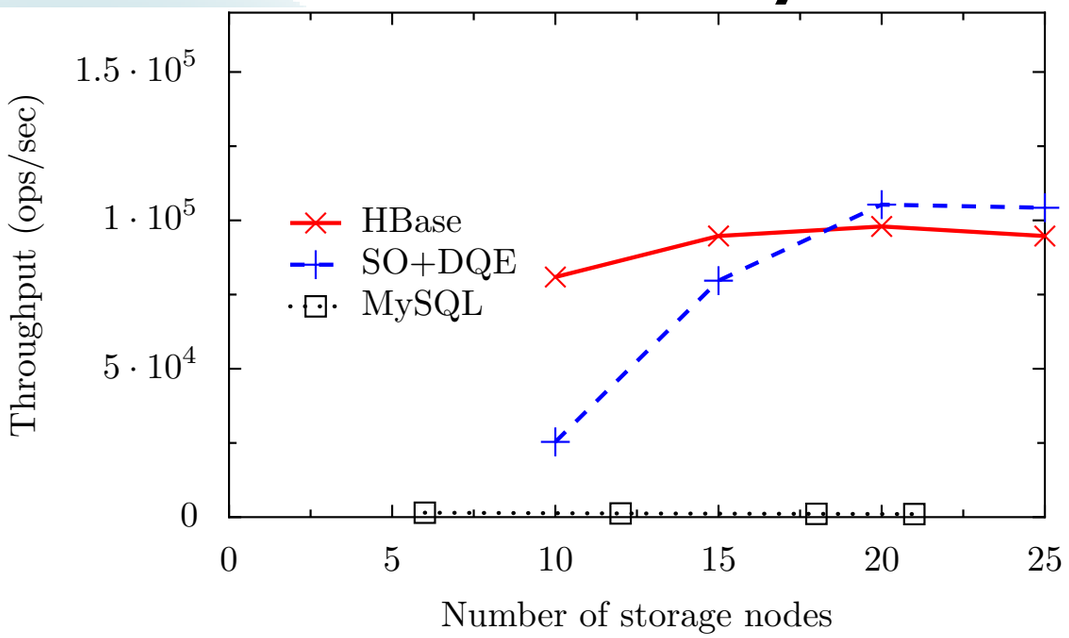
- We ran the experiments on a cluster of forty-two machines with 3.10GHz GHz Quad-Core i3-2100 CPU, with 4GB memory and a local SATA disk.
- Number of client machines from one to ten, each running a hundred and fifty client threads. Each client machine also ran an DQE instance in embedded mode.
- One machine was used to run the HDFS namenode, HBase Master and Zookeper .
- The remaining machines are RegionServers, each configured with a heap of 3GB, and also running a HDFS datanode instance.
- The TPC-C database ranges from five warehouses for a single RegionServer to a hundred and fifty warehouses for thirty RegionServers.

DQE transactional evaluation setting

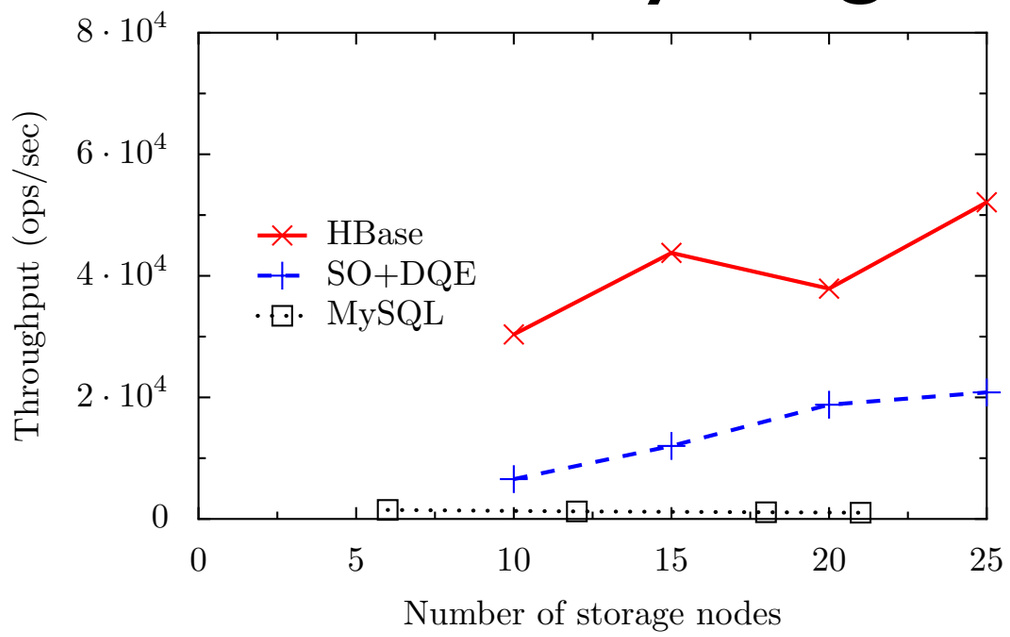
- Cluster of thirty-six machines
- Six of which are used to run the HDFS namenode, HBase Master, the SO server, a Zookeeper ensemble.
- The YCSB workload is run from five machines, each running a hundred client threads.
- read-only, single-row
- write, multi-row write, and and 30% scan
- Remaining machines as RegionServers, each configured with a heap of 16GB.
- Each machine has two Xeon Quad-Core 2.40Ghz processors, 24GB of memory, gigabit Ethernet and four SATA hard disks.
- MySQL is configured with a single management node and a single MySQL daemon.
- Redundancy of 3.

DQE transactional scalability

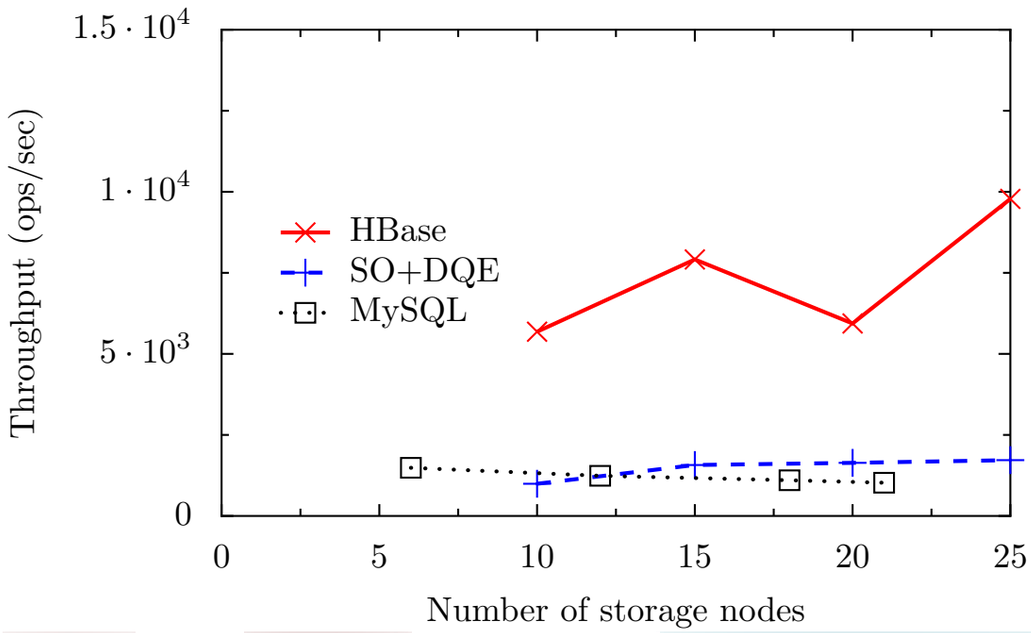
Read-Only



Write-Only single



Write-Only multi



Mixed