A correlation-aware data placement strategy for key-value stores

Ricardo Vilaça, Rui Oliveira, and José Pereira
High-Assurance Software Laboratory
Universidade do Minho
Portugal

Large scale tuple stores

- Scalable, elastic and dependable systems.
- Amazon's Dynamo, Yahoo's PNUTS, Google's Bigtable, Facebook's Cassandra
- Solve internal data management problems and support their current or future Cloud services.
- A simple tuple store interface, that allows applications to insert, query, and remove individual elements.
- Only single tuple operations or at most range operations over tuples.

Motivation

- Data placement strategies in existing distributed tuple stores (Bigtable,PNUTS,Dynamo,Cassandra) only efficiently support single tuple or range queries.
- Most applications have however general multi-tuple operations that request reads and/or writes to a specific subset of tuples.
- Performance of multi-tuple operations is highly affected by the data placement strategy.
- The probability of a pair of tuples being requested together in a query is not uniform but often highly skewed.
- Correlation is mostly stable over time for real applications.
- How to achieve such placement in a decentralized fashion?

Data Droplets

- Offers a simple application interface providing the atomic manipulation of tuples and the flexible establishment of arbitrary relations among tuples.
- Multi-tuple operations leverage disclosed data relations to manipulate sets of comparable or arbitrarily related elements.
- Provides additional consistency guarantees.
- Ease the migration from current RDBMS.

Data Model

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

- Each tuple is a triple consisting of a unique key drawn from some partially ordered set, a value that is opaque and a set of tags.
- Tuples can be organized in collections.

API

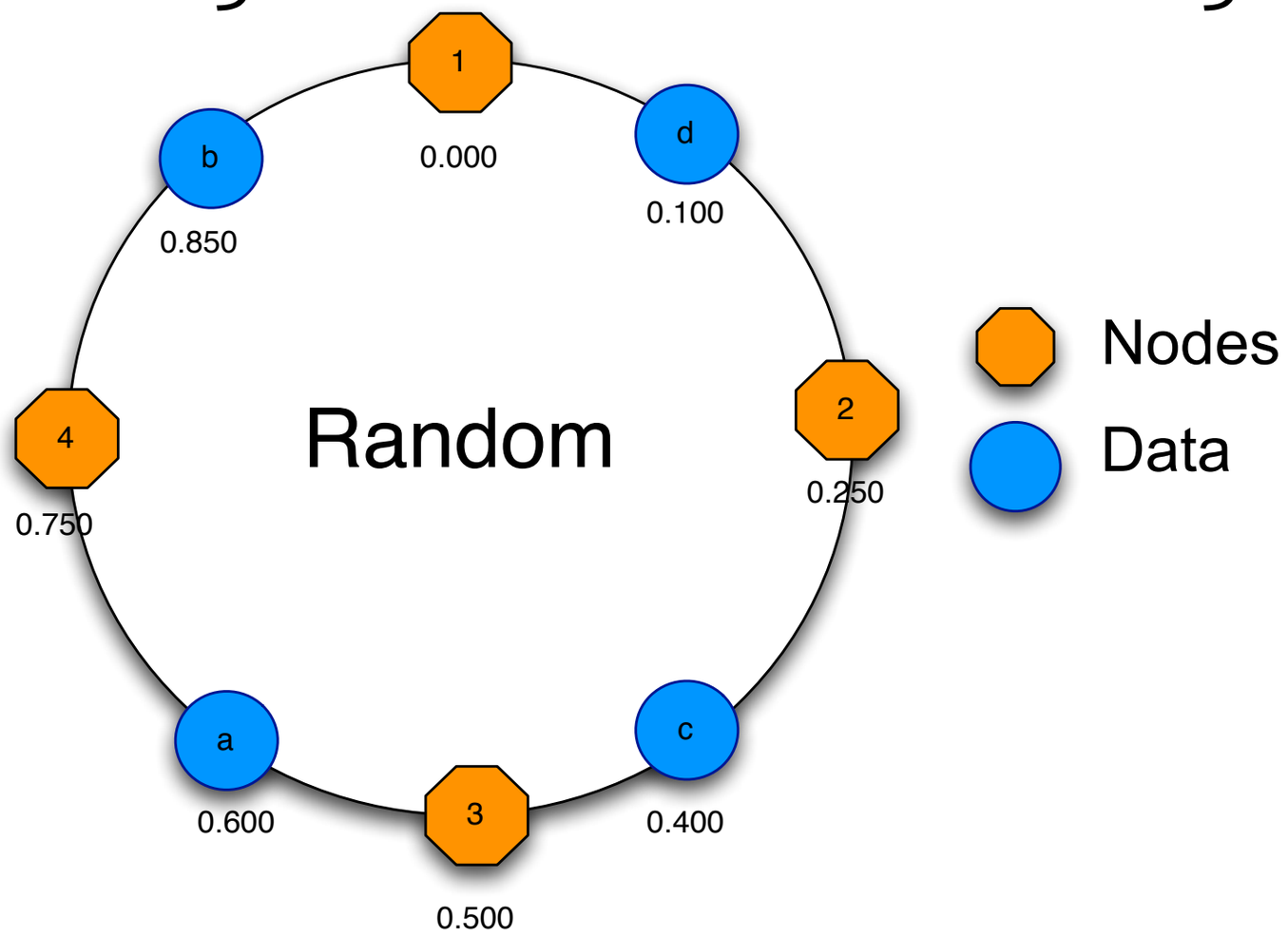
- **put**(String collection,K key, V value, $\mathcal{P}$String tags)
- V **get**(String collection,K key)
- V **delete**(String collection,K key)
- **multiPut**(K $\rightharpoonup$ (V $\times$ $\mathcal{P}$String) mapItems)
- K $\rightharpoonup$ V **multiGet**($\mathcal{P}$(String $\times$ K) keys)
- K $\rightharpoonup$ V **getByRange**(K min, K max)
- K $\rightharpoonup$ V **getByTags**($\mathcal{P}$String tags)

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.

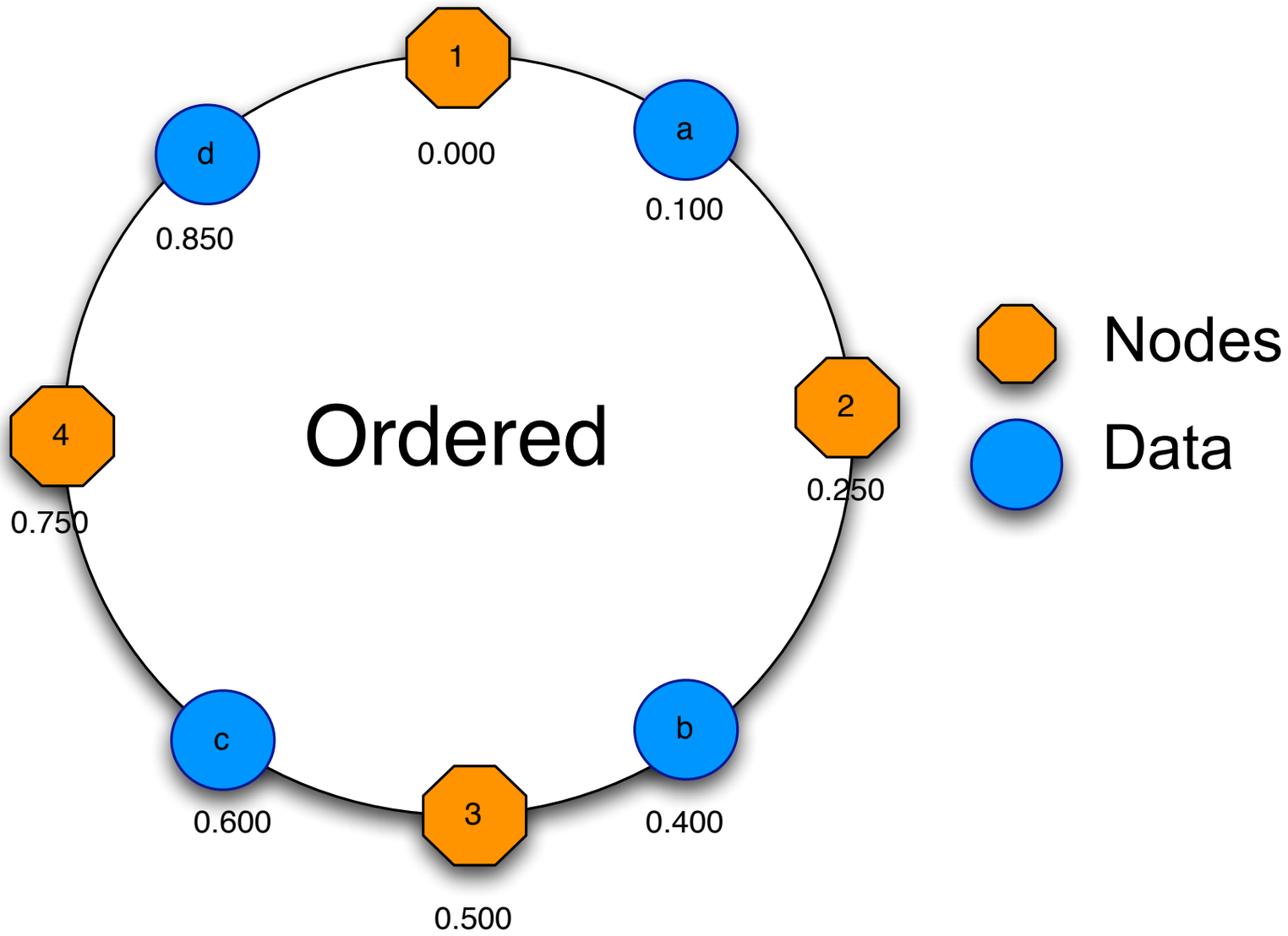
Random

- 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

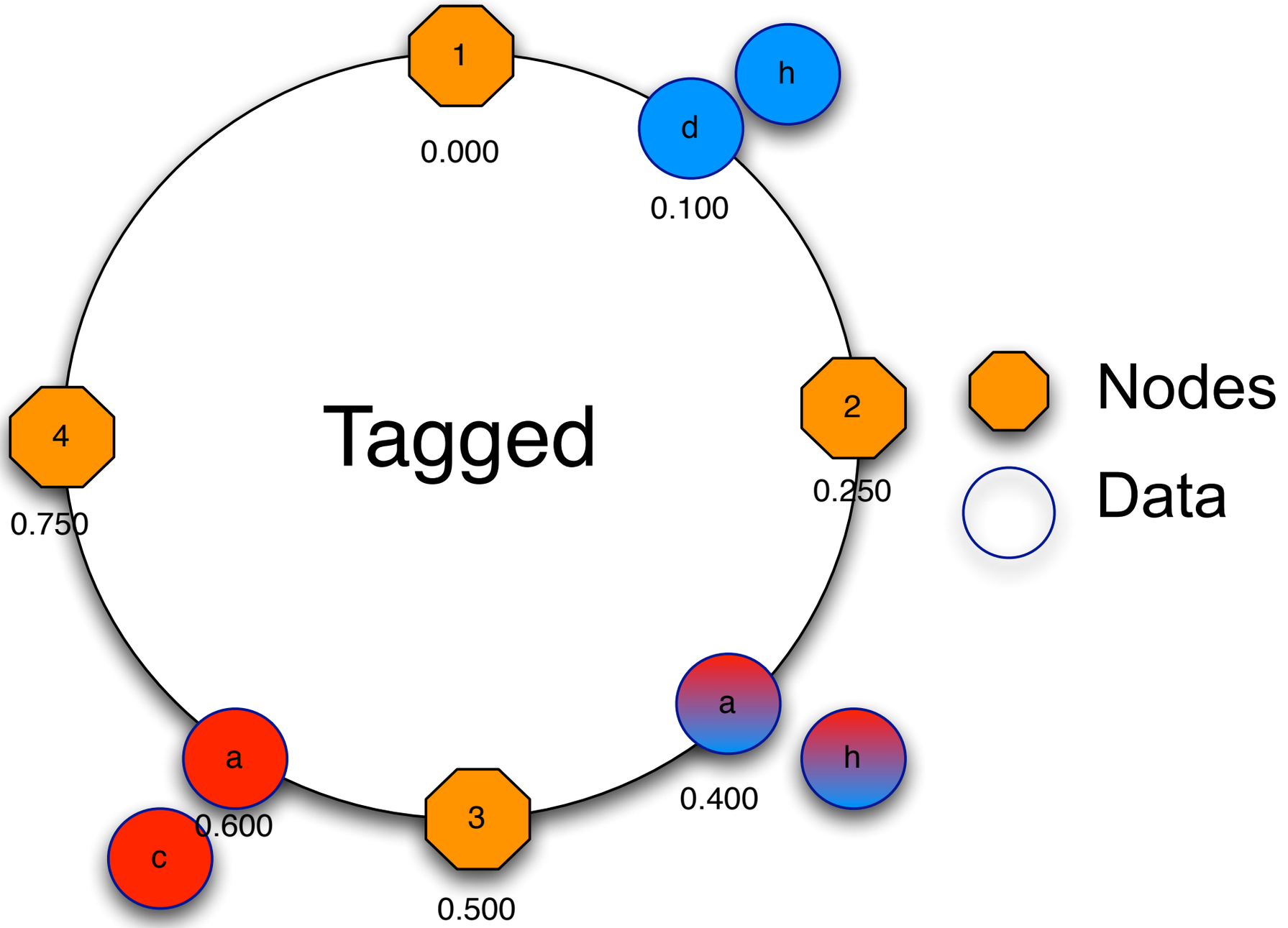
- The ordered strategy places tuples according to the partial order of the tuple' keys.



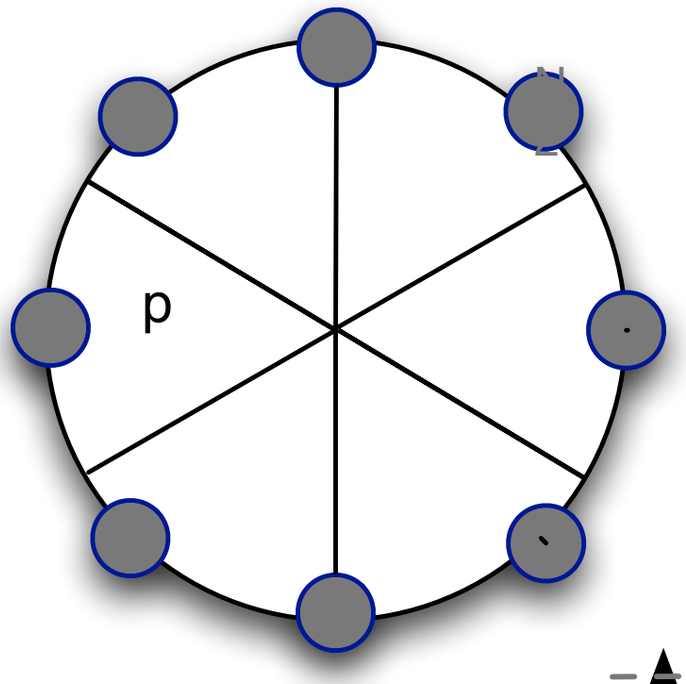
Our approach: Tagged

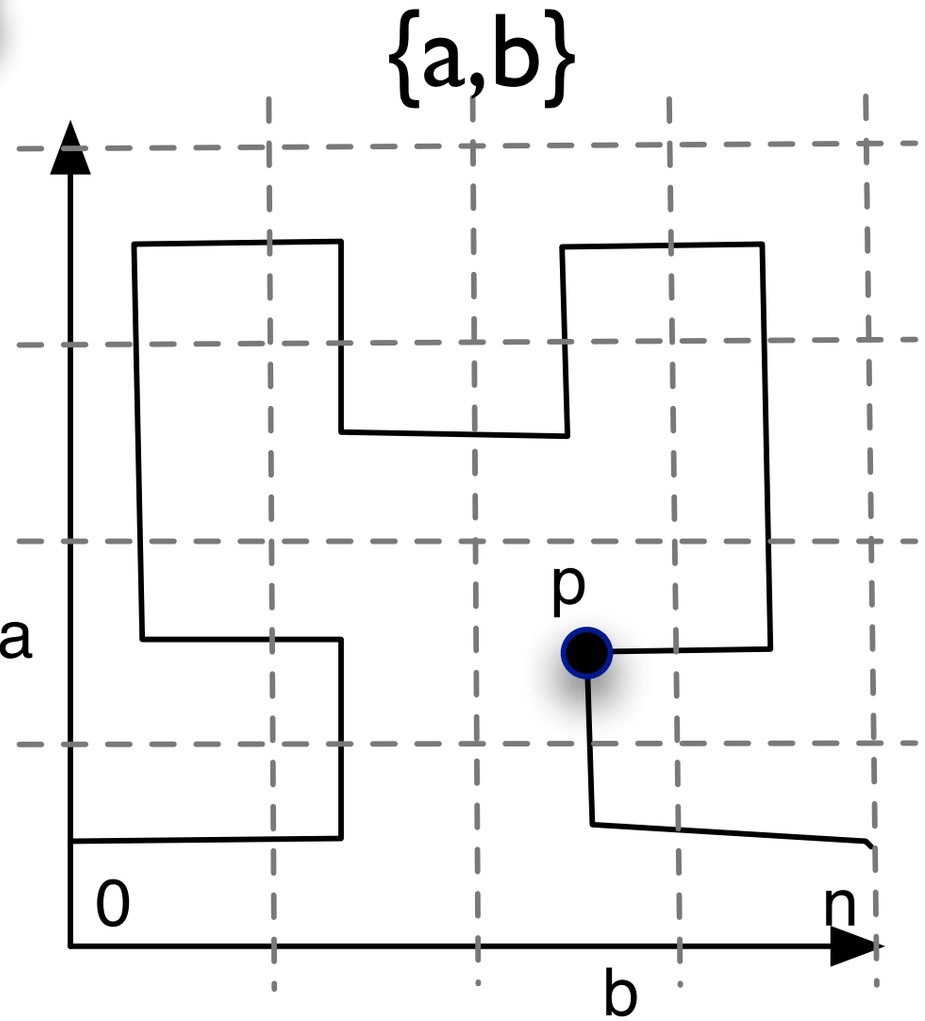
- Set of tags defined per tuple.
- Uses a dimension reducing and locality-preserving indexing scheme that maps the multidimensional information space to the identifier space.
- Keeps the minimal precision needed for the multidimensional information space.

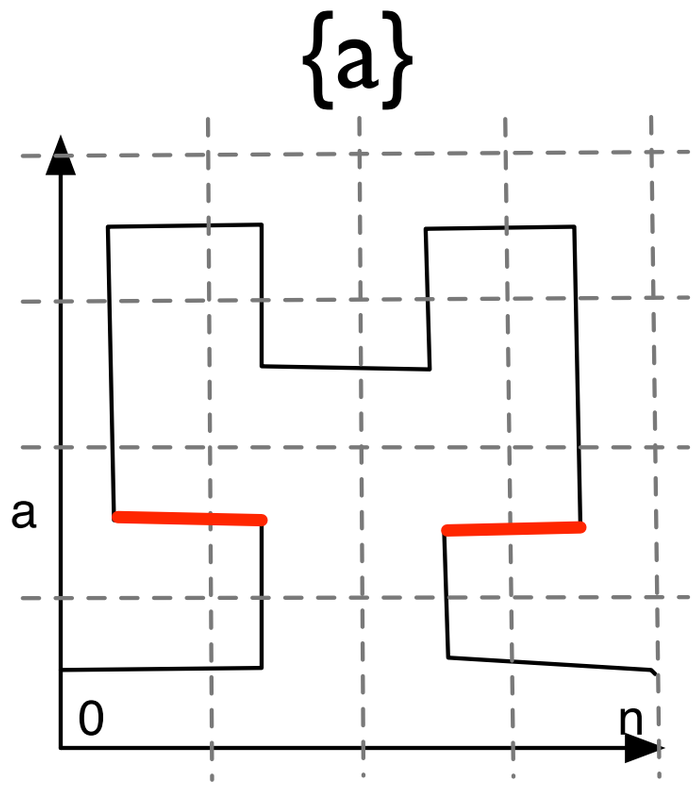
Our approach: Tagged



Our approach: Tagged







Our approach: Tagged

- Problem
  - All tuples with equal set of tags have equal identifiers.
  - Not fixed by load-balancing.
- Solution
  - Higher order n bits from tagged strategy and remaining from other strategy (random or ordered).

Evaluation Setting

- Simulated
  - 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.

Evaluation Setting

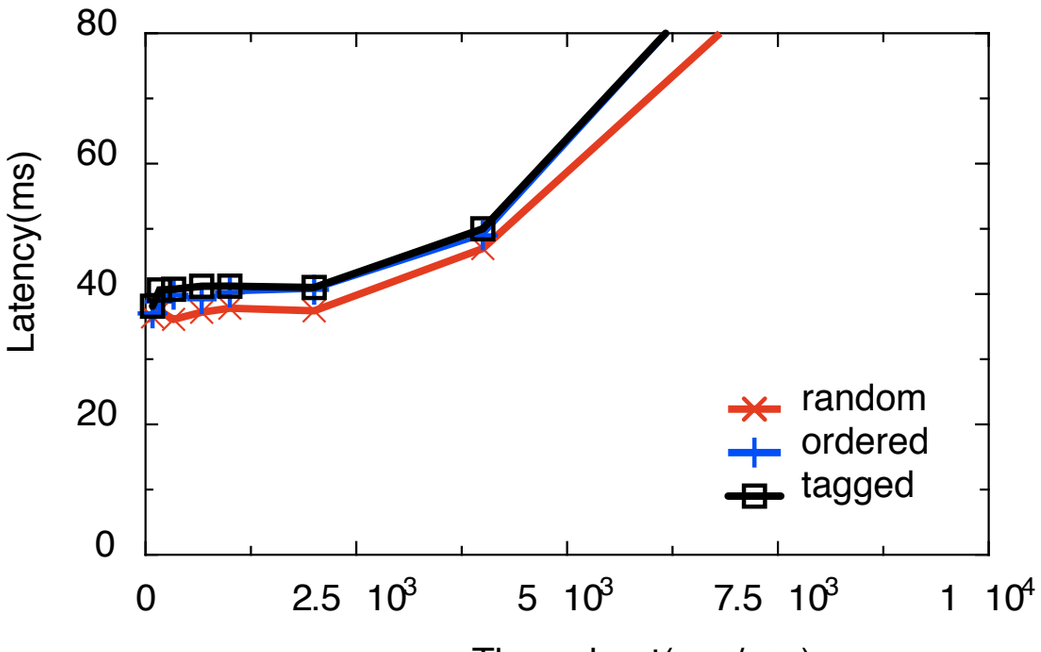
- Real
  - 24 AMD Opteron Processor cores running at 2.1GHz, 128GB of RAM.
  - 20 instances of Java Virtual Machine (1.6.0) running ProtoPeer.
  - Apache MINA 1.1.3 for communication.
  - All data persistently stored using Berkeley DB Java edition 4.0.5.
  - Populated with 2500 concurrent users and the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

Workload

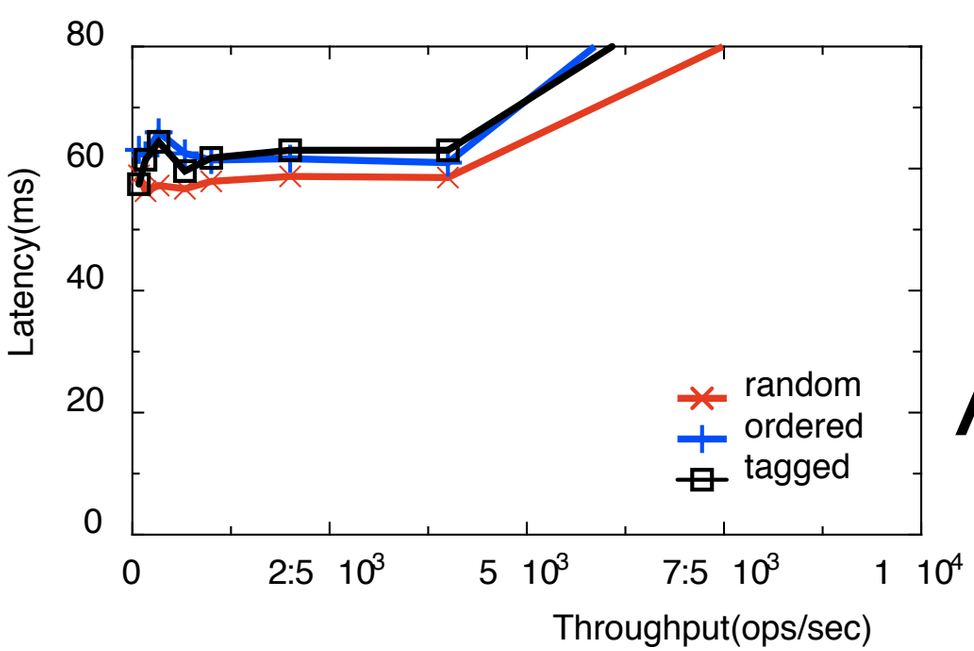
- Workload mimics a Twitter alike application.
- The workload needs three collections to store the needed information: users, tweets and users_timeline.
- Operations
  - List<Tweet>**statuses_user_timeline**(String userID,int start,int count)
  - List<Tweet>**statuses_friends_timeline**(String userID,int 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)

Results write operations

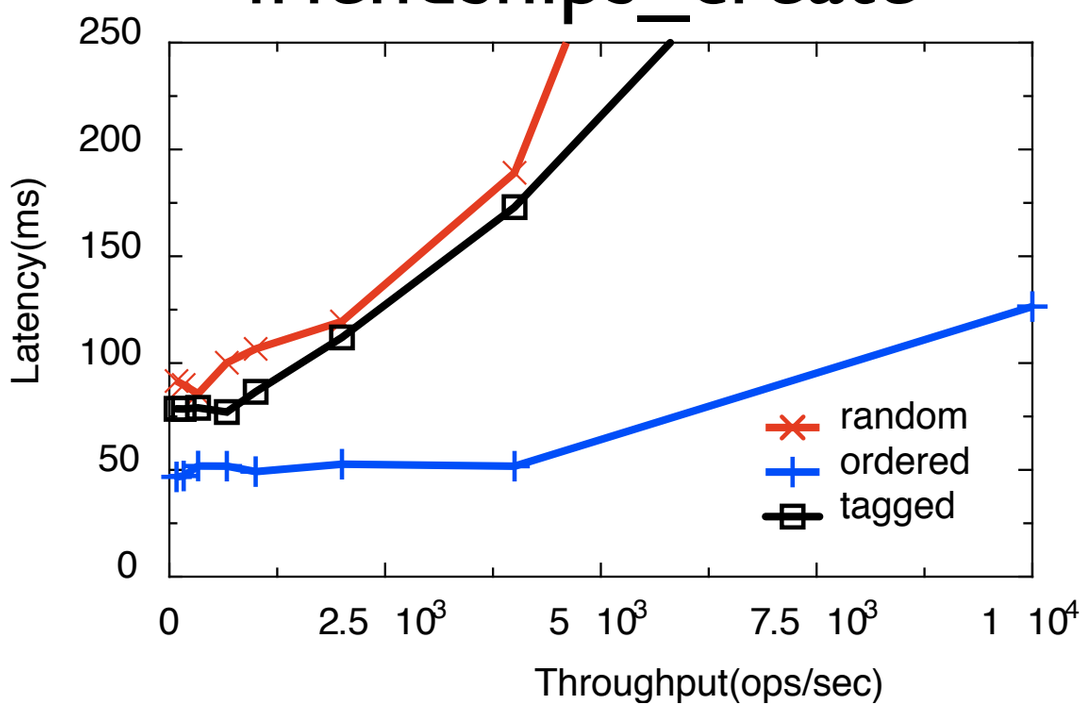
statuses_udpate



friendships_destroy



friendships_create



Population

10000 users

Active Users

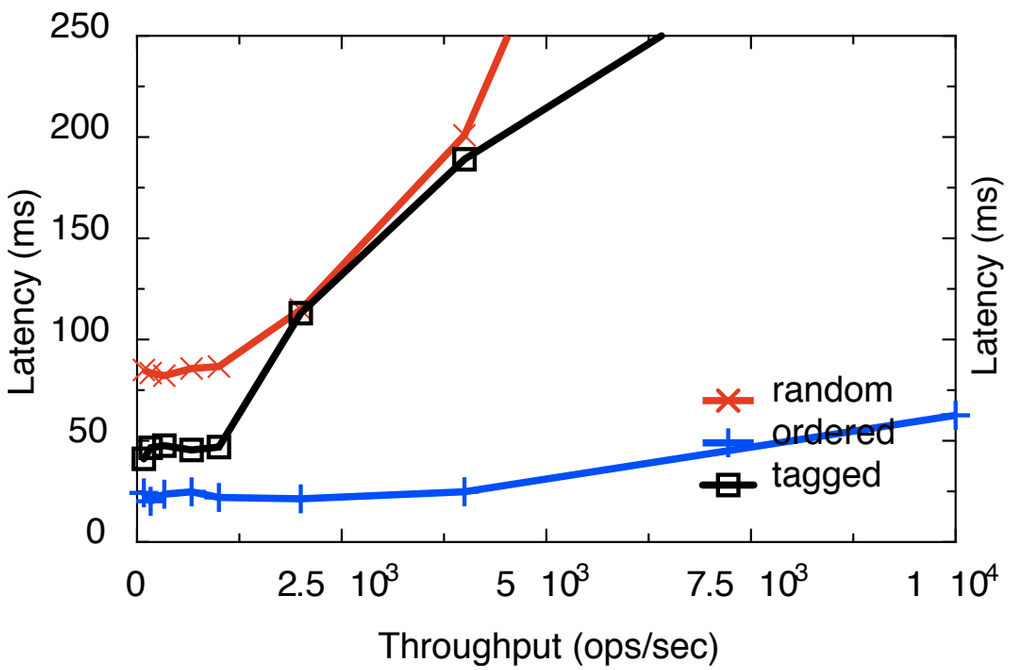
10000

100 nodes

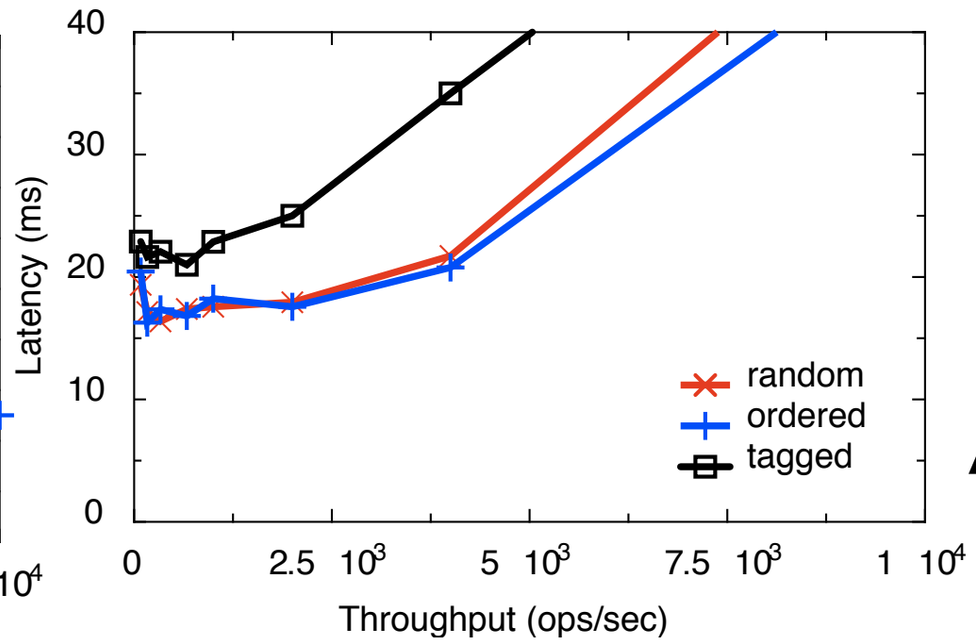
Without replication

Results read-only operations

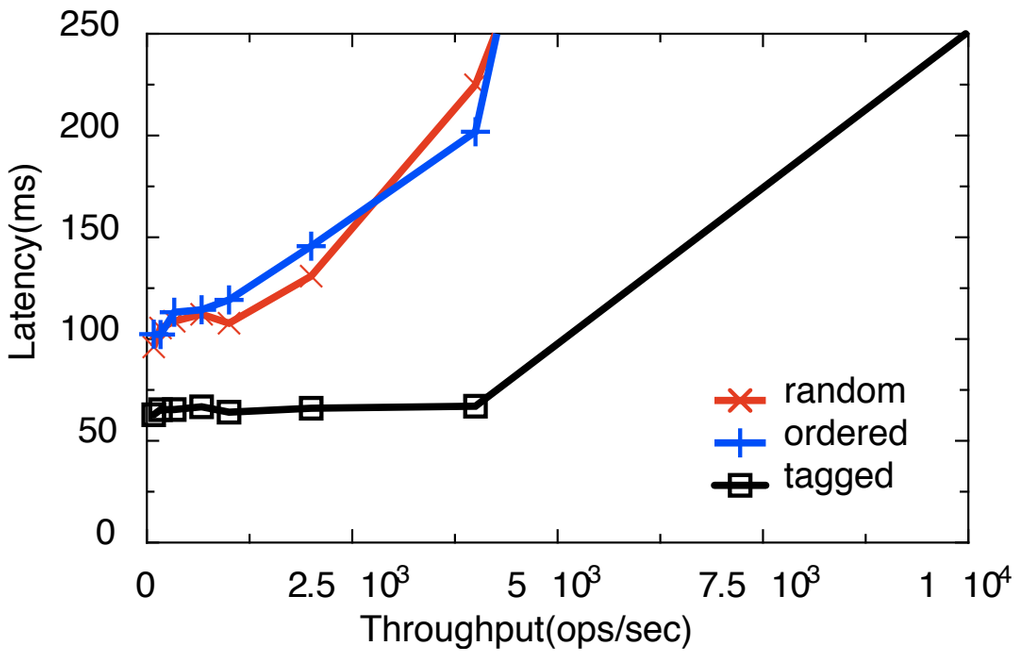
statuses_user_timeline



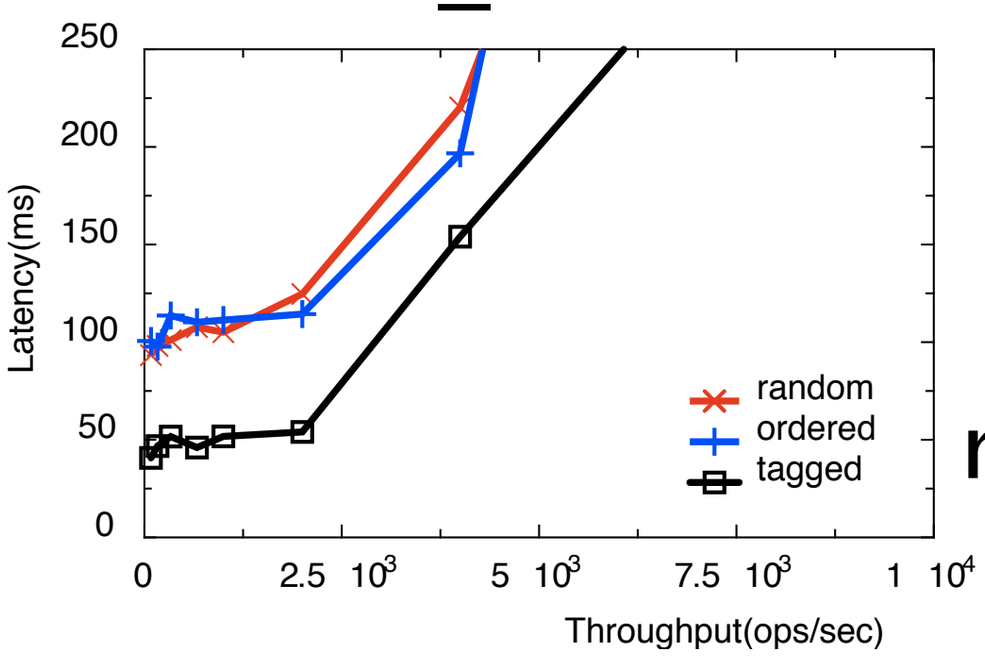
statuses_friends_timeline



search_contains_hashtag



status_mentions



Population

10000 users

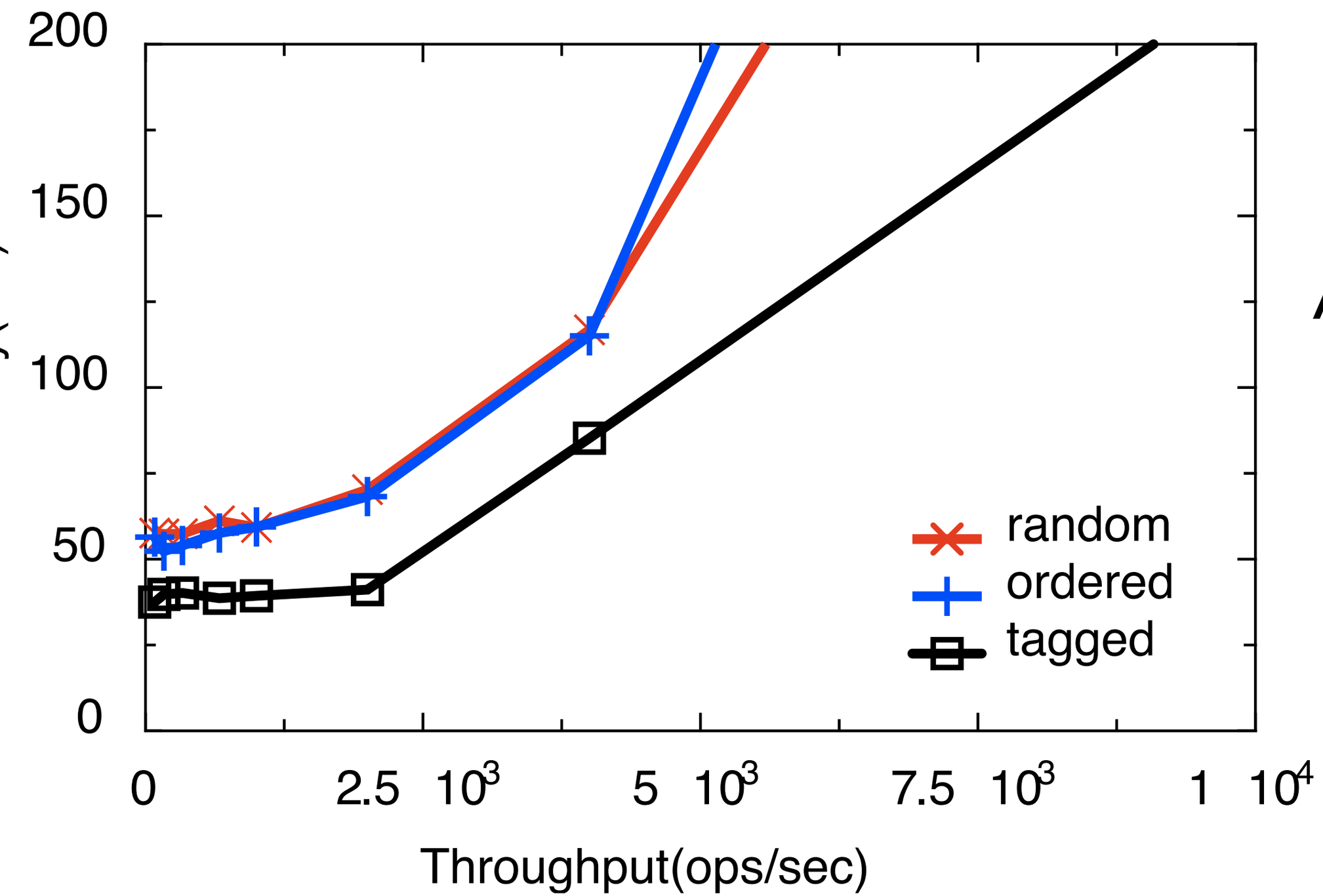
Active Users

10000

100 nodes

Without replication

Overall simulated results



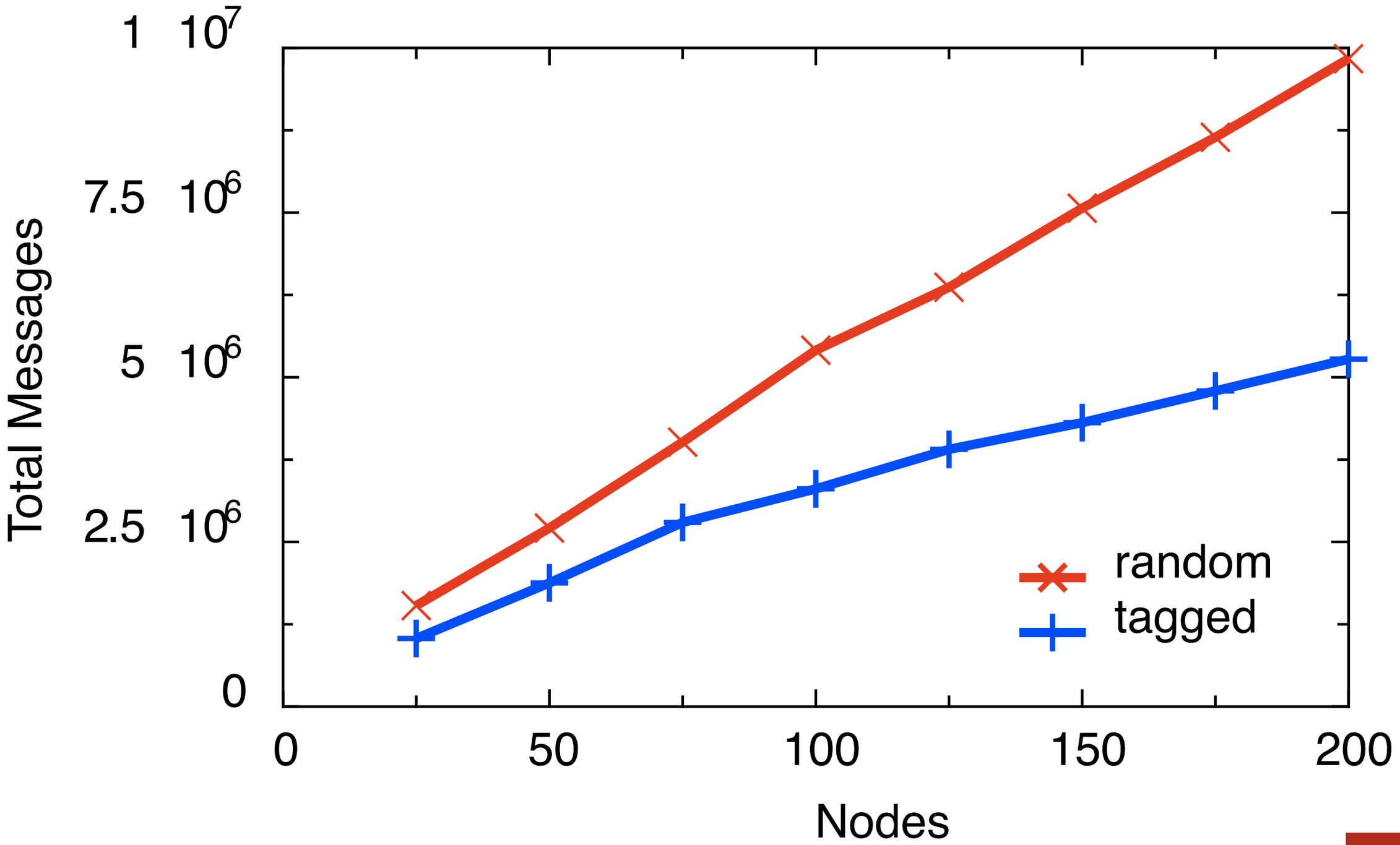
Population
10000 users

Active Users
10000

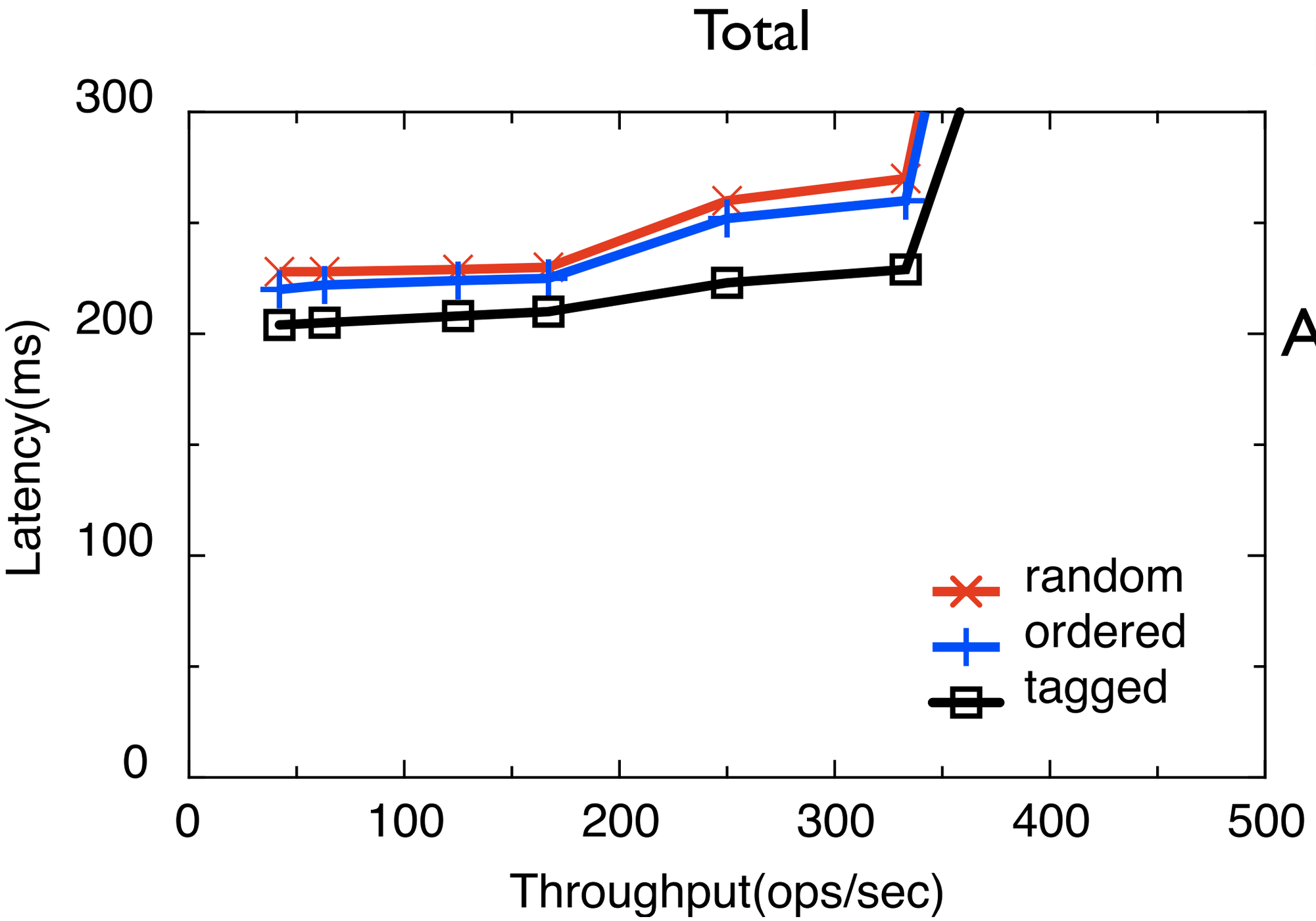
100
nodes

Without
replication

Number of exchanged messages



Overall real results



Population

2500 users

Active Users

2500

20 nodes

Without replication

Conclusion

- A novel data placement strategy based on multidimensional locality preserving mappings.
- Fits access patterns found in many current applications, which arbitrarily relate and search data by means of free-form tags.
- Provides better results in overall query performance.
- Usefulness of having multiple simultaneous placement strategies in a multi-tenant system.
- A simple but realistic benchmark for elastic key-value stores based on Twitter and currently known statistical data about its usage.