Evaluation of NoSQL databases for large-scale decentralized microblogging

Cassandra & Couchbase

Alexandre Fonseca, Anh Thu Vu, Peter Grman

Decentralized Systems - 2nd semester 2012/2013
Universitat Politècnica de Catalunya

Microblogging

- Blogging with very small & concise posts.
- Users subscribe to others by following them.
- Real-time interactions between growing userbase.
- Useful for:
    - Real-time updates & news.
    - Marketing and public relations campaigns.
    - Research into social interaction & perception.
- Examples:
    - Twitter, Tumblr, identi.ca,
    - Facebook & Google+ status updates.

Microblogging - decentralization

- Twitter:
  - Over 500 million registered users.
  - 340 million tweets per day.
  - More than 12TB of data per day in 2010.

- Hard to provide this service relying solely on a small number of centralized servers.
  - Scaling up has its limits.

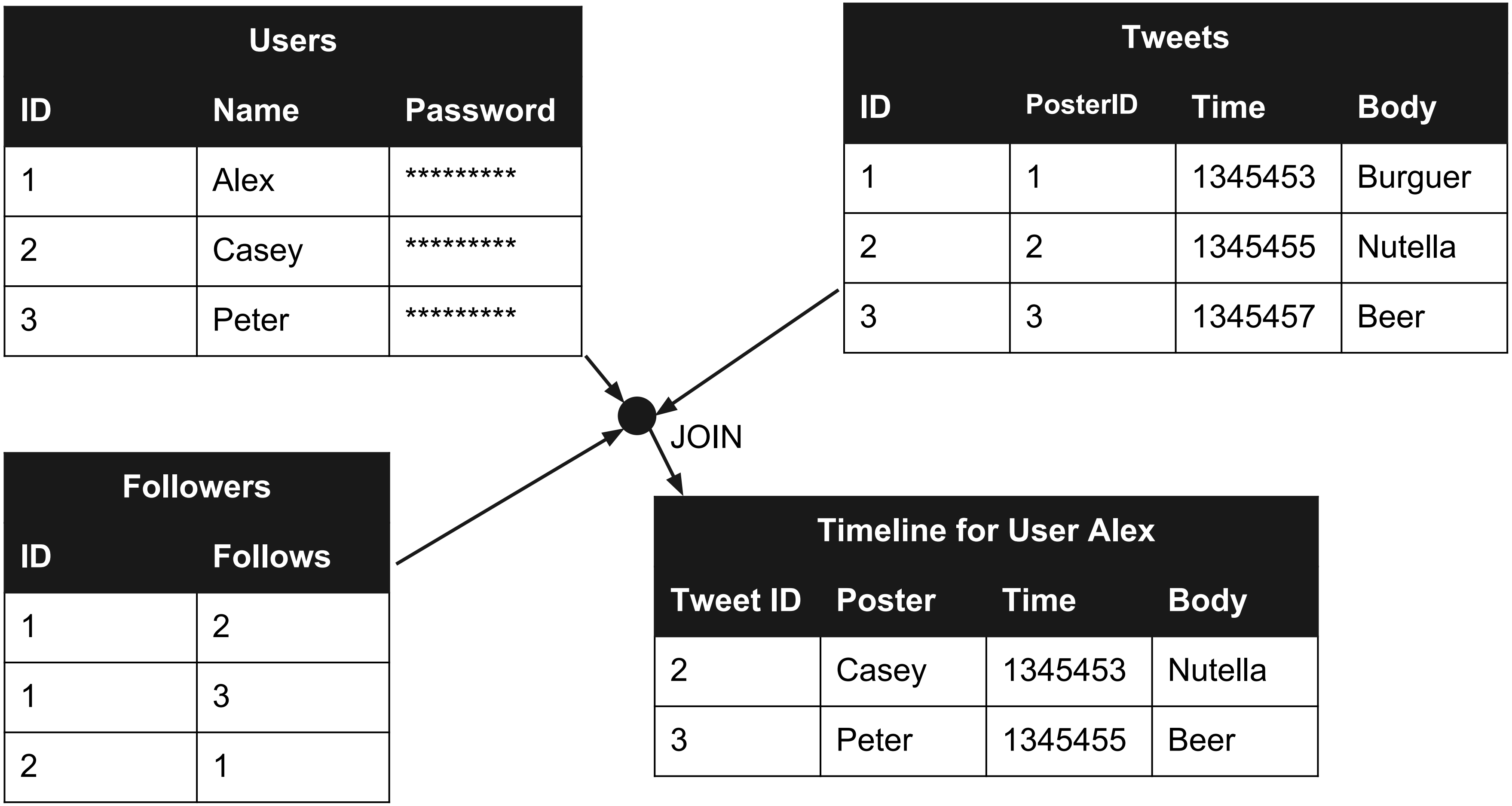
Microblogging - decentralization

- 2 possible ways to decentralize:
  - Decentralize dedicated servers managed by the service provider - better load sharing.
  - Allow users to share the system load between them (similar to voluntary computing).

Relational Database Systems (RDS)

- Typical enterprise and service data storage.
- Tables and table rows as storage units.
- Highly normalized.
- Storage divided into entity types.
    - One table per entity type.
    - One row per entity instance.
    - Many-to-many relationship tables.
- Transaction support for maximum consistency.

RDS - Twitter Example



Users		
ID	**Name**	**Password**
1	Alex	*********
2	Casey	*********
3	Peter	*********

Tweets			
ID	**PosterID**	**Time**	**Body**
1	1	1345453	Burguer
2	2	1345455	Nutella
3	3	1345457	Beer

Followers	
ID	**Follows**
1	2
1	3
2	1

Timeline for User Alex			
Tweet ID	**Poster**	**Time**	**Body**
2	Casey	1345453	Nutella
3	Peter	1345455	Beer

NoSQL

- New (or is it?) data storage paradigm:
  - KISS.
  - Simple key-value stores in its essence.
  - (Usually) no transaction control or strict schemas.
  - Incentive denormalization.
  - Let the application worry about business logic and entity relationships.
  - Focus on:
    - Flexibility (of schemas, topologies, configurations)
    - Scalability
    - Robustness
    - Performance

NoSQL - Twitter Example

Users

```
[{'id': 1, 'name': 'Alex',
'password': '***', 'following':[2, 3]},
{'id': 2, 'name': 'Casey',
'password': '***', 'following':[1]},
{'id': 3, 'name': 'Peter',
'password': '***', 'following':[1, 2]}]
```

Tweets

```
[{'id': 1, 'posterID': 1,
'time': 1345453, body: 'Burger'},
{'id': 2, 'posterID': 2,
'time': 1345455, body: 'Nutella'},
{'id': 3, 'posterID': 3,
'time': 1345457, body: 'Beer'}]
```

Timeline-View

```
[{'user': 1, 'tweetID': 2, 'postedBy': 2, tweetBody: 'Nutella'},
{'user': 1, 'tweetID': 3, 'postedBy': 3, tweetBody: 'Beer'},
{'user': 2, 'tweetID': 1, 'postedBy': 1, tweetBody: 'Burger'},
{'user': 3, 'tweetID': 1, 'postedBy': 1, tweetBody: 'Burger'},
{'user': 3, 'tweetID': 2, 'postedBy': 2, tweetBody: 'Nutella'}]
```

NoSQL - Twitter Example

Users

```
[{'id': 1, 'name': 'Alex',
'password': '***', 'followers':[2, 3]},
{'id': 2, 'name': 'Casey',
'password': '***', 'followers':[1, 3]},
{'id': 3, 'name': 'Peter',
'password': '***', 'followers':[1]}]
```

Tweets

```
[{'id': 1, 'posterID': 1,
'time': 1345453, body: 'Burger'},
{'id': 2, 'posterID': 2,
'time': 1345455, body: 'Nutella'},
{'id': 3, 'posterID': 3,
'time': 1345457, body: 'Beer'}]
```

Timeline-View

```
[{'user': 1, 'tweetID': 2, 'postedBy': 2, tweetBody: 'Nutella'},
{'user': 1, 'tweetID': 3, 'postedBy': 3, tweetBody: 'Beer'},
{'user': 2, 'tweetID': 1, 'postedBy': 1, tweetBody: 'Burger'},
{'user': 3, 'tweetID': 1, 'postedBy': 1, tweetBody: 'Burger'},
{'user': 3, 'tweetID': 2, 'postedBy': 2, tweetBody: 'Nutella'}]
```

Cassandra

- Initially developed by Facebook.
- Released in 2008.
- Apache Top-Project by 2010.
- Latest stable release (1.2.4, April 2013)
- Data model:
  - Hybrid between key-value and tabular storage.
  - Data split in column families (like RDS tables).
  - Column families split into rows (indexed by row keys).
  - Dynamic columns (schema).
- Interface: CQL via Thrift.

Cassandra

- All database nodes play the same role.
- Nodes divided into variable number of virtual nodes:
    - Organized into ring structure.
    - Assigned tokens determine data assignment:
        - Consistent hashing of partition key (1st column of primary key).
        - Hashing can be parametrized.
- Cluster can be given topology awareness:
    - Try to keep request resolution local to rack or datacenter.
    - Replicate data over different data centers and racks.
- Replication and consistency options all configurable.

Couchbase

=
CouchDB
relax
+

Current release v2.0.1 (Apr 9, 2013)

Couchbase

- Key - Value
  - Key: string, no space allowed, 250 bytes limit
  - Value: JSON document, integer or serializable byte stream, 20MB limit
- View:
  - MapReduce: to index all data
  - Incrementally and periodically updated

Couchbase

- All nodes play the same role
- A bucket (or database) is
  - divided into small subsets, called vBuckets
  - distributed in the cluster.
- Replication factor:
  - configurable for each bucket with a max of 3.
- Consistency:
  - Strong consistency for access with "key"
  - Eventual consistency for queries on views

PyDLoader

- Custom Python script
- 2 components:
  - Manager:
    - Interactive console.
    - Deploy new nodes.
    - Install and control slaves on those nodes.
  - Slaves:
    - Automatic setup, populating and takedown of database clusters (database slaves).
    - Generation of application workload towards database clusters (workload slaves).
- Used libraries: Boto (AWS), Paramiko (SSH), RpyC (RPC).

Evaluation

- Done using Amazon Web Services (AWS).
- 6 database nodes (m1.small)
  - 1.7 GB of RAM
  - 1 compute unit
  - 150GB of storage
- 12 workload nodes (t1.micro)
  - 615 MB or RAM
  - 1 compute unit
  - No local storage (NAS)
- Distributed equally over 2 datacenters in Ireland.

Evaluation

- Focus on 3 main operations:
  - Tweet
  - Userline (all tweets by user)
  - Timeline (all tweets by people user is following).
- Limit to 50 items per userline/timeline.
- Basic data structure:
  - Aggregate data according to operations, not entities:
    - Timeline table/bucket:
      - UserID, TweetID, PostedBy, Body
    - Userline table/bucket:
      - UserID, TweetID, Body

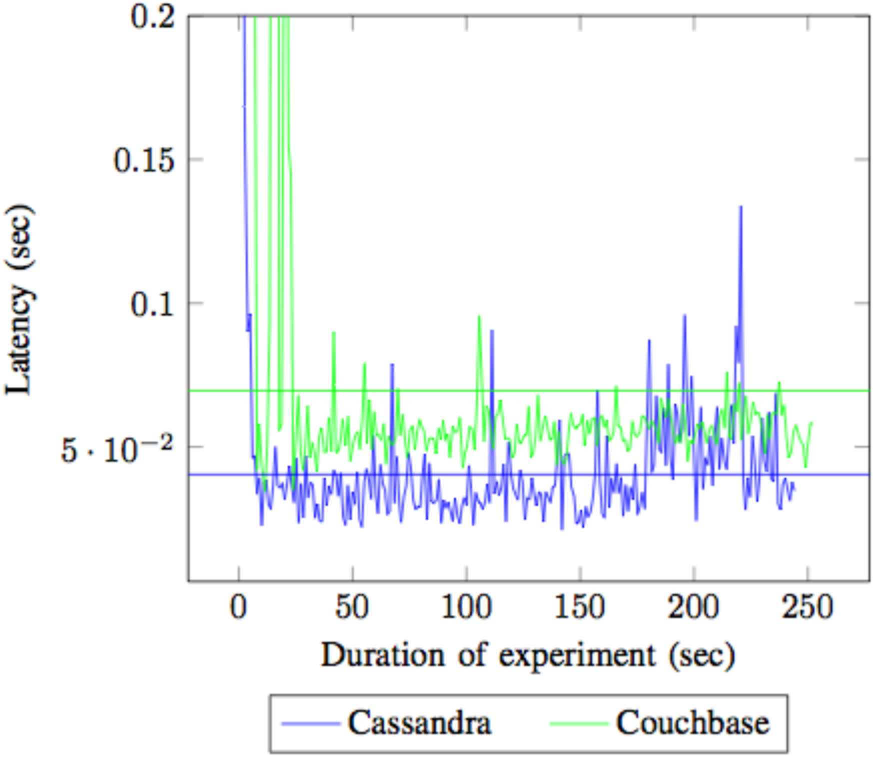
Ease of Setup

- Cassandra - Awesome!
  - Install, configure addresses, partitioner, snitch, replication factors and seed nodes, launch!
  - Automatic partitioning and replication.
  - Automatic adjustment to churn.
- Couchbase - Equivalently Awesome!
  - Install, configure RAM, directory, launch!
  - Easily add/failover/remove servers
  - Rebalance: background process, asynchronous, incremental
  - Automatic replication, partitioning and node failure detection.

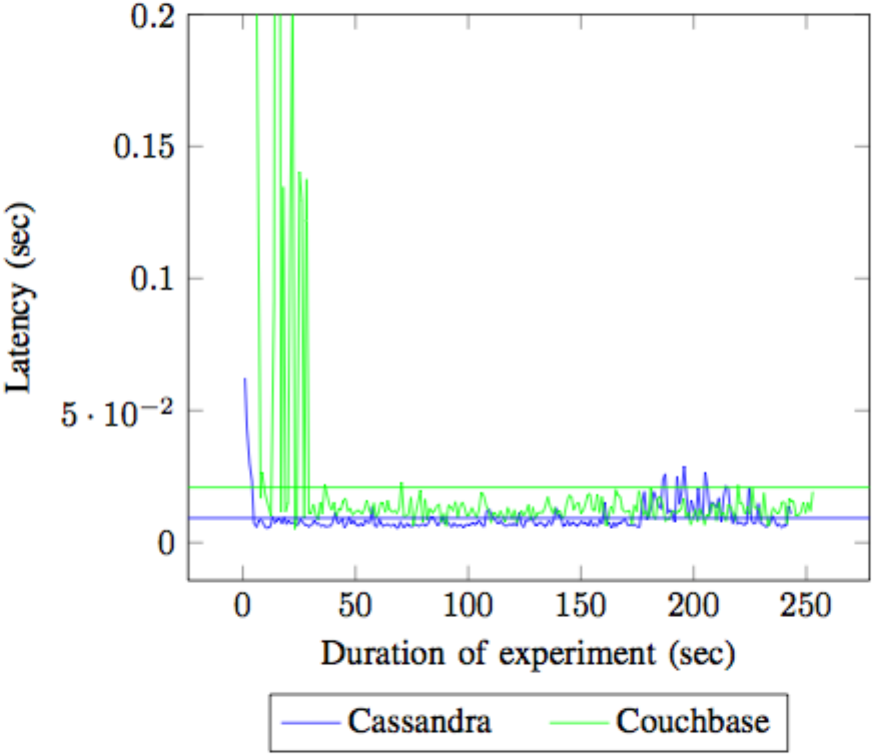
Setup time

- Cassandra:
  - 6 nodes form and stabilize ring after ~2 minutes.
  - Populating with sample data:
    - ~1 minute in 1 node configuration.
    - 6.5 minutes in 6 node configuration.
- Couchbase:
  - 6 nodes form and stabilize ring after ~2 minutes.
  - Populating with sample data:
    - ~1 minute in 1 node configuration.
    - ~2 minutes in 6 node configuration.

Latency

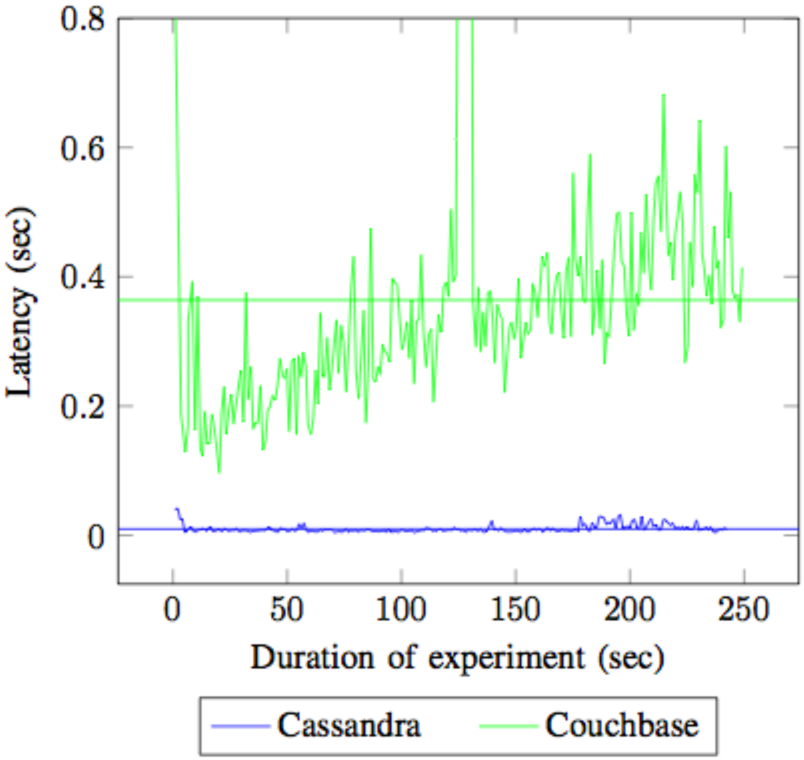


Tweet Latency



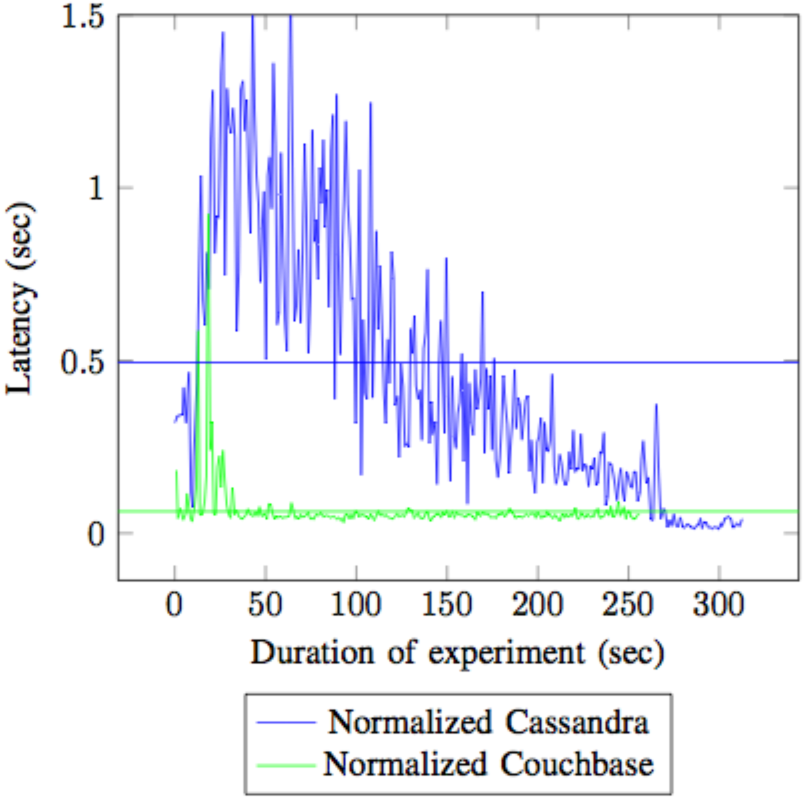
Userline Latency

Latency



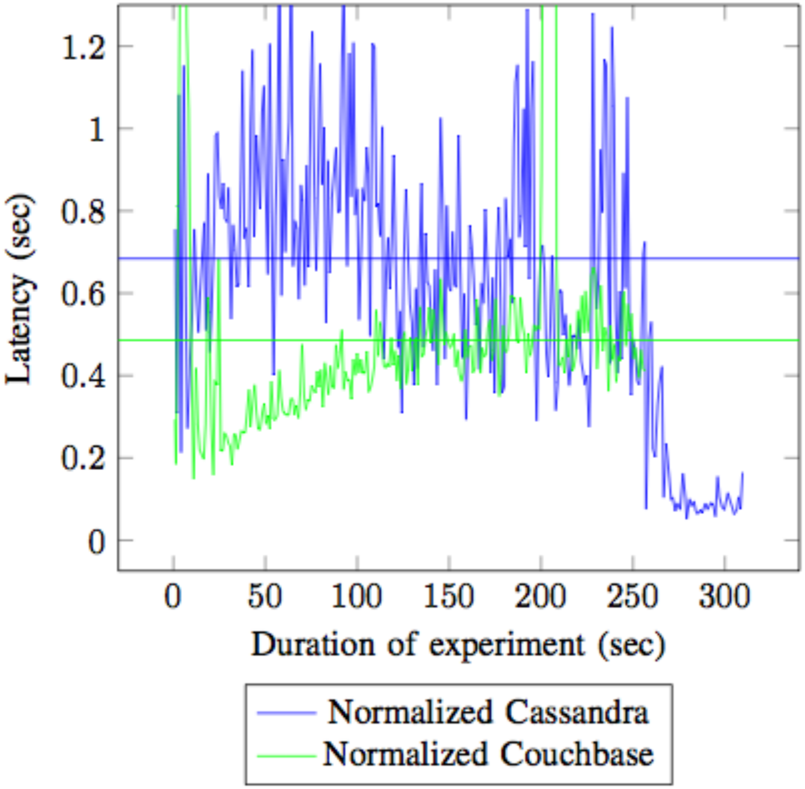
Timeline Latency

Normalized latency

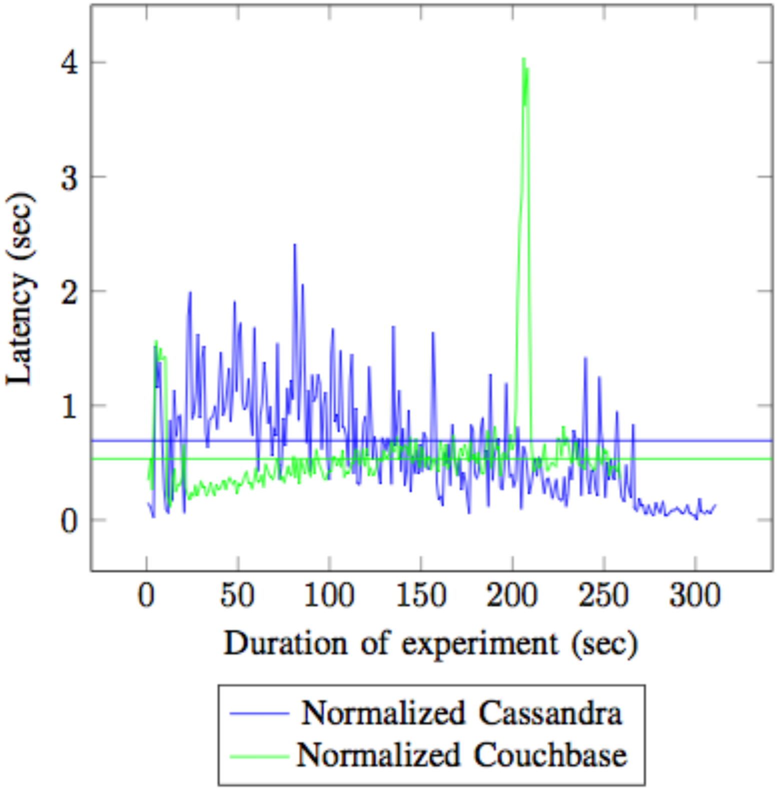


Tweet Latency

Normalized latency

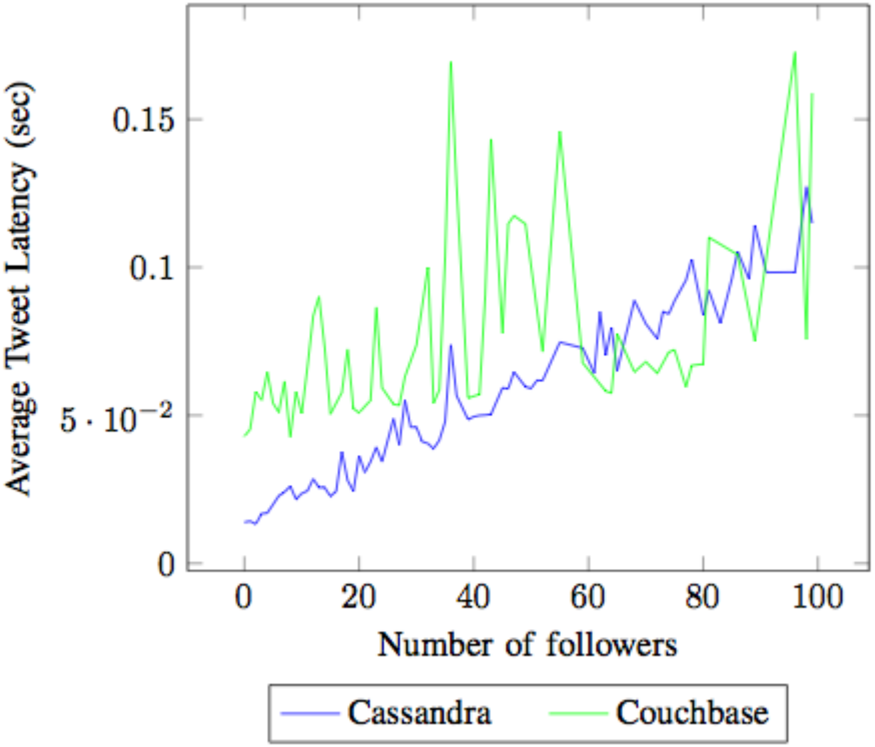


Userline Latency



Timeline Latency

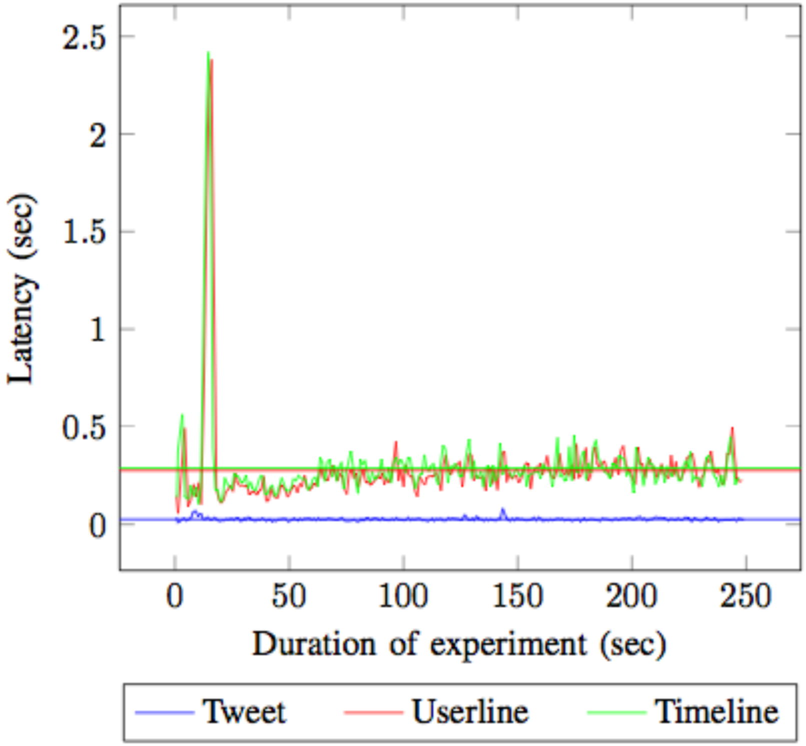
Tweets & Denormalization



Tweet Latency v.s. Number of Followers

Tweets & Denormalization

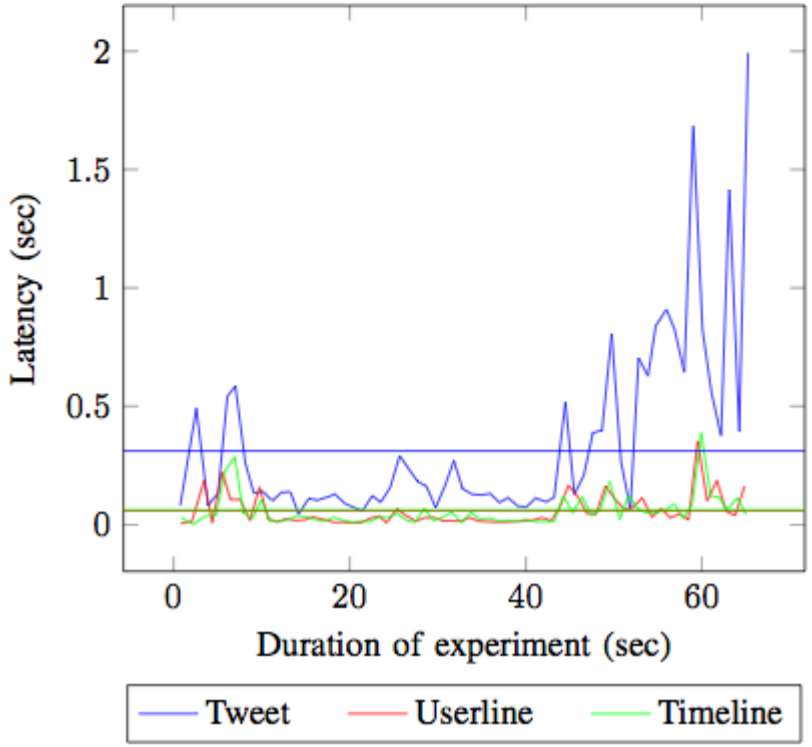
- Immediate denormalization not scalable.
- How to make asynchronous?
    - Cassandra:
      No native support.
      External processing.
    - Couchbase: Views!



Reconfiguration Latency

- Cassandra

  Adding a new node
  to a cluster: ~5 mins.



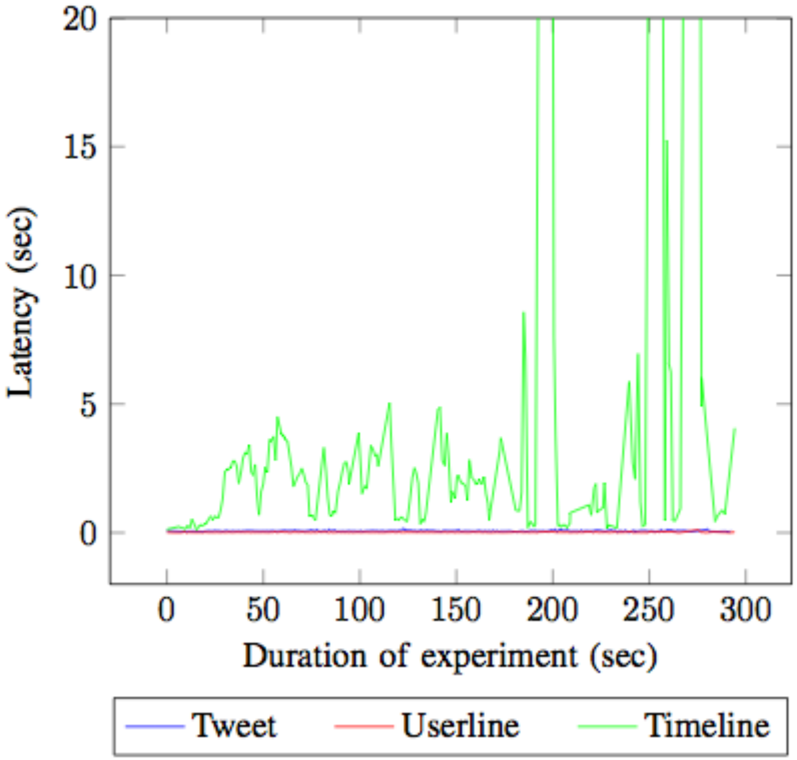
Latency while node joining cluster

Reconfiguration Latency

- Couchbase

  Adding a new node to a cluster: immediate

  BUT rebalance ~ 30 mins



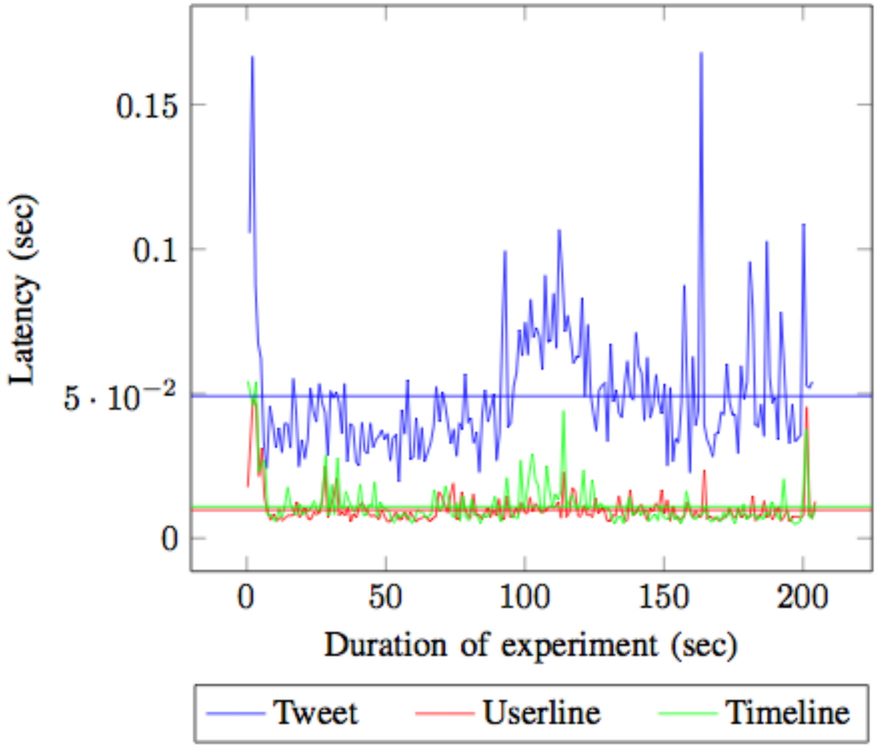
Latency while node joining cluster

Consistency/Convergence

- Cassandra:
  - Average of 0.096498 seconds to detect new tweet.
  - Standard deviation of 0.096319:
    - Same datacenter => very fast detection.
    - Different datacenter => slower detection.
- Couchbase:
  - Average of 0.001593 seconds to detect new tweet with standard deviation of 0.000526
  - The delay for new tweet to appear on timeline proportional to the schedule period

Replication

- Cassandra:
  - Very flexible in terms of replication configuration.
  - Per datacenter replication factors with no hard-coded limitations.

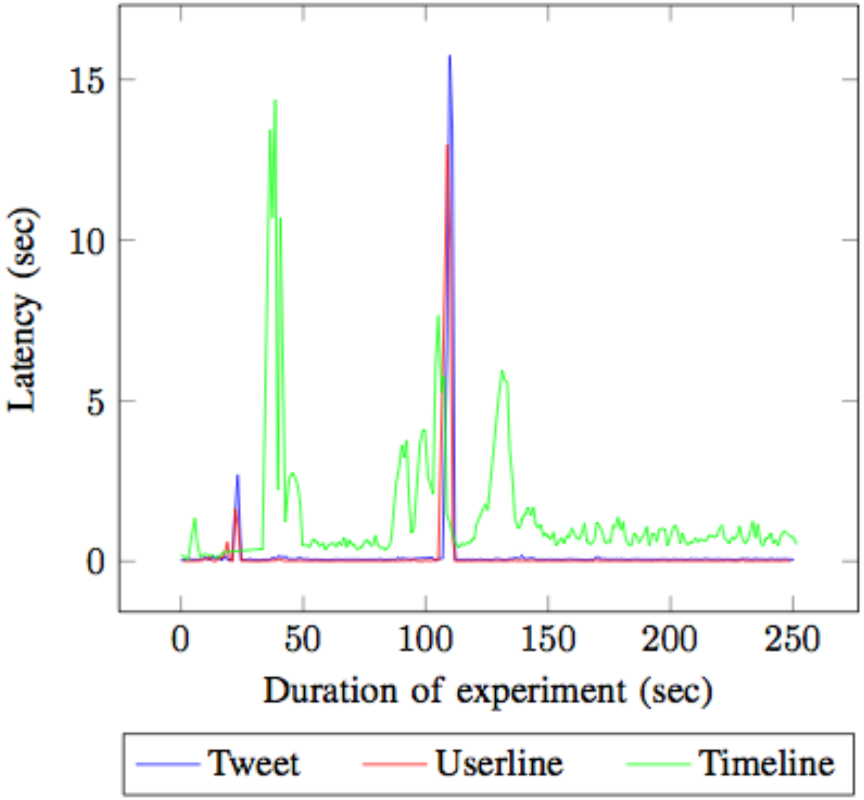


Behaviour under crash of 2 nodes @ second 100th

Replication

- Couchbase

  Automatic and configurable per bucket with a limit of (1+3) replicas.



Behaviour under crash of 2 nodes @ second 30th, 100th

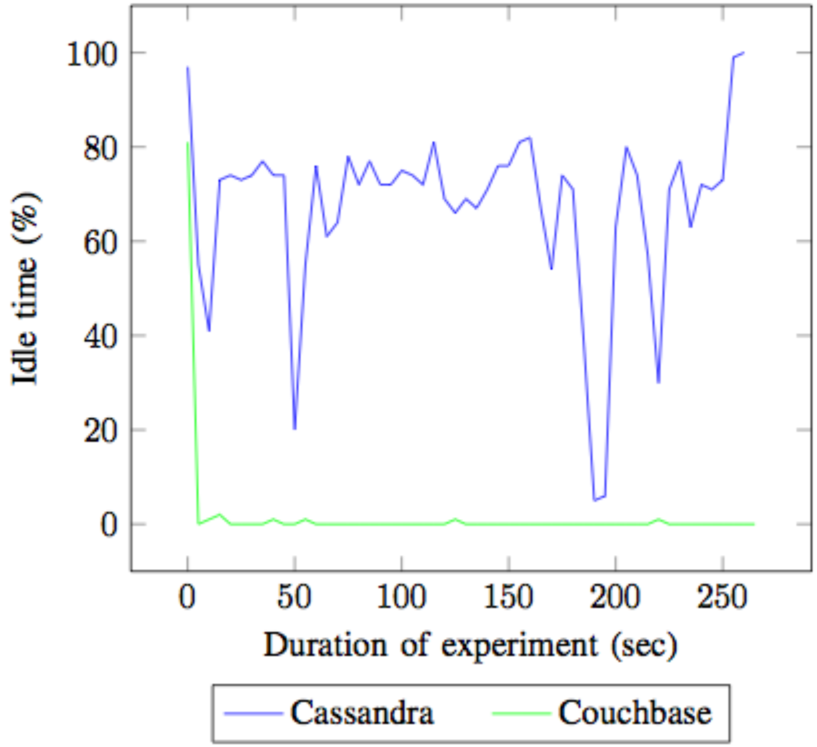
Load Balancing

- Cassandra:
  - Average data ownership per node: 16.68%
  - Standard deviation: 1.23%
- Couchbase:
  - Evenly distributed with standard deviation:

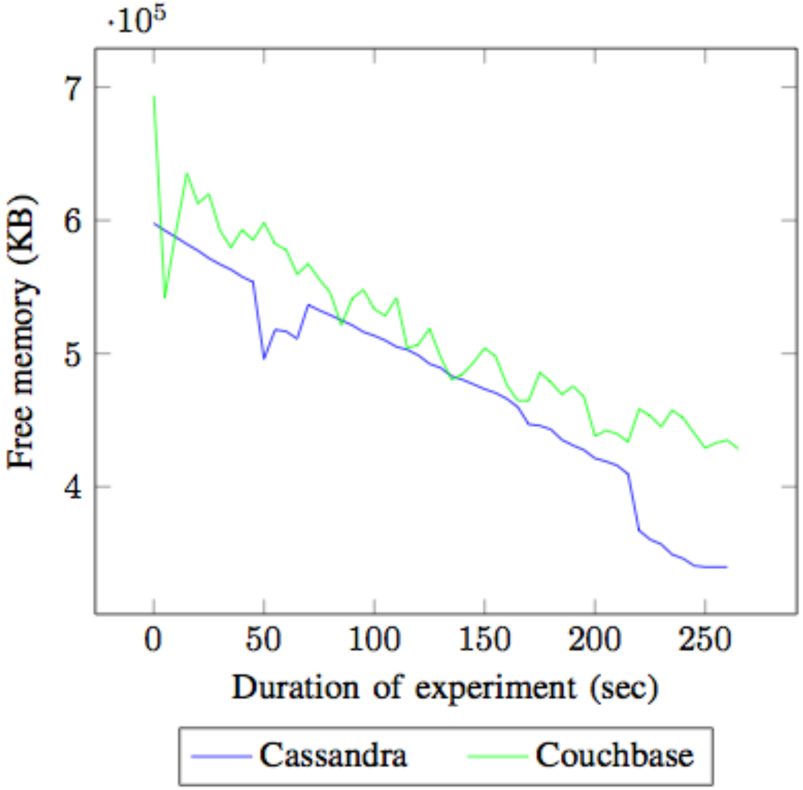
    0.65% for data on disk

    0.04% for RAM usage

System Load

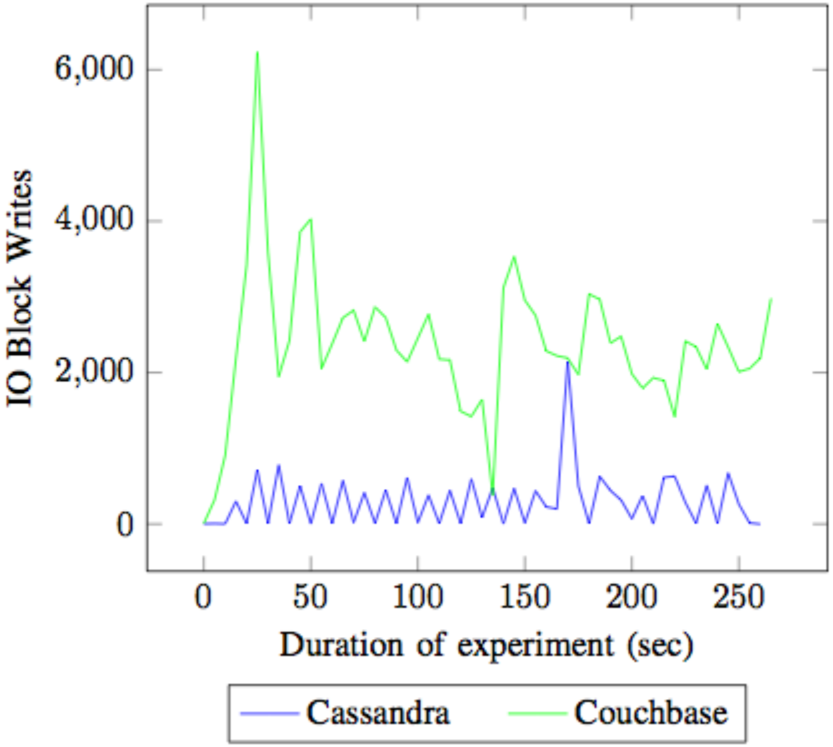


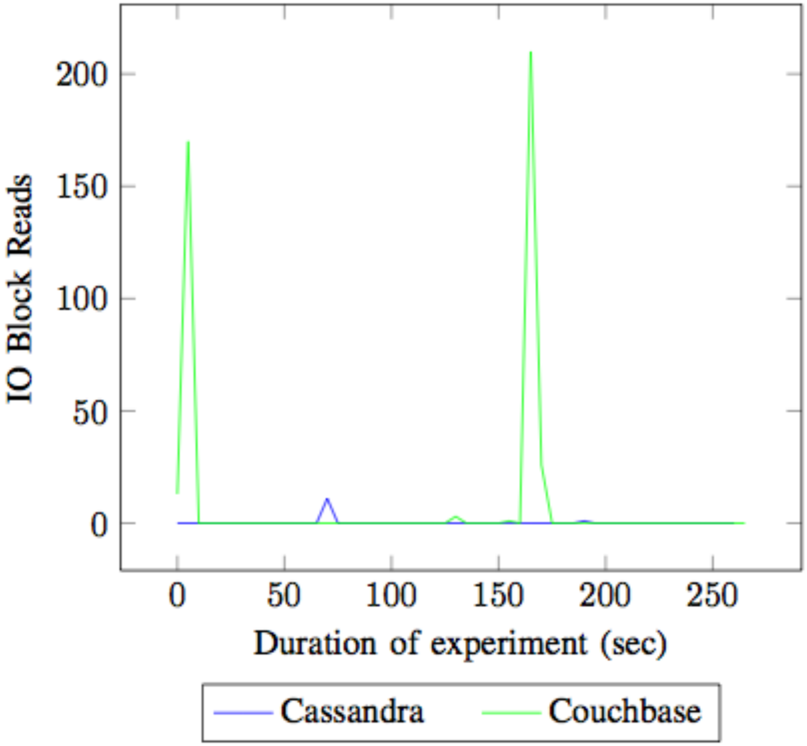
CPU Idle Time



Free Memory

System Load





Disk IO

Disk Space Usage

- SQLite database: 1.6MB
- Cassandra:
    - Fully denormalized (single node): 16.31MB.
    - Fully denormalized (6-nodes): 9.5MB/node.
        - Includes partitioning and replication.
    - "Normalized" (no body in userline/timeline):
        - Single node: 2.8MB
        - 6-nodes: 1.6MB/node.
    - Commit log after populating: 54MB
    - Need 50% of free disk space at all times:
        - Column family compactions.
        - Data redistributions.

Disk Space Usage

- Couchbase:
  - Total of 250MB distributed across 6 nodes.
  - No minimum free space requirement.

- Lack of disk usage limitations in both DBs:
  - Not ideal for voluntary computing systems.

Conclusion

- Easy cluster setup
  - Allows horizontal scaling over multiple nodes
- Improved performance through denormalization
- Higher storage requirements
- The future is hybrid
  - A mix of RDS and NoSQL-Systems
  - Cassandra's CQL, CouchBase's Views, NoSQL in RDS