On Brewing Fresh Espresso:

LinkedIn's Distributed Data Serving Platform

Lin Qiao, Kapil Surlaker, Shirshanka Das, Tom Quiggle, Bob Schulman, Bhaskar Ghosh, Antony Curtis, Oliver Seeliger, Zhen Zhang, Aditya Auradkar, Chris Beavers, Gregory Brandt, Mihir Gandhi, Kishore Gopalakrishna, Wai Ip, Swaroop Jagdish, Shi Lu, Alexander Pachev, Aditya Ramesh, Abraham Sebastian, Rupa Shanbhag, Subbu Subramanium, Yun Sun, Sajid Topiwala, Coung Tran, Jemiah Westerman, David Zhang

SIGMOD'13, June 22-27, 2013, New York, New York, USA.

OUTLINE

- Introduction
- Features
- External Interface
- System Architecture
- Implementation
- Experimental Evaluation
- Summary

INTRODUCTION

- Espresso: A document oriented, distributed data serving platform.
- LinkedIn's requirements:
  - Scale and Elasticity
  - Consistency
  - Integration
  - Bulk Operations
  - Secondary Indexing
  - Schema Evolution
  - Cost to Serve

FEATURES

- Transaction Support
- Consistency Model
- Integration with the complete data ecosystem
- Schema Awareness and Rich functionality

EXTERNAL INTERFACE

- Data Model
- API
- Bulk load and Export

EXTERNAL INTERFACE: *Data model*

- The two primary forms of relationships:
  - Nested Entities
  - Independent Entities
- Document
- Table
- Document Group
- Database

```
"Title" : "CouchDB",
"Last editor" : "172.5.123.91",
"Last modified": "9/23/2010",
"Categories": ["Database", "NoSQL", "Document Database"],
"Body": "CouchDB is a ...",
"Reviewed": false
```

EXTERNAL INTERFACE: *API*

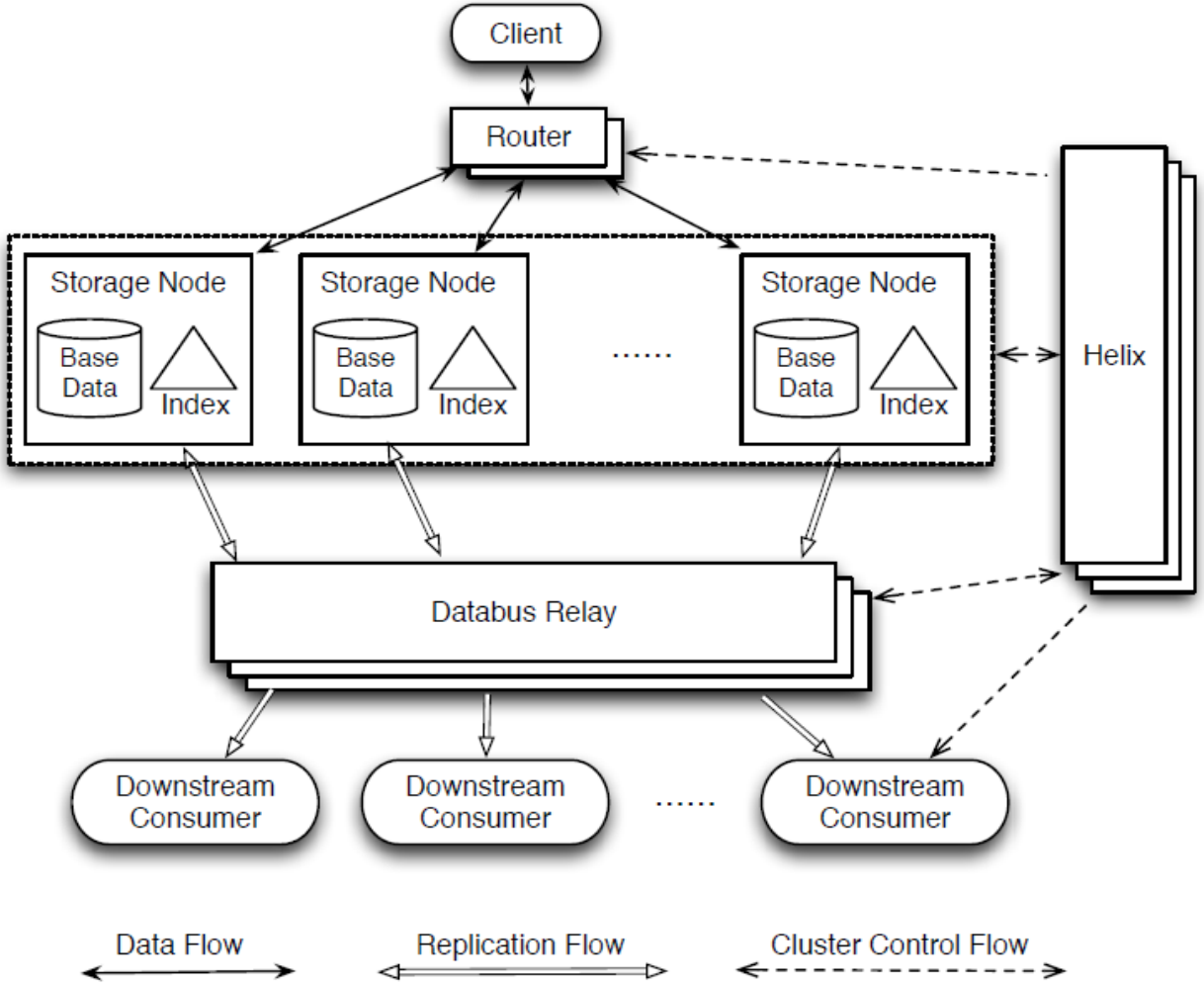
- REST API
- The capabilities provided to the application developer:
  - Read Operations
  - Write Operations
  - Conditionals
  - Multi Operations
  - Change Stream Listeners

EXTERNAL INTERFACE: *Bulk load and export*

- Efficient ingestion of large amounts of data from offline environments like Hadoop into Espresso.

- Data Export is important to support ETL and other offline data analysis use-cases.

SYSTEM ARCHITECTURE



IMPLEMENTATION

- Secondary Index
- Partitions and Replicas
- Internal Clock and Timeline
- Replication and Consistency
- Fault Tolerance
- Cluster Expansion
- Multi Datacenter

IMPLEMENTATION: *Secondary indexing*

- Fundamental building block: Inverted Index
- Local secondary indexes
  - Secondary indexes on document groups
- Global secondary indexes
  - Secondary indexes on independent entities.
- First attempt: Lucene
- Second attempt (indexing solution) : Prefix

IMPLEMENTATION: *Partitions and replicas*

- Partition of data is performed to serve the following purposes:
  - Load balancing
  - Efficient and predictable cluster expansion
- Data is partitioned into a large number of partitions.
- Over partitioning keeps partition size small.
- Each partition is mastered at one node and replicated on N nodes.

IMPLEMENTATION: *Internal clock and timeline*

- Each database partition operates as an independent commit log.
- Each commit log acts as a timeline of data change events that occurred on that partition.
- A timeline consists of ordered sets of changes in the partition.

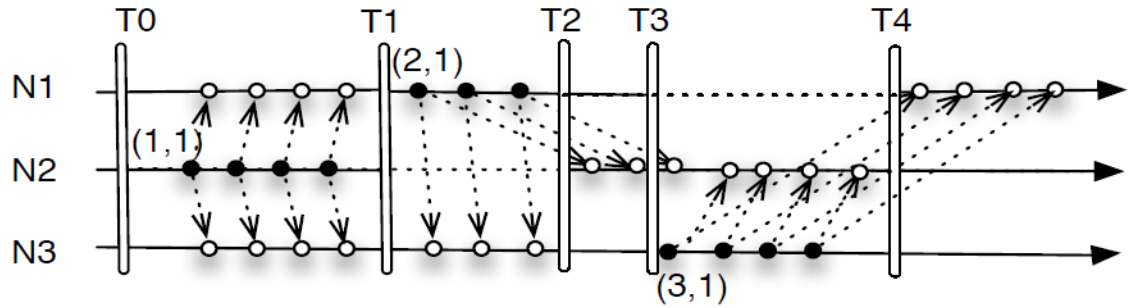


Figure 3: Timeline of Events

IMPLEMENTATION : *Replication and consistency*

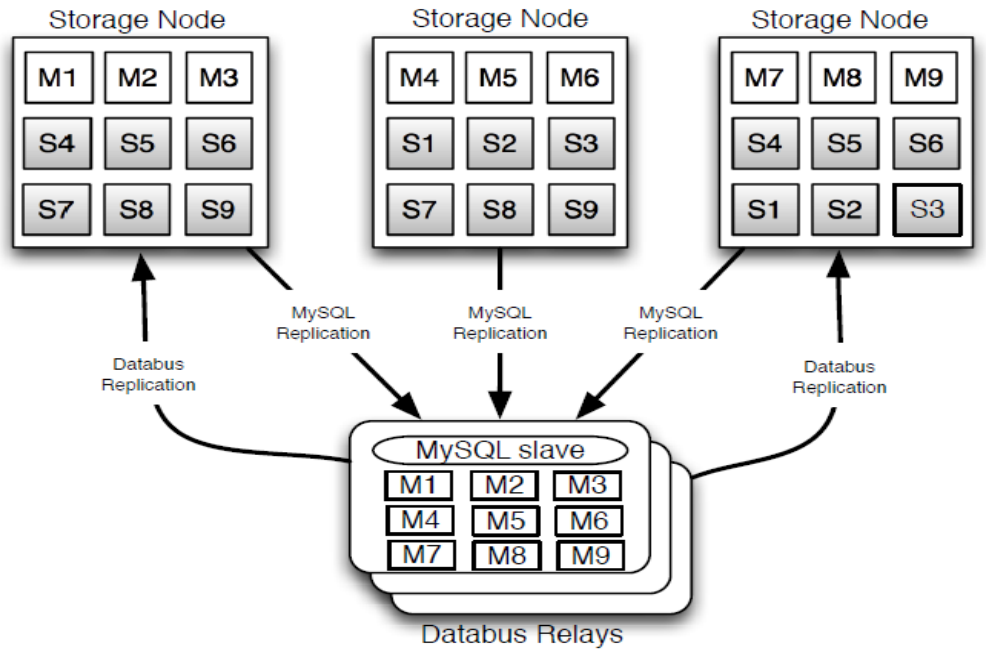


Figure 2: Partitions and Replication Flow

- Consistency is ensured with the help of consistency checker.

IMPLEMENTATION*: Fault tolerance*

- Each component in Espresso is fault tolerant.
- Storage node Failure:
  - Helix calculates a new set of master partitions from existing slaves.
- Failure Detection: Helix
  - Zookeeper heartbeat for hard failure
  - Monitor performance metrics reported by the router and the storage nodes
- Data bus is also fault tolerant

IMPLEMENTATION: *Cluster expansion*

New storage nodes can be added as the data size or the request rate approaches the capacity limit of a cluster.

IMPLEMENTATION: *Multi data center*

A warm standby is located in a geographically remote location to assume responsibility in the event of a disaster.

EXPERIMENTAL EVALUATION

- Availability
- Elasticity
- Performance

EXPERIMENTAL EVALUATION: *Availability*

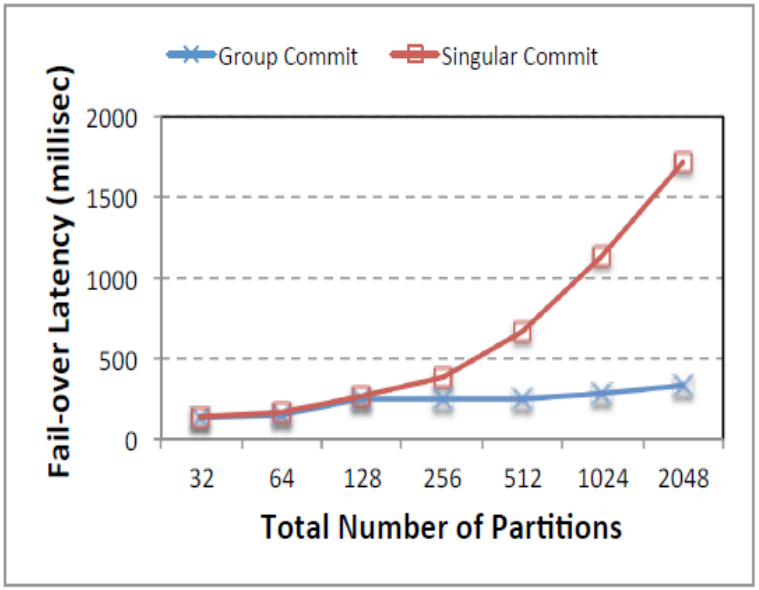


Figure 4: Fail-over Latency

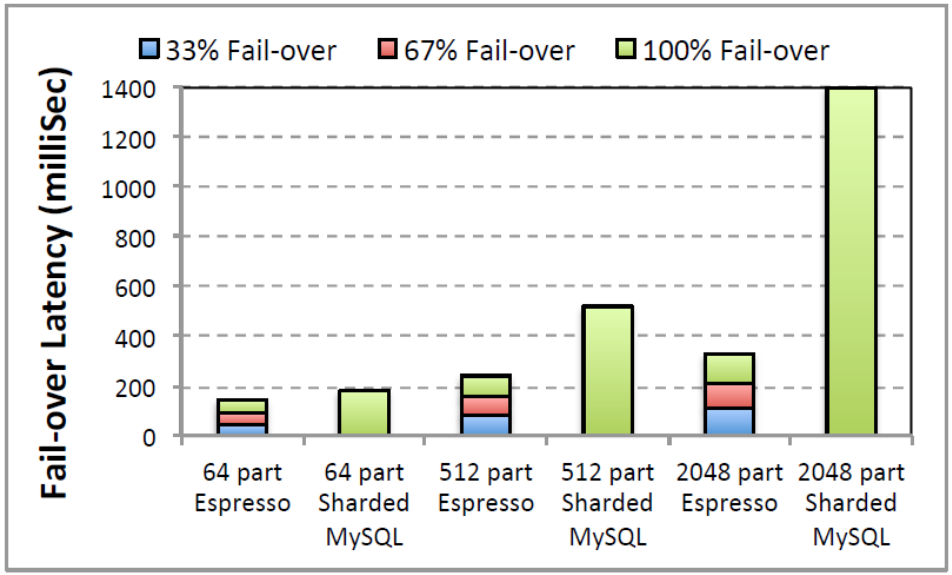


Figure 5: Espresso vs MySQL Fail-over Comparison

EXPERIMENTAL EVALUATION: *Elasticity*

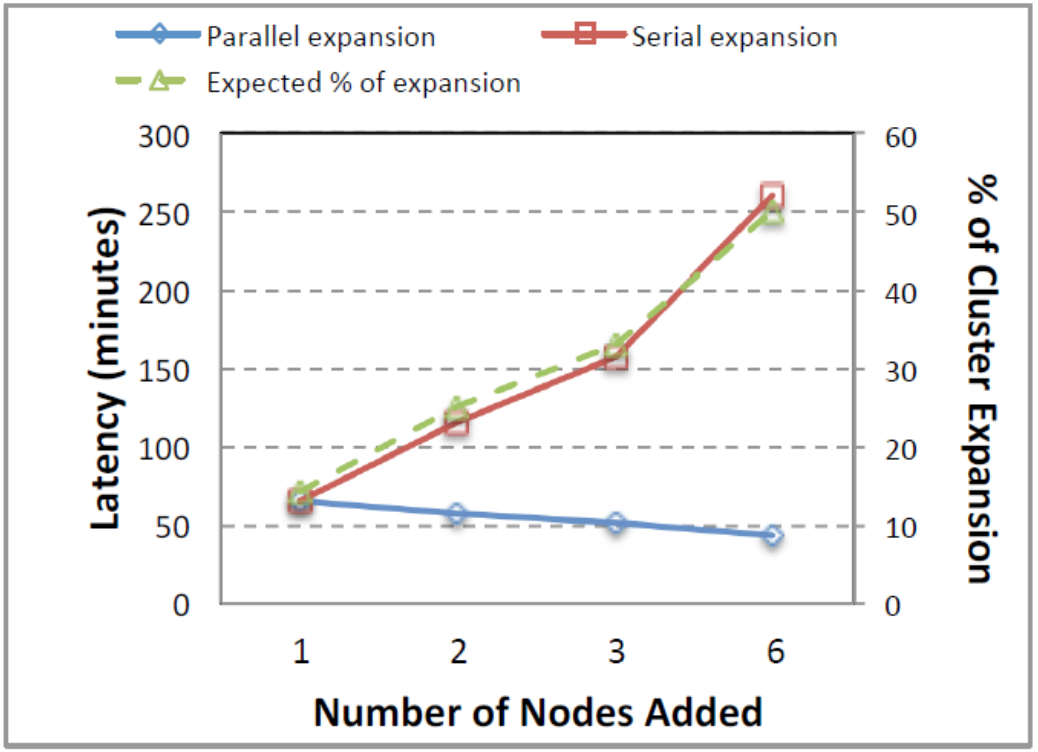
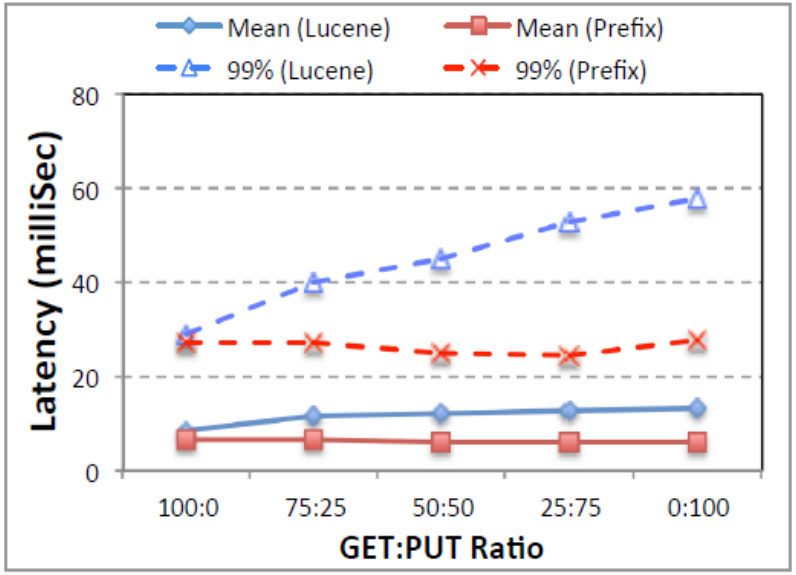
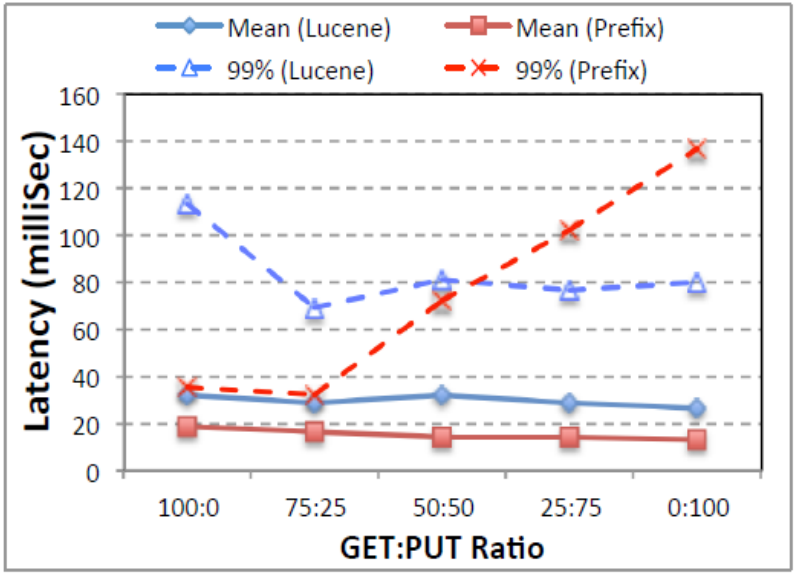


Figure 6: Cluster Expansion Performance

EXPERIMENTAL EVALUATION: *Performance*



(a) Small MailBoxes



(b) Large MailBoxes

Figure 7: Index Performance Comparison

SUMMARY

- Espresso is a distributed document oriented database.
- It is timeline consistent, provides rich operations on documents and supports seamless integration with near line and online environments.
- Espresso uses master-slave architecture.
- Apache Helix, uses Zookeeper as the coordination service to store cluster metadata.
- Espresso has been developed in Java and is in production since June 2012.