Replex: A Scalable, Highly Available Multi-Index Data Store

By Amy Tai et al. - *Princeton University*

Presented by Camilla Stormoen

Overview

- Motivation
- System Design
- Hybrid Replex
- Evaluation
- Conclusion

Motivation

- Need for scalable, high-performance data stores → NoSQL databases

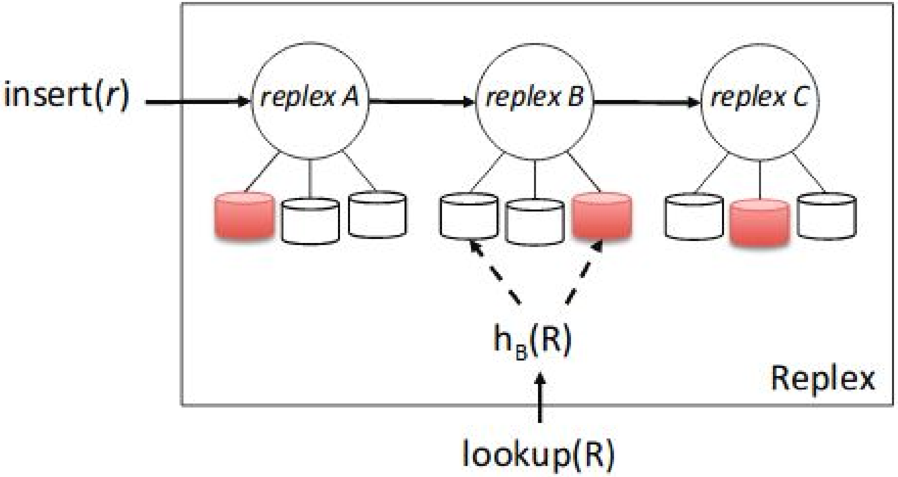
How do we put back the secondary indices, without compromising scalability, availability and performance?

- Replex
  - enables efficient querying on multiple keys
- Hybrid Replexes
  - enable rich design space for trading off steady-state performance with faster recovery

System Design

- Data Model and API
  - Stores data in the form of RDBMS tables
  - Table has schema with columns
  - Table-column primary key ➡ index of table

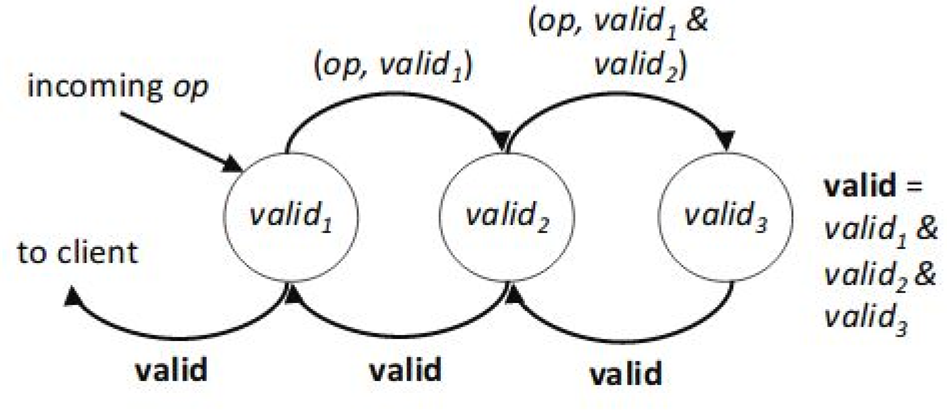
- Data Partitioning with Replexes
  - Fast query: Partitioned by index
  - Replex stores a table and shards the rows across multiple partitions
  - Chain replication



System Design: Replication Protocol Failure

- Individual replexes can have requirements
    - Can both be valid and invalid depending on the replex

- Chain replication includes a consensus protocol

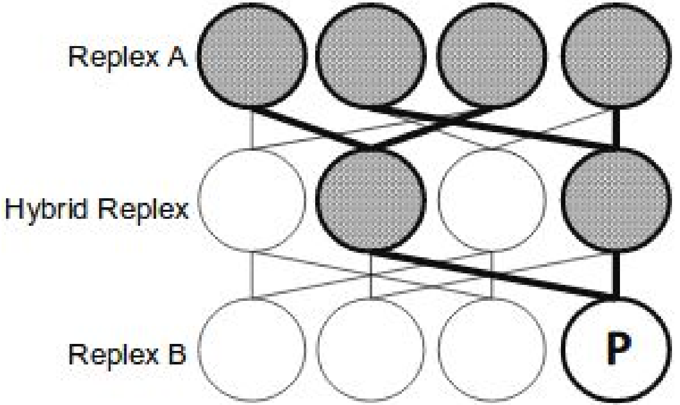


System Design: Amplification

- Two concerns:
    1. How to reconstruct the failed partition
    2. How to respond to queries that would have been serviced by the failed partition.
- Partition fails: read must broadcast to every partition and iterate through the entire local storage of partition
- Within a threshold: introduce f replexes with same sharding function
    - - Cost is storage and network overhead

Hybrid Replexes

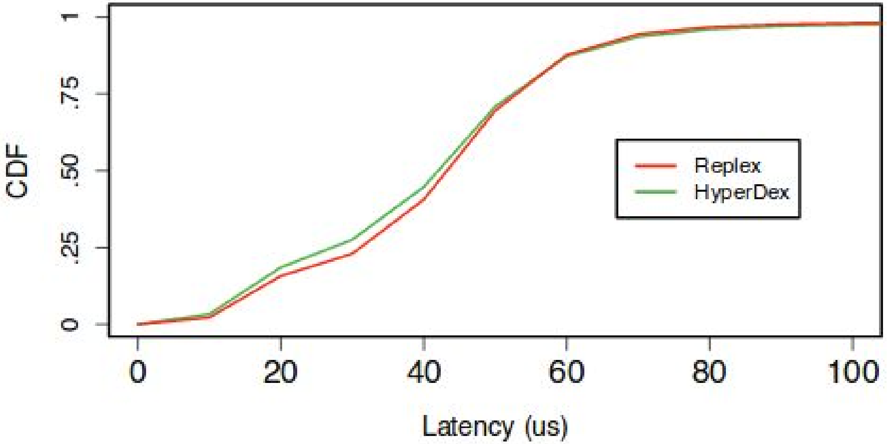
- Increase failure resilience of any number of replexes
- 2-Sharing
  - Two partitions share data if there is a path between them
  - P shares data with two partitions in hybrid replex and four in replex A
  - Read redirect to hybrid replex is faster compared to redirect to hybrid A



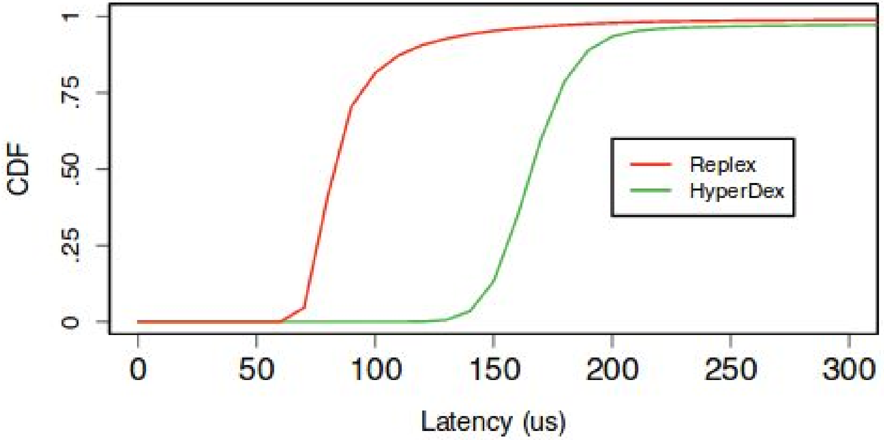
Evaluation: Steady-State Performance

- *How does Replex's design affect steady-state index performance?*
- Single hybrid replex (*Replex-3* - 3 replexes, tolerant to 2 failures)

- Read latency



- Insert latency



Evaluation: Failure evaluation

- *How do hybrid replexes enable superior recovery performance?*
- 12 virtual partitions per subspace/replex.
- Table with primary and secondary index. Split 50:50 reads
- Recovery time: reduce by 2-3x
- Recovery throughput: Replex-2 minimal throughput

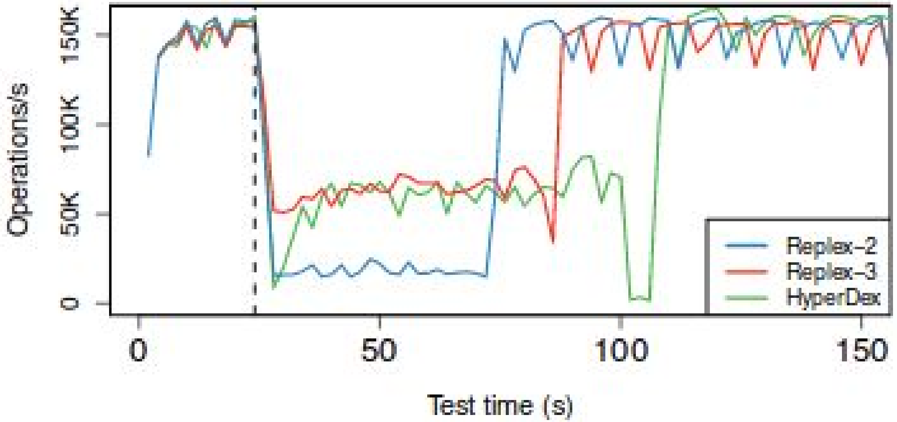
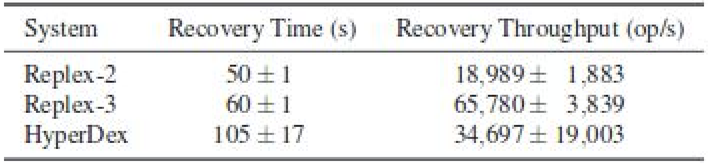
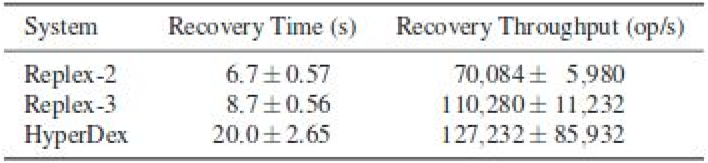


Table 1: 10 millions rows of size 100 bytes records



System	Recovery Time (s)	Recovery Throughput (op/s)
Replex-2	50 ± 1	18,989 ± 1,883
Replex-3	60 ± 1	65,780 ± 3,839
HyperDex	105 ± 17	34,697 ± 19,003

Table 2: 1 million rows of size 1 KB records



System	Recovery Time (s)	Recovery Throughput (op/s)
Replex-2	6.7 ± 0.57	70,084 ± 5,980
Replex-3	8.7 ± 0.56	110,280 ± 11,232
HyperDex	20.0 ± 2.65	127,232 ± 85,932

Conclusion

- Able to query data by more than just a single key
- Replex’s steady-state performance 76% better than HyperDex
- Multi-Index, scalable, highly-available NoSQL data store is possible and a better choice

Questions?