



BENCHMARKING AVAILABILITY AND FAILOVER PERFORMANCE OF LARGE-SCALE DATA STORAGE APPLICATIONS

Wei Sun and Alexander Pokluda

December 2, 2013

Outline

- Goal and Motivation
- Overview of Cassandra and Voldemort Design
- Benchmark Setup and Methodology
- Preliminary Results and Status Report
- Conclusion

Goal and Motivation

- **Goal:** To understand failover characteristics of large-scale data storage applications
- Few benchmarks of failover characteristics have been done
- We have chosen to study the following:
 - Cassandra
 - Voldemort
 - HBase (if time permits)
- Cassandra and Voldemort were chosen because both emphasize availability and performance
- HBase, which emphasizes consistency and performance, was chosen to cover a wider range of architectures

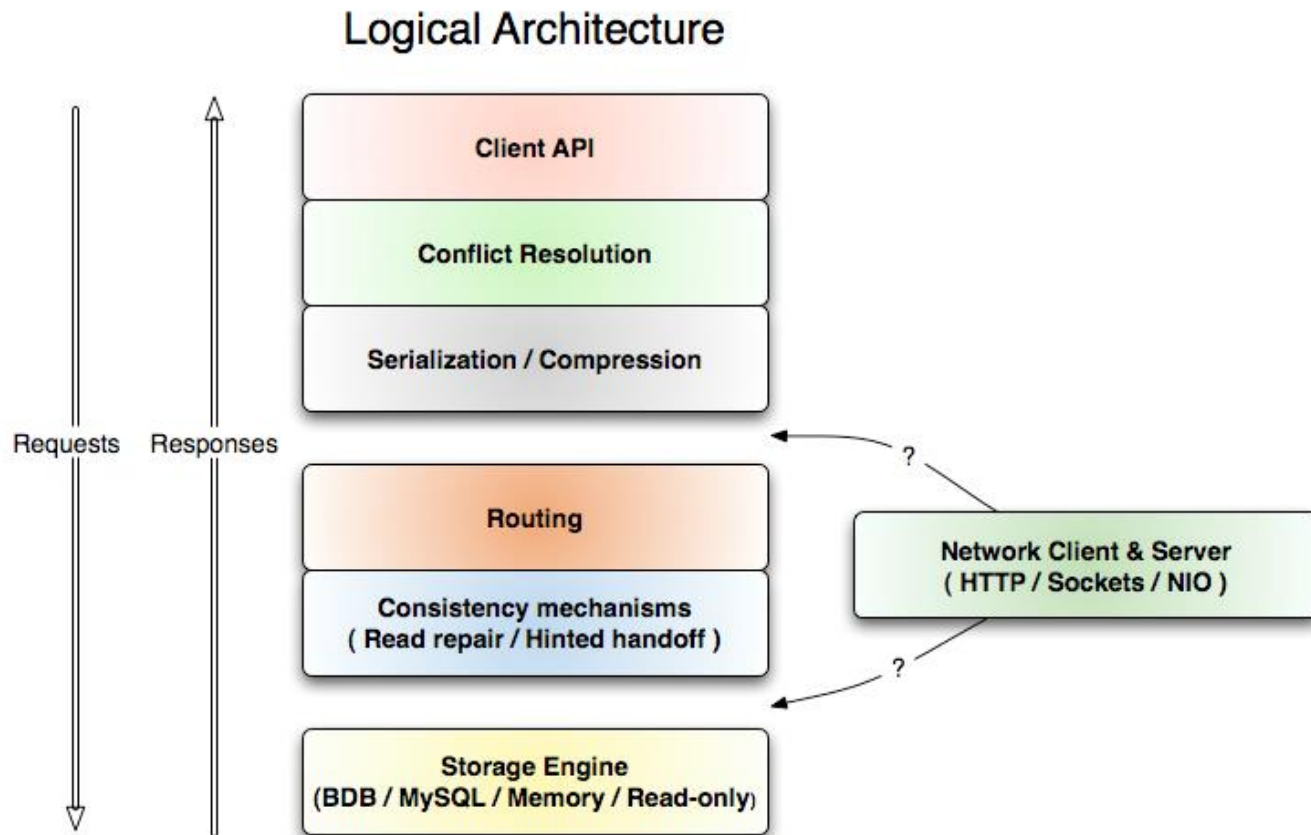


OVERVIEW OF CASSANDRA AND VOLDEMORT DESIGN

Cassandra

- Architecture
 - Mixture of Dynamo and BigTable
- Consistent hashing
 - Order preserving hash function
- Various replication options
 - Rack unaware, rack aware, datacenter shard
- Fault tolerance
 - Accrual Failure Detection
- Node Failure
 - Down and up: Zookeeper
 - Down entirely: replacement

Voldemort



Voldemort

- Consistent hashing
- Zone aware replication
 - User-defined per zone replication factor
- Consistency
 - Read-repair & quorum
 - Vector-clock versioning
- Node failure
 - hinted handoff

Cassandra vs. Voldemort

	Cassandra	Voldemort
Data Model	column database multi dimensional	key-value datastore hash table
Replication	synchronous/asynchronous chosen by application: Rack unaware, rack aware, across datacenter	Zone aware replication
Partitioning	consistent hashing (order preserving hash function)	consistent hashing
Consistency Model	tunable all the way from "writes never fail" to "block for all replicas to be readable", with the quorum level in the middle	tunable Quorum, read-repair, hinted handoff
Data Storage	Disk	Pluggable Storage Engines: BDB-JE, MySQL, Read-Only
Developed	by Facebook in Java	by LinkedIn in Java



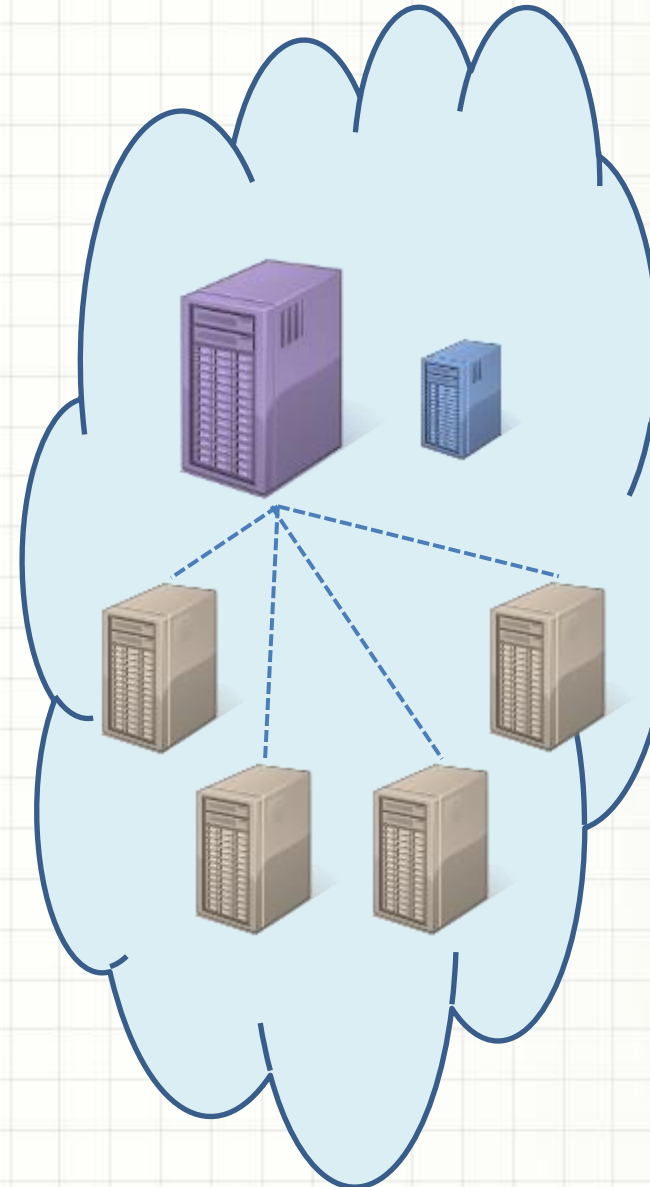
BENCHMARK SETUP AND METHODOLOGY

Methodology

- Using Yahoo! Cloud Serving Systems Benchmark (YCSB) for load generation and reporting
 - Extensible and generic framework for evaluation of key-value stores that has become an industry standard
 - Can generate synthetic workloads that consist of a configurable distribution of CRUD operations
- Measure latency for a variety of throughputs
- Measure throughput vs time and error count for blue-sky and failure scenarios
- Node failures simulated using **kill -9**
- Network failures simulated using **tcpkill** and firewalls
- Nodes run on AWS

Cluster Configuration

- Experimental setup consists of 5 nodes on AWS
- Database Servers
 - 4x m1.xlarge Spot Instances
 - 4 vCPU (8 ECU), 15 GiB RAM
 - 4x420 GB disk as RAID 1+0
- YCSB Load Generator
 - 1x m3.2xlarge Spot Instance
 - 8 vCPU (26 ECU), 30 GiB RAM
- Network throughput between database and YCSB servers consistently > 960 Mbit/s
- NFS Server
 - 1x m1.small On-Demand Instance
 - 1 vCPU (1 ECU), 1.7 GiB RAM
 - 10 GB persistent EBS volume mounted at /home on all servers

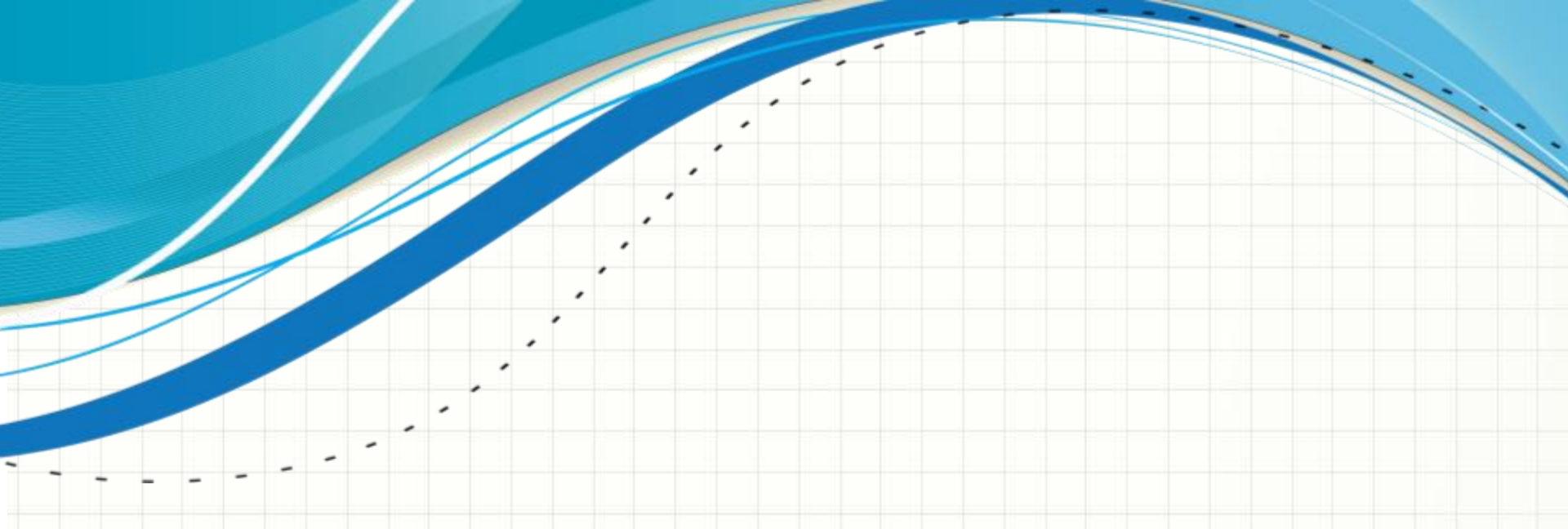


Workloads and Parameters

- **Workload A:** Update heavy workload
 - 50% Reads, 50% Writes
 - **Example:** session store recording recent actions
- **Workload B:** Read mostly workload
 - 95% Reads, 5% Writes
 - **Example:** photo tagging
- **Workload C:** Read only
 - 100% Read
 - **Example:** user profile cache

Workloads and Parameters

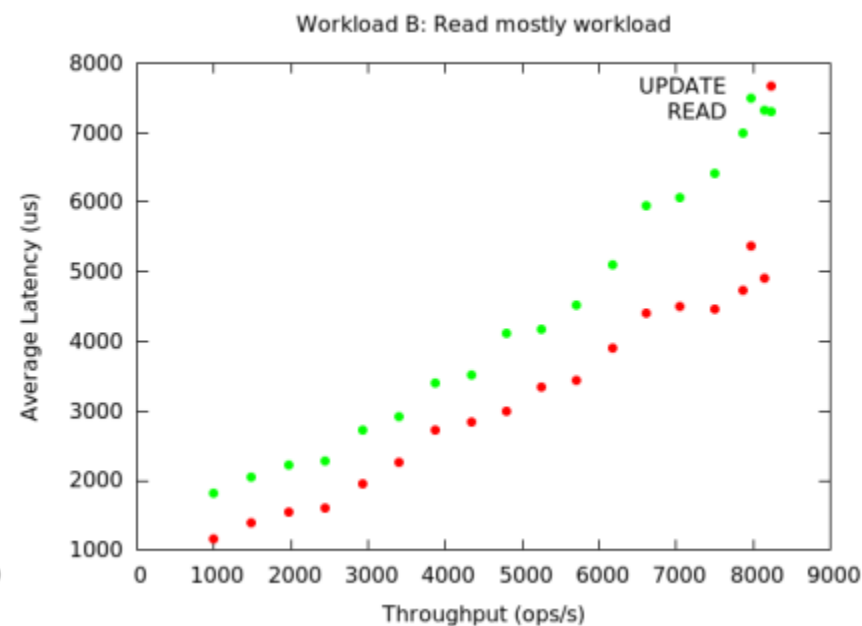
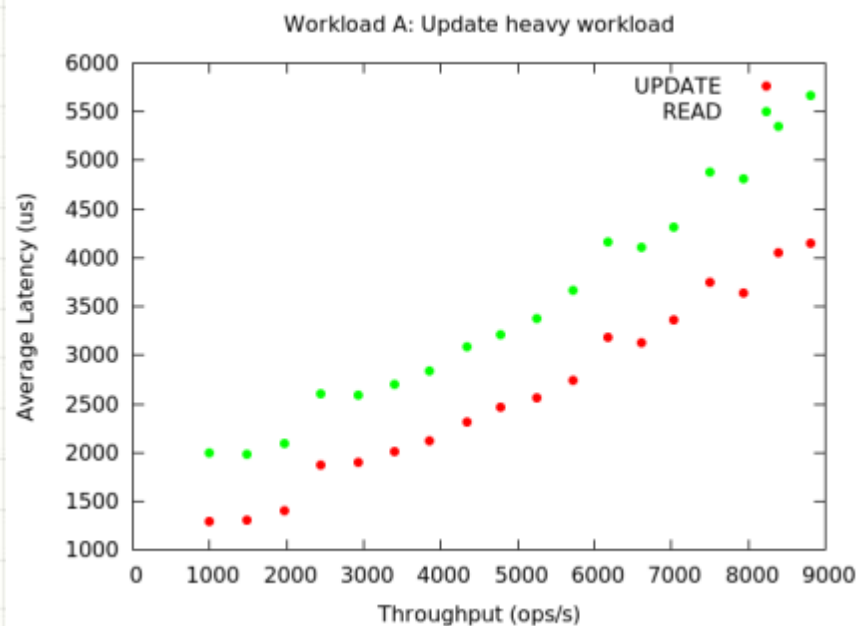
- **Workload D:** Read latest workload
 - New records inserted, reads mostly on latest inserted
 - **Example:** User status updates
- **Workload E:** Short ranges
 - Short ranges queried
 - **Example:** Threaded conversations (clustered by thread ID)
- **Workload F:** Read-modify-write
 - Records read, modified and written back
 - **Example:** User database

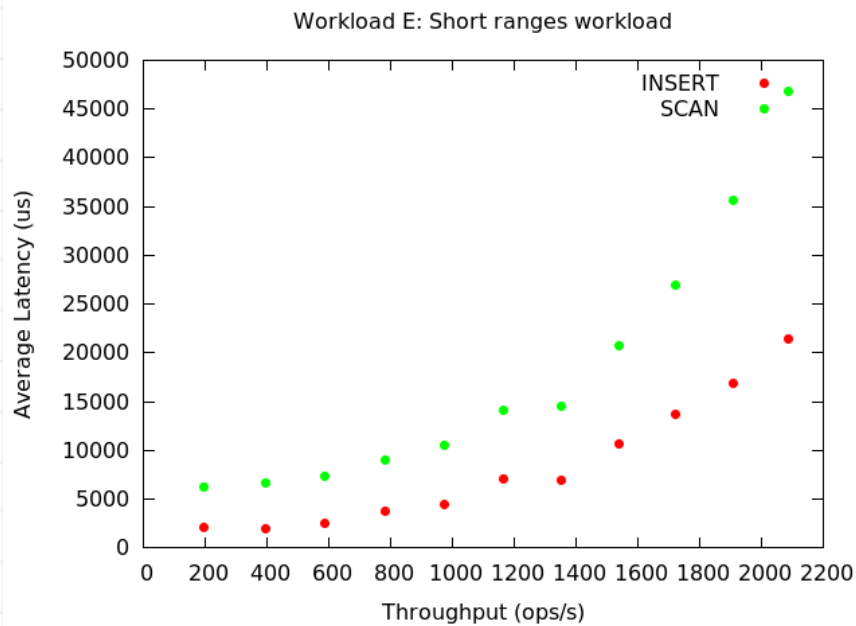
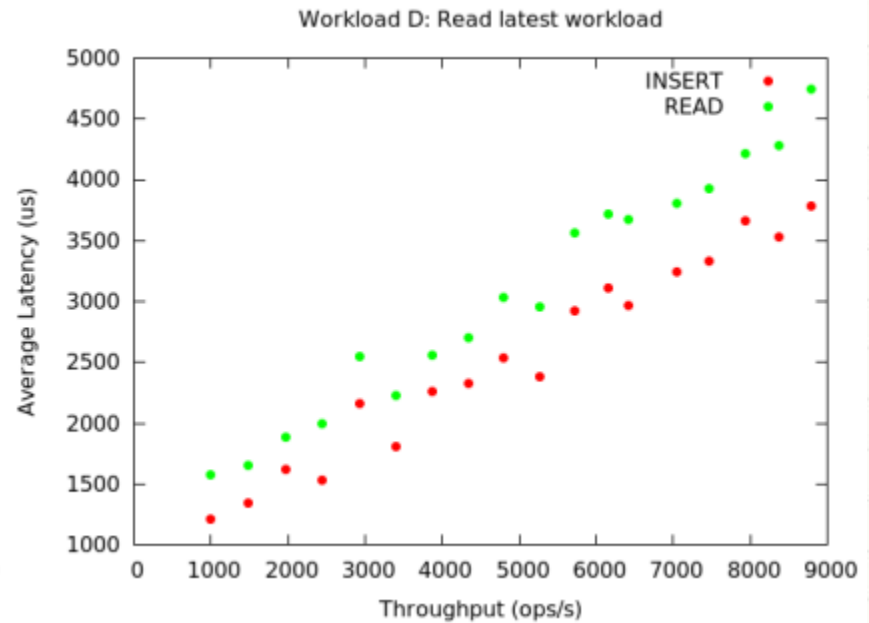


PRELIMINARY RESULTS AND STATUS REPORT

Cassandra: Latency vs Throughput

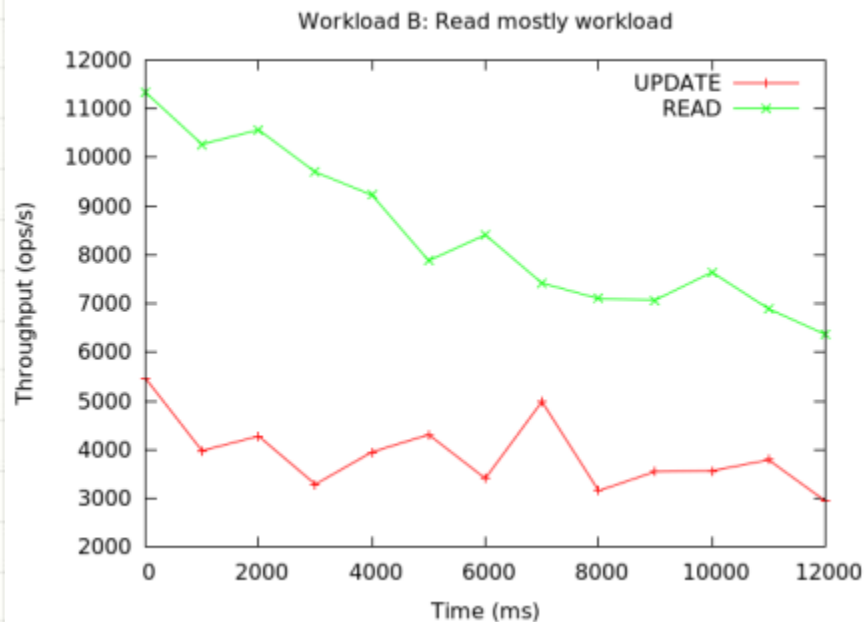
- Cassandra 2.0.2
- No failures





Cassandra: Throughput vs Time

- **Left:** No failures
- **Right:** 1 of 4 nodes killed at 20000 ms



Challenges

- The YCSB github repository is out of date and the documentation is incomplete
 - Only Cassandra 0.5, 0.6, and 0.7 are supported
 - A patch was submitted for Cassandra 1.0.6 but not documented
 - Only Voldemort 0.81 is supported (but this is not documented)
 - After a long search I found someone's personal fork of YCSB with support for Cassandra 2.0 (CQL) and an updated patch for Voldemort 0.96 in one of the Voldemort contributor's github repository

Status Report

- Need to re-run Cassandra tests with correct **threadcount** parameter and fork of YCSB that supports Cassandra 2.0
- **ObsoleteVersionExceptions** are preventing Voldemort benchmark from progressing
 - Contacted Voldemort developers through issue tracker: they said some **ObsoleteVersionExceptions** are normal
 - I'm working on patching the code based on stotch's recommendations
- Need to test failure scenarios
- Need to run Hbase tests (time permitting)



CONCLUSION

Summary

- Our goal is to understand failover characteristics of large-scale data storage applications
- Few benchmarks of failover characteristics have been done
- Presented an overview of the design of Cassandra and Voldemort
- Presented preliminary benchmark results using YCSB on a small cluster of nodes running in AWS

Lessons Learned

- Learned how to use AWS EC2 and VPC
 - Learned differences between EBS, Instance Store and S3 and how to create AMIs
 - Learned about instance types, placement groups and on-demand vs spot instances
 - Learned about Regions and availability zones and how AWS is designed
 - Designed to isolate failures
- Learned about the design and implementation and how to install, configure and tune several NoSQL Systems



QUESTIONS?