Benchmarking Cloud Serving Systems with YCSB

詹剑锋 2012年6月27日

Motivation

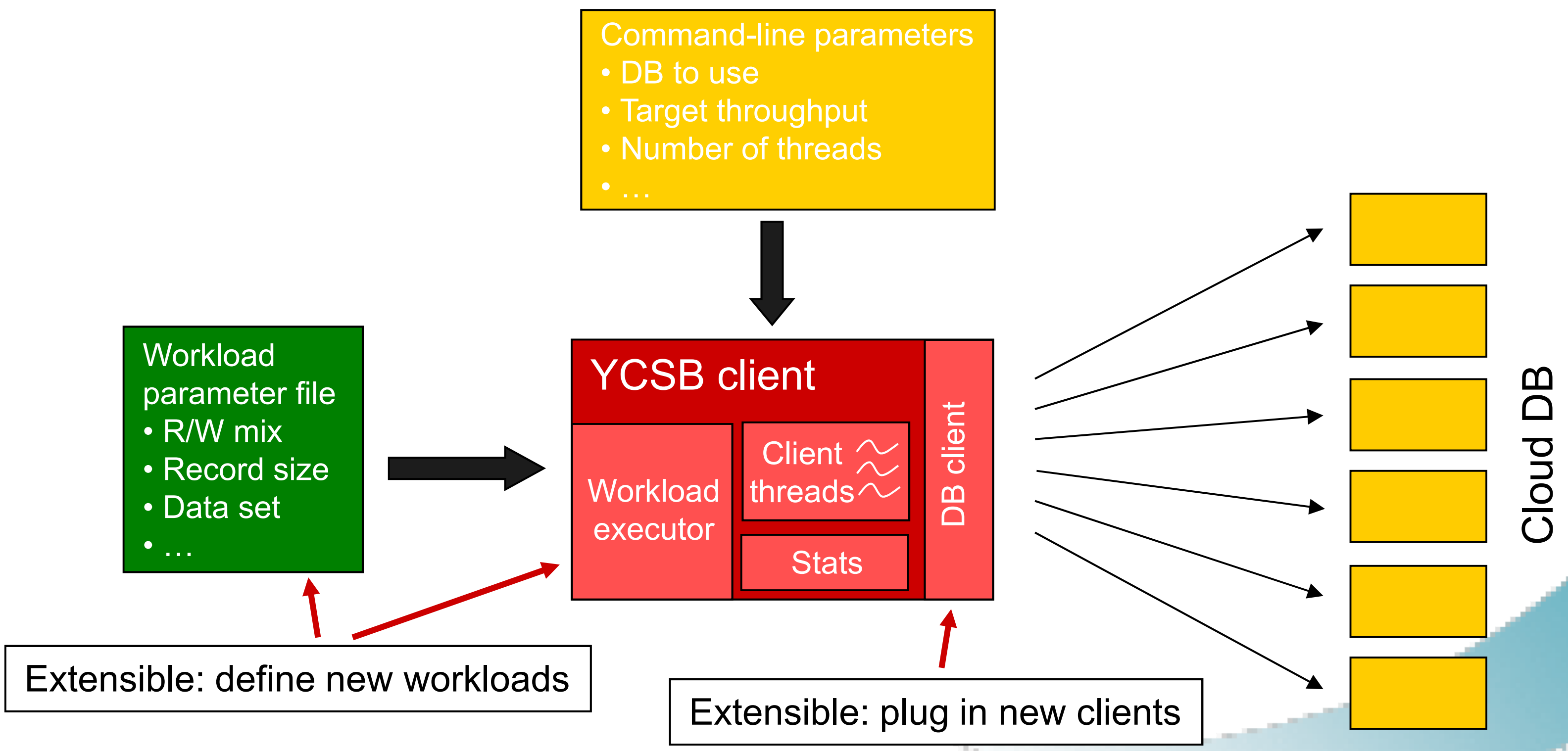
- There are many "cloud DB" and "nosql" systems out there
  - PNUTS
  - BigTable
    - HBase, Hypertable, HTable
  - Megastore
  - Azure
  - Cassandra
  - Amazon Web Services
    - S3, SimpleDB, EBS
  - CouchDB
  - Voldemort
  - Riak
  - Etc: Tokyo, Redis, MongoDB, Dynomite
- How do they compare?
  - Feature tradeoffs
  - Performance tradeoffs
  - Not clear!

Goal

- Implement a standard benchmark
  - Evaluate different systems on common workloads
  - Focus on performance and scale out
    - Future additions – availability, replication
- Artifacts
  - Open source workload generator
  - Experimental study comparing several systems

Benchmark tool

- Java application
  - Many systems have Java APIs
  - Other systems via HTTP/REST, JNI or some other solution



Benchmark tiers

- Tier 1 – Performance
  - Latency versus throughput as throughput increases
  - "Sizeup": the hardware is kept constant but the size of the workload increases.
- Tier 2 – Scalability
  - Latency as database, system size increases
  - "Scaleup": How does the database perform as the number of machines increases?

  - Latency as we elastically add servers
  - "Elastic speedup": How does the database perform as the number of machines increases while the system is running?

Test setup

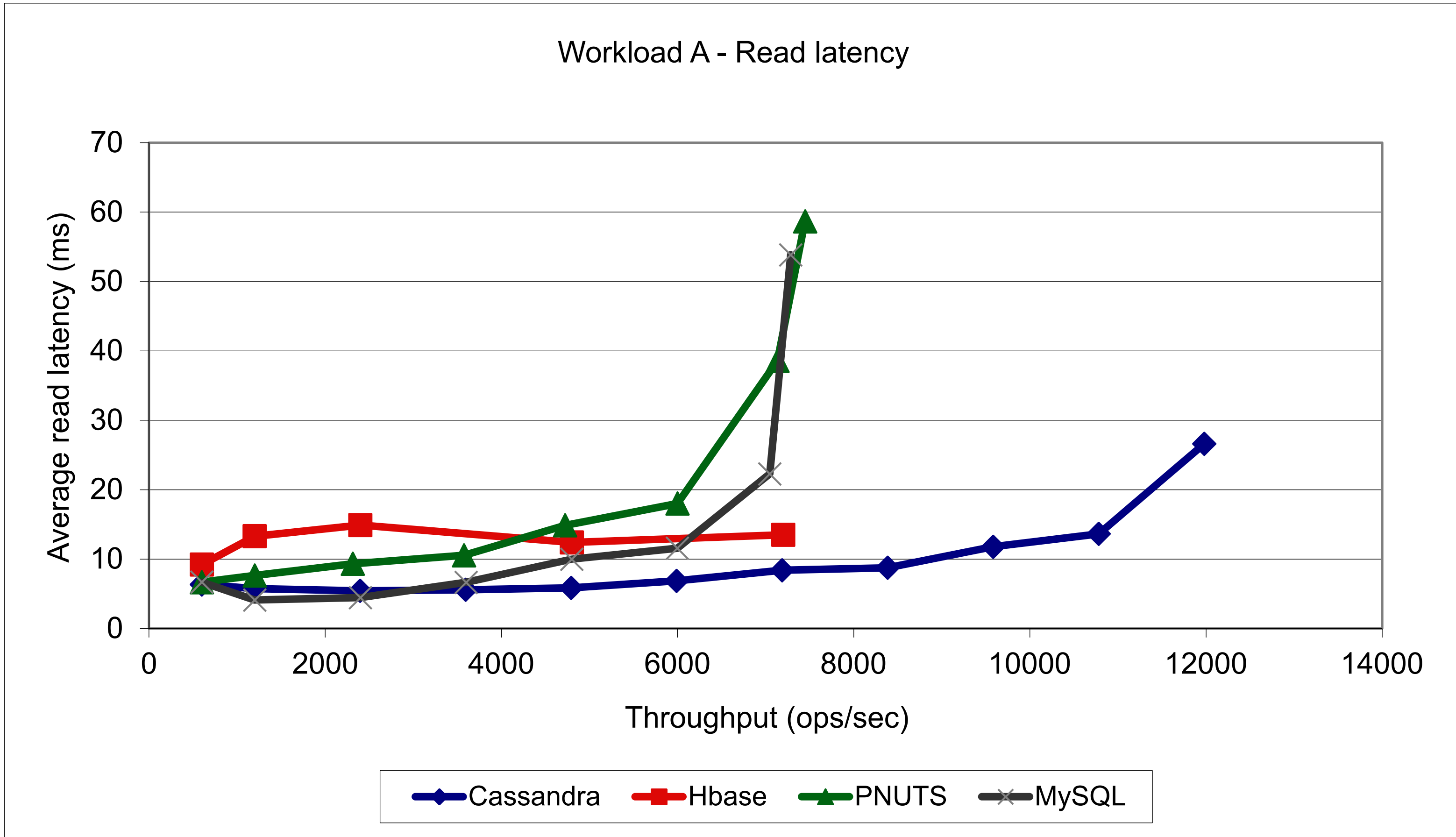
- Setup
  - Six server-class machines
    - 8 cores (2 x quadcore) 2.5 GHz CPUs, 8 GB RAM, 6 x 146GB 15K RPM SAS drives in RAID 1+0, Gigabit ethernet, RHEL 4
  - Plus extra machines for clients, routers, controllers, etc.
  - Cassandra 0.5.0 (0.6.0-beta2 for range queries)
  - HBase 0.20.3
  - MySQL 5.1.32 organized into a sharded configuration
  - PNUTS/Sherpa 1.8 with MySQL 5.1.24
  - No replication; force updates to disk (except HBase, which primarily commits to memory)
- Workloads
  - 120 million 1 KB records = 20 GB per server
- Caveat
  - We tuned each system as well as we knew how, with assistance from the teams of developers

Basic operations

- Each operation against the data store
  - **Insert**: Insert a new record.
  - **Update**: Update a record by replacing the value of one field.
  - **Read**: Read a record, either one randomly chosen field or all fields.
  - **Scan**: Scan records in order, starting at a randomly chosen record key. The number of records to scan is randomly chosen.

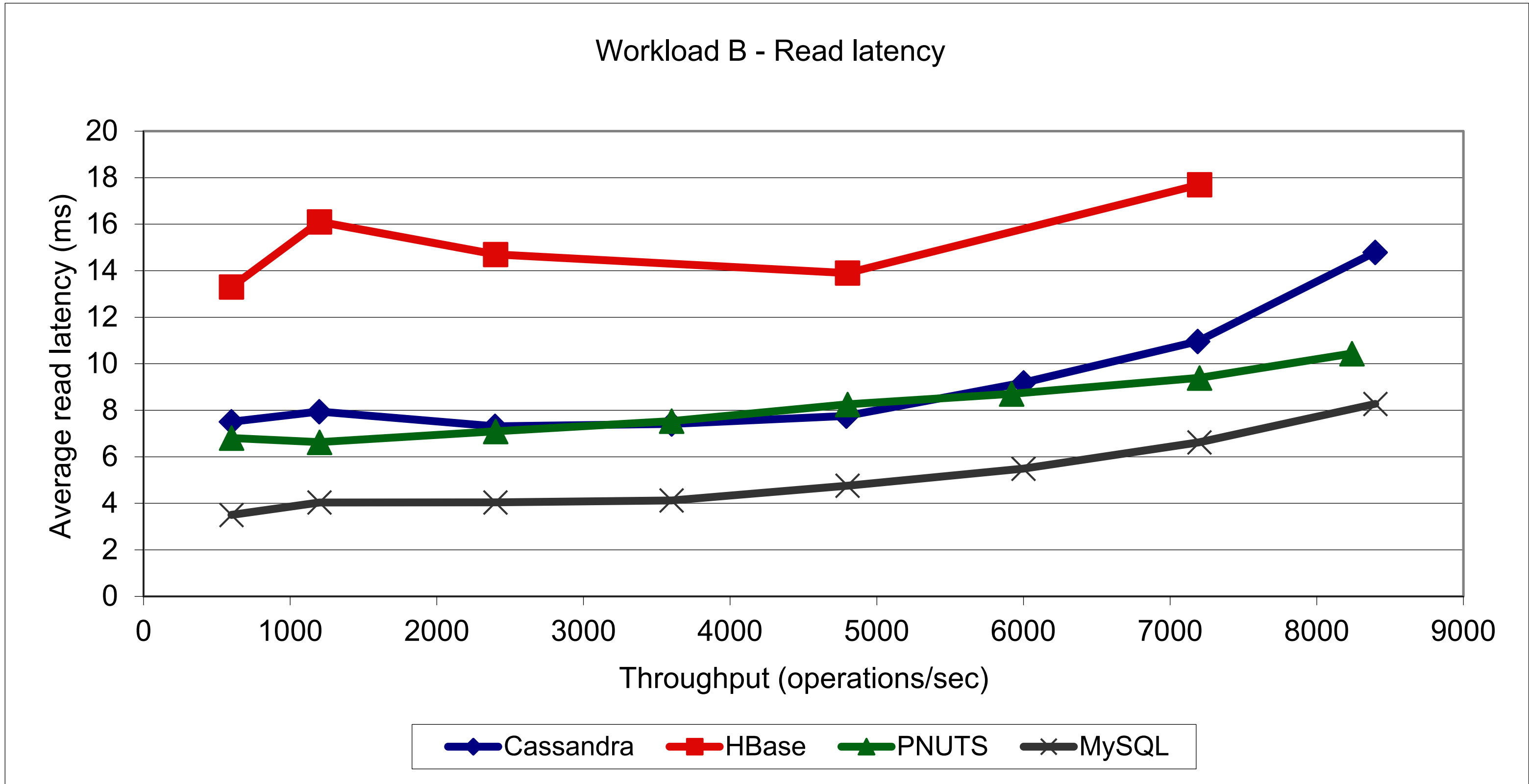
Workload A – Update heavy

- 50/50 Read/update



Workload B – Read heavy

- 95/5 Read/update

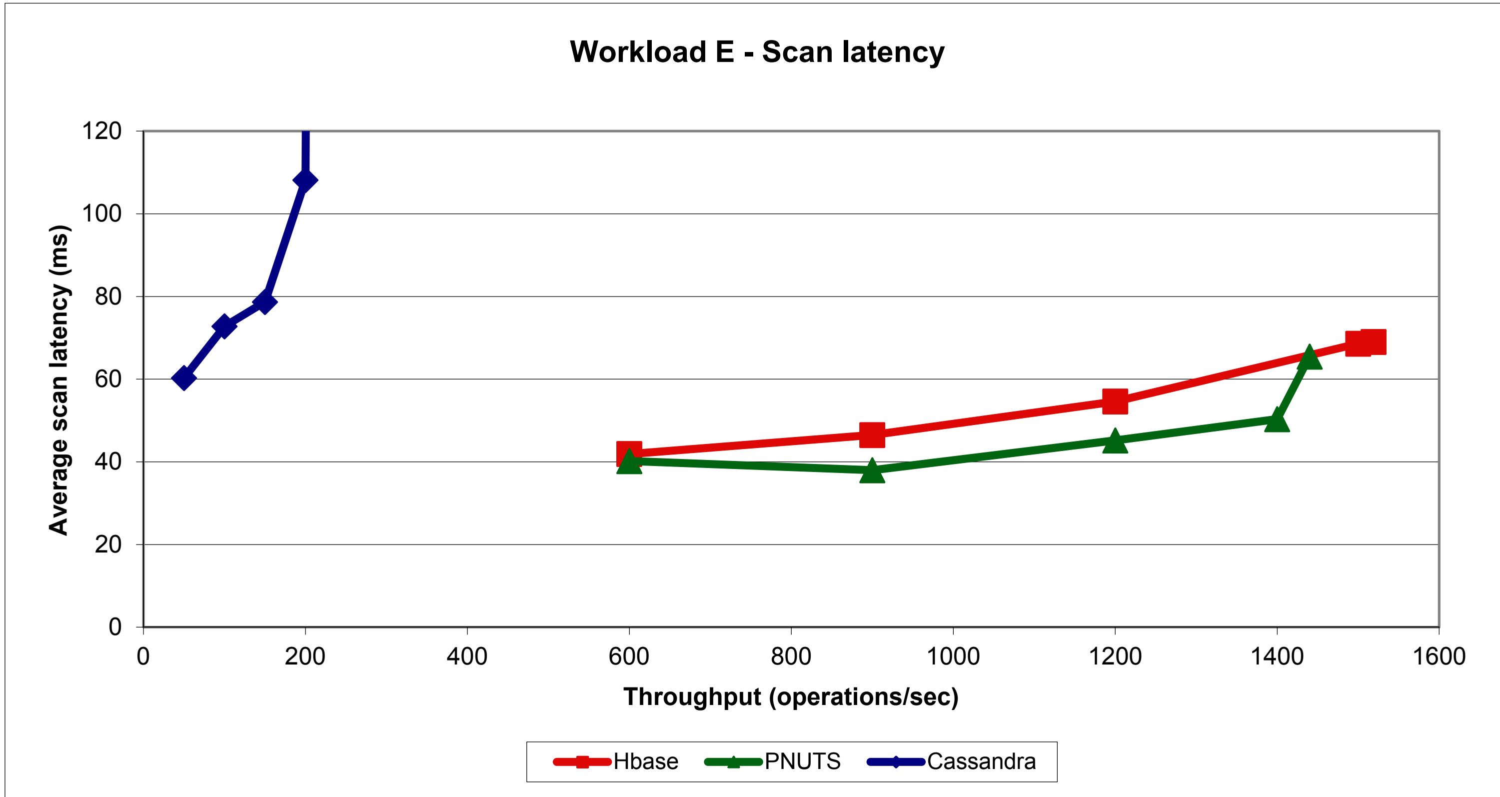


Why Hbase performs poorly?

- Records on disk are never overwritten
  - instead, updates are written to a buffer in memory, and the entire buffer is written sequentially to disk.
- Multiple updates to the same record may be flushed at different times to different parts of the disk.
- The result is that to perform a read of a record, multiple I/Os are needed to retrieve and combine the various updates.

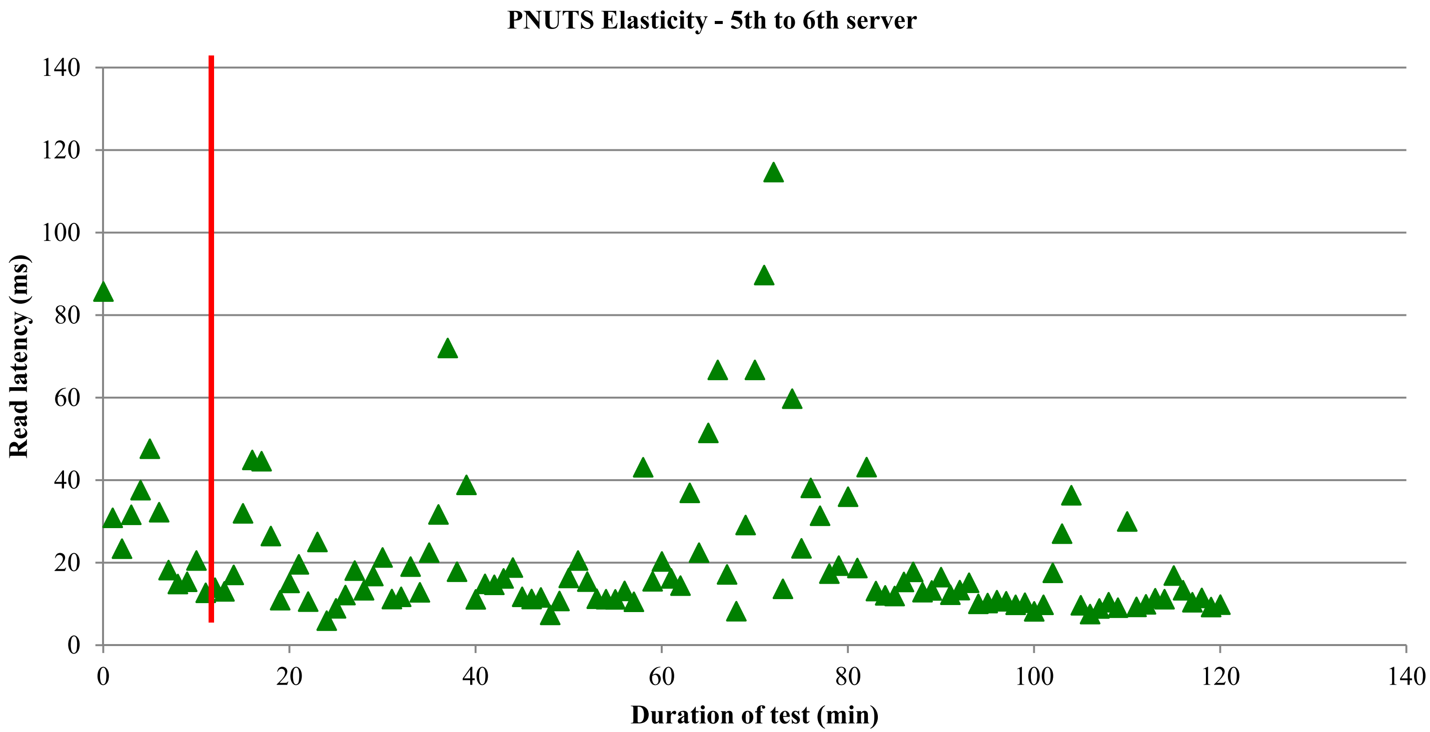
Workload E – short scans

- Scans of 1-100 records of size 1KB



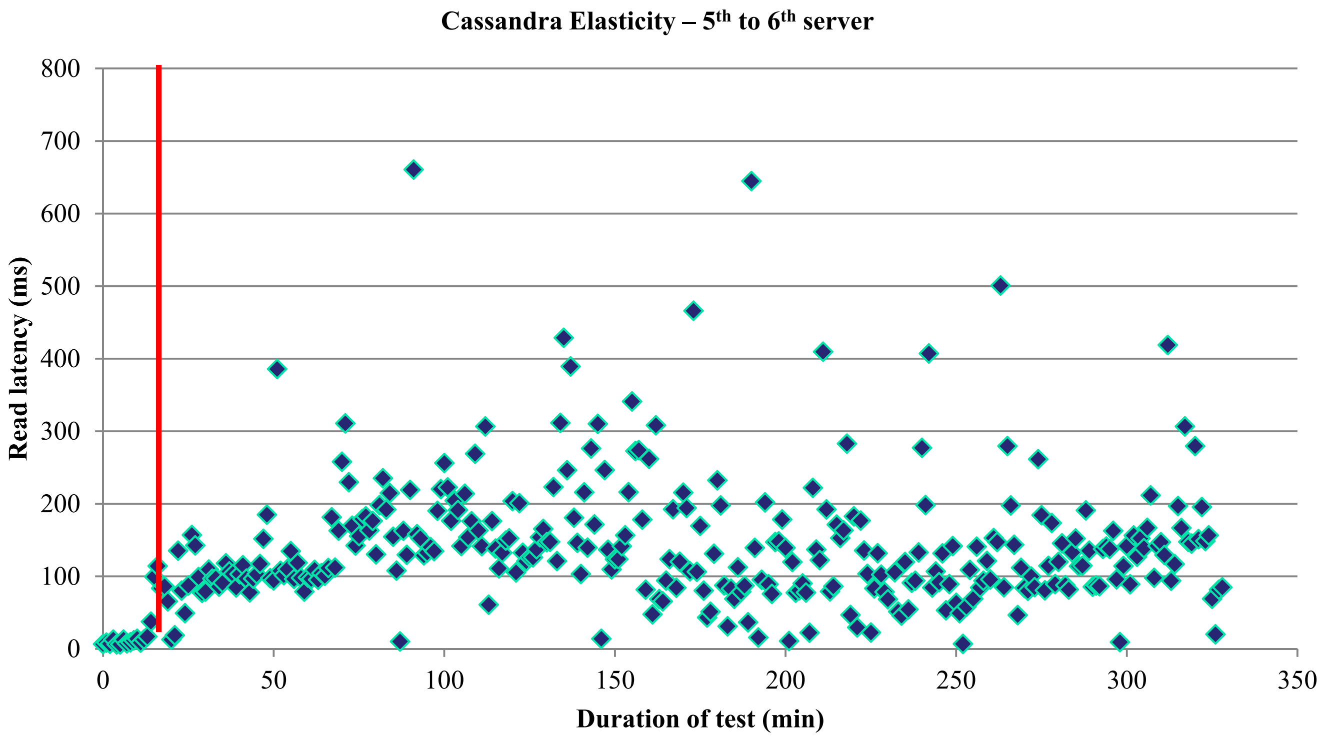
Elasticity

- Run a read-heavy workload on 2 servers; add a 3rd, then 4th, then 5th, then 6th server.



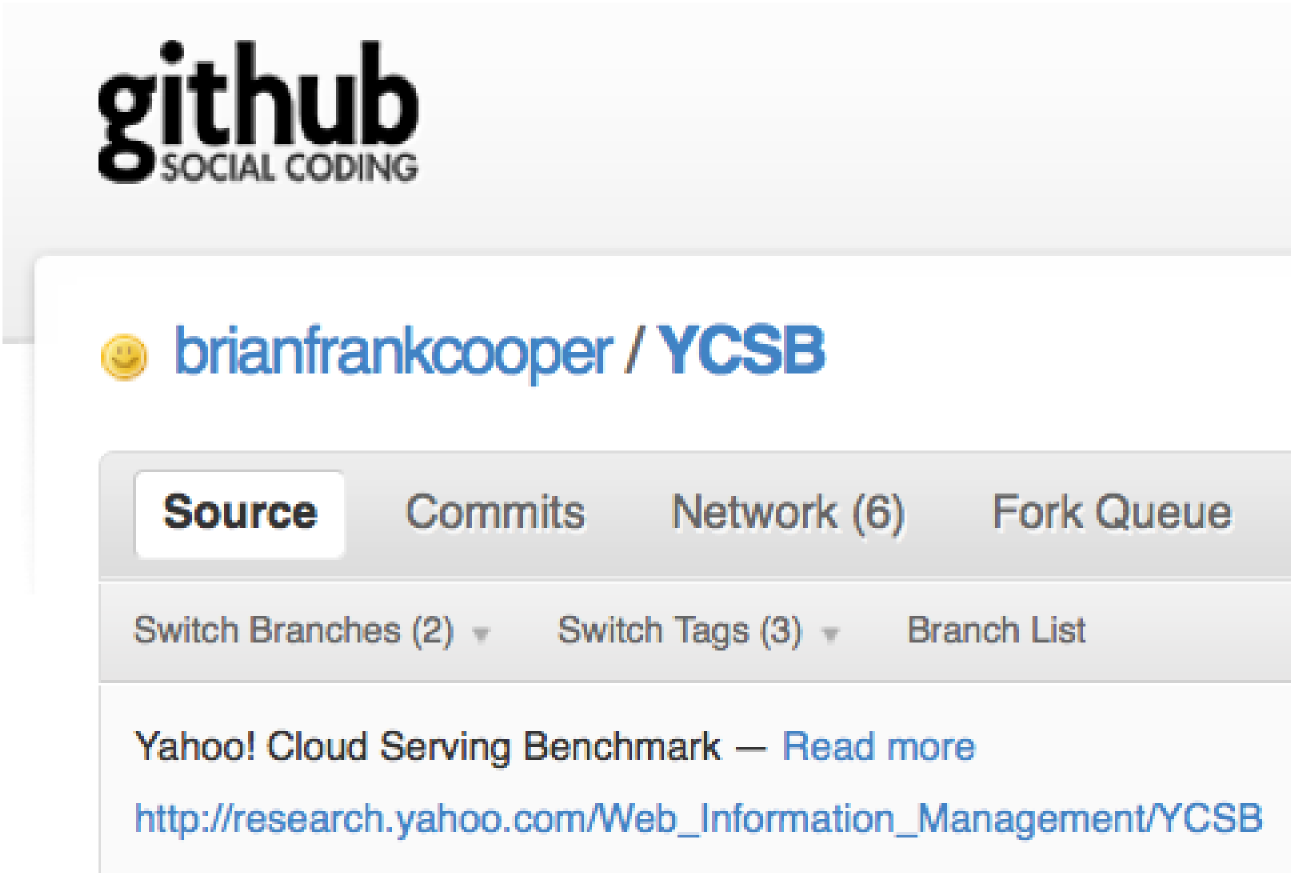
Elasticity

- Run a read-heavy workload on 2 servers; add a 3rd, then 4th, then 5th, then 6th server.



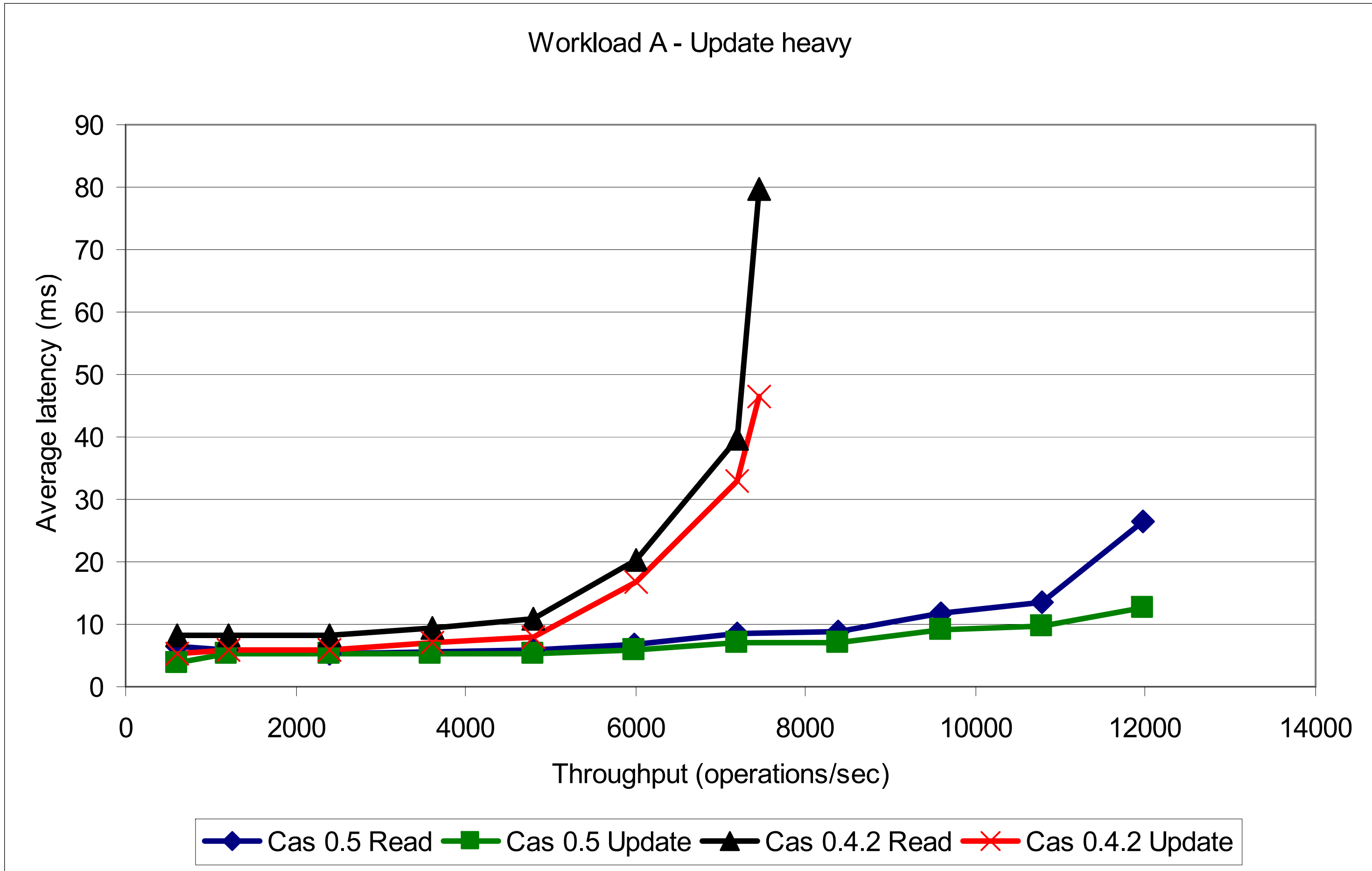
Experiences

- The benefits of an open-source toolkit



Experiences

- The rapid evolution of cloud systems



Future work

- Tiers for replication, fault tolerance
- More database bindings
- More scenarios
- More expressive experimental setups

Conclusions

- YCSB is an open benchmark for cloud serving systems
- Experimental results show tradeoffs between systems
- The benchmark (and the systems themselves) are evolving

Reading list

- Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10). ACM, New York, NY, USA, 143-154.

- http://wiki.github.com/brianfrankcooper/YCSB/

谢谢