copenhagen goto; conference

EVALUATING NOSQL PERFORMANCE: TIME FOR BENCHARKING

Sergey Bushik
Senior RnD Engineer, Altoros

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

Agenda

- Introduction
- Few lines about benchmarking client
- Workloads
- Cluster setup
- Meet evaluated databases: short stop on each
- Diagrams and comments
- Summary
- Questions

Introduction

- ✓ Database's list is extensive (RDBMS 90+, NoSQL 122+)
- ✓ Core NoSQL systems types
  - Key value stores
  - Document oriented stores
  - Column family stores
  - Graph databases
  - XML databases
  - Object database management systems
- ✓ Different APIs and clients
- ✓ Different performance

How do they compare?

- ✓ Yahoo! team offered “standard” benchmark
- ✓ Yahoo! Cloud Serving Benchmark (YCSB)
  - Focus on database
  - Focus on performance
- ✓ YCSB Client consists of 2 parts
  - Workload generator
  - Workload scenarios

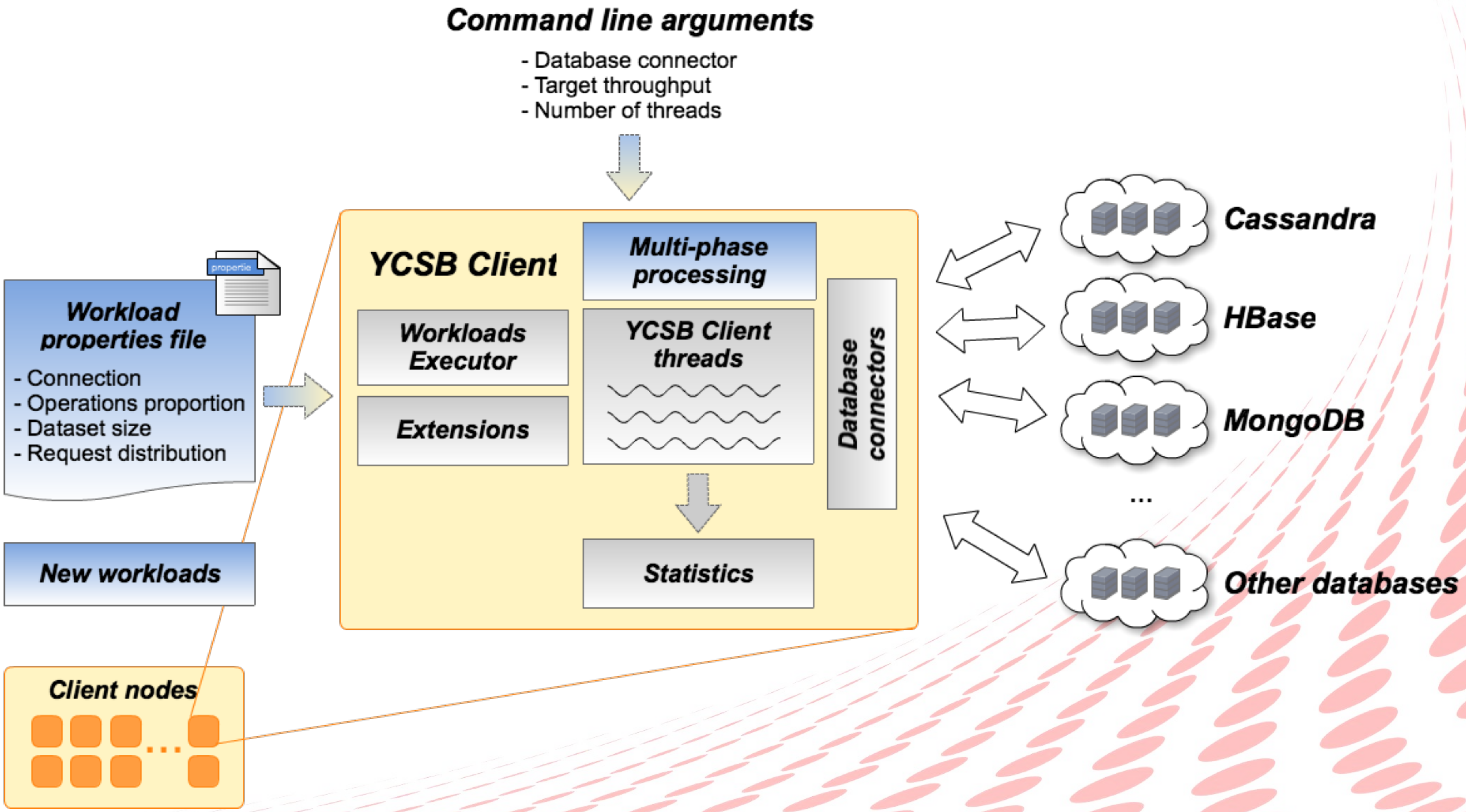
YCSB features

- ✓ Open source
- ✓ Extensible
- ✓ Has connectors

  Azure, BigTable, Cassandra, CouchDB,
  Dynomite, GemFire, HBase, Hypertable,
  Infinispan, MongoDB, PNUTS, Redis,
  Connector for Sharded RDBMS (i.e. MySQL),
  Voldemort, GigaSpaces XAP
- ✓ We developed few connectors

  Accumulo, Couchbase, Riak,
  Connector for Shared Nothing RDBMS (i.e. MySQL Cluster)

YCSB architecture



Workloads

- ✓ Workload is a combination of key-values:
  - Request distribution (uniform, zipfian)
  - Record size
  - Operation proportion (%)
- ✓ Types of workload phases:
  - Load phase
  - Transaction phase

Workloads

- ✓ Load phase workload
  - Working set is created
  - 100 million records
  - 1 KB record (10 fields by 100 Bytes)
  - 120-140G total or ≈30-40G per node

- ✓ Transaction phase workloads
  - Workload A (read/update ratio: 50/50, zipfian)
  - Workload B (read/update ratio: 95/5, zipfian)
  - Workload C (read ratio: 100, zipfian)
  - Workload D (read/update/insert ratio: 95/0/5, zipfian)
  - Workload E (read/update/insert ratio: 95/0/5, uniform)
  - Workload F (read/read-modify-write ratio: 50/50, zipfian)
  - Workload G (read/insert ratio: 10/90, zipfian)

Cluster setup

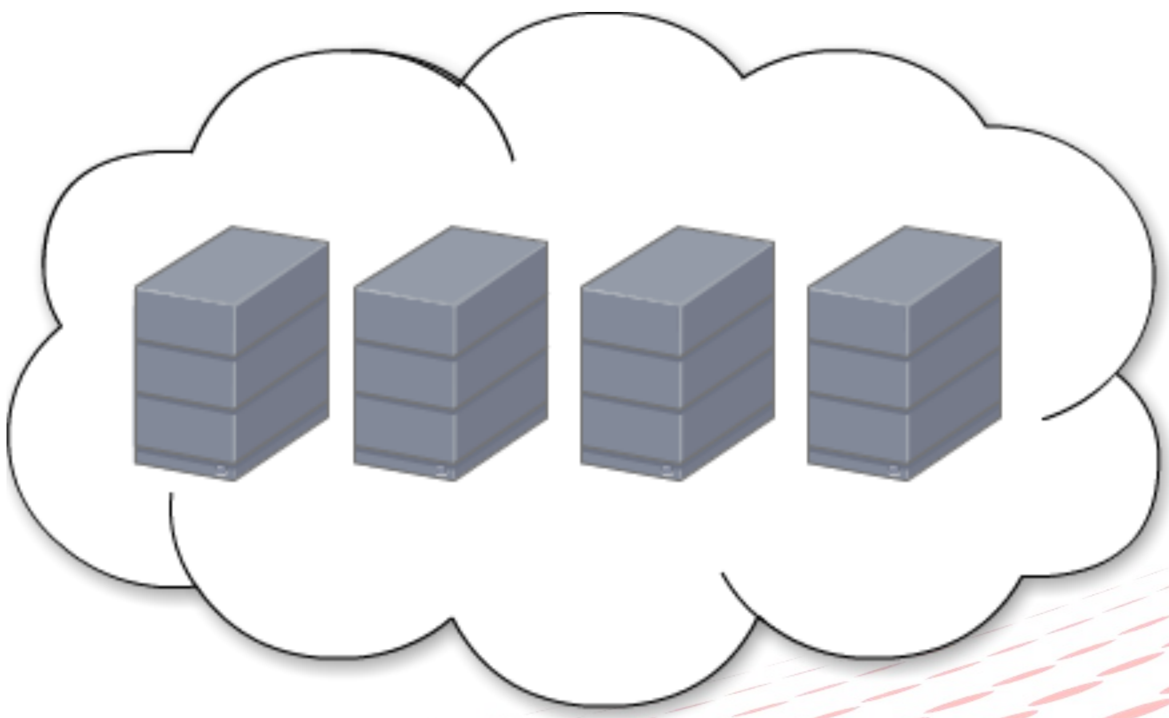
- ✓ Amazon EC2 as a cluster infrastructure
- ✓ No replication (replication factor = 0)
- ✓ EBS volumes in RAID0 (stripping) array for data storage directory
- ✓ OS swapping is OFF

Cluster specification

YCSB Client

Amazon m1.large Instance

7.5 GB memory
2 virtual cores
8 GB instance storage
64-bit Amazon Linux (CentOS binary compatible)



Amazon m1.xlarge Instances * 4

15 GB memory
4 virtual cores
4 EBS 50 GB volumes in RAID0
64-bit Amazon Linux

* Extra nodes for masters, routers, etc

Databases

The list of databases

- Cassandra 1.0
- HBase 0.92.0
- MongoDB 2.0.5
- MySQL Cluster 7.2.5
- MySQL Sharded 5.5.2.3
- Riak 1.1.1

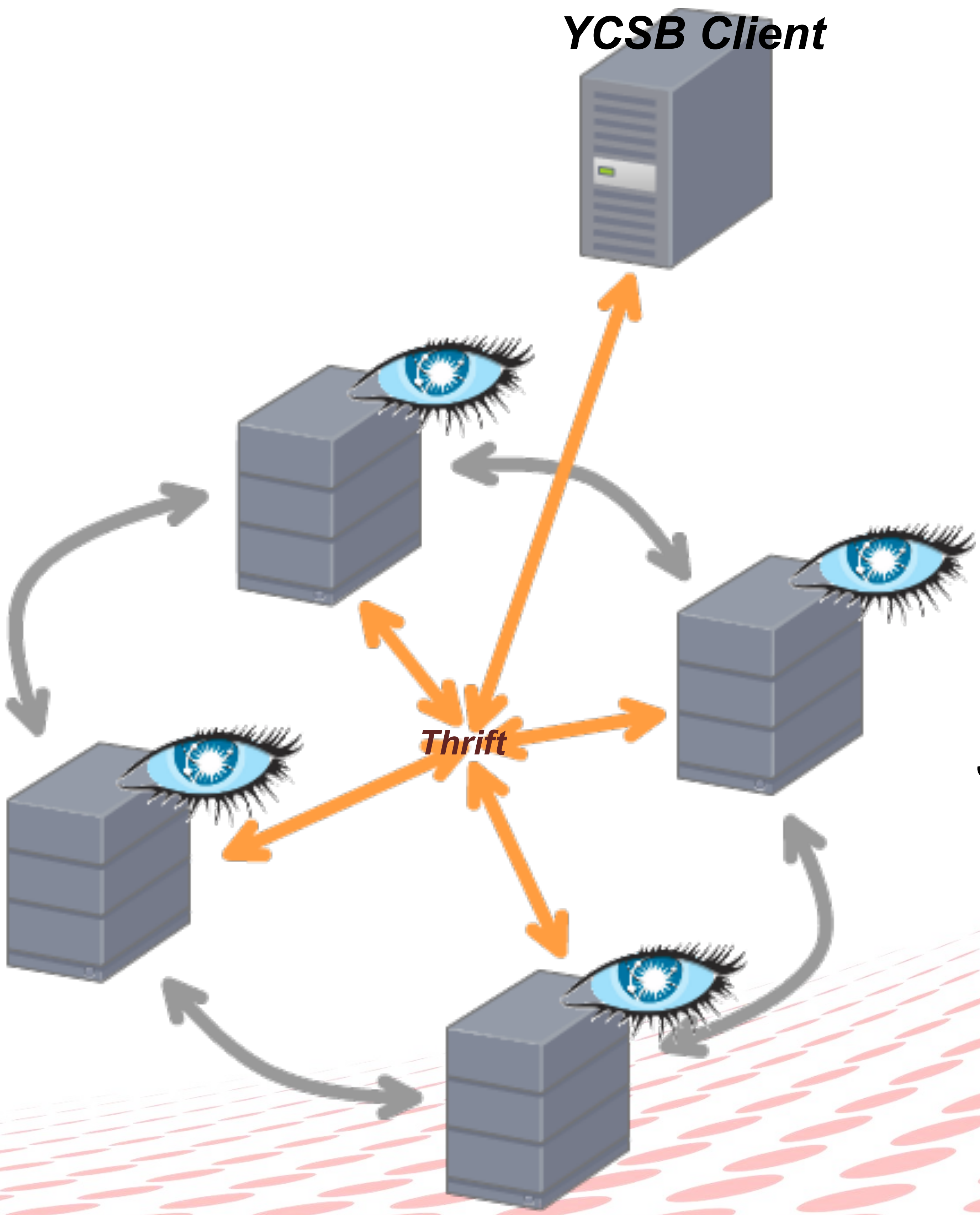
We tuned each system as well as we knew how

Let's see who is worth the prize

Cassandra 1.0

- ✓ Column-oriented
- ✓ No single point of failure
- ✓ Distributed
- ✓ Elastically scalable
- ✓ Tuneably consistent
- ✓ Caching
    - Key cache
    - Off-heap/on-heap row cache
    - Memory mapped files

Cassandra 1.0



Cassandra configuration

Random partitioner
Initial token space: 2^127 / 4
Memtable space: 4G
Commit log is on the separate disk
Concurrent reads: 32 (8 * 4 cores)
Concurrent writes: 64 (16 * 4 disks)
Compression: Snappy

JVM tuning

MAX_HEAP_SIZE: 6G
HEAP_NEWSIZE: 400M
Rest of 15G RAM is for OS caching

Cassandra 1.0

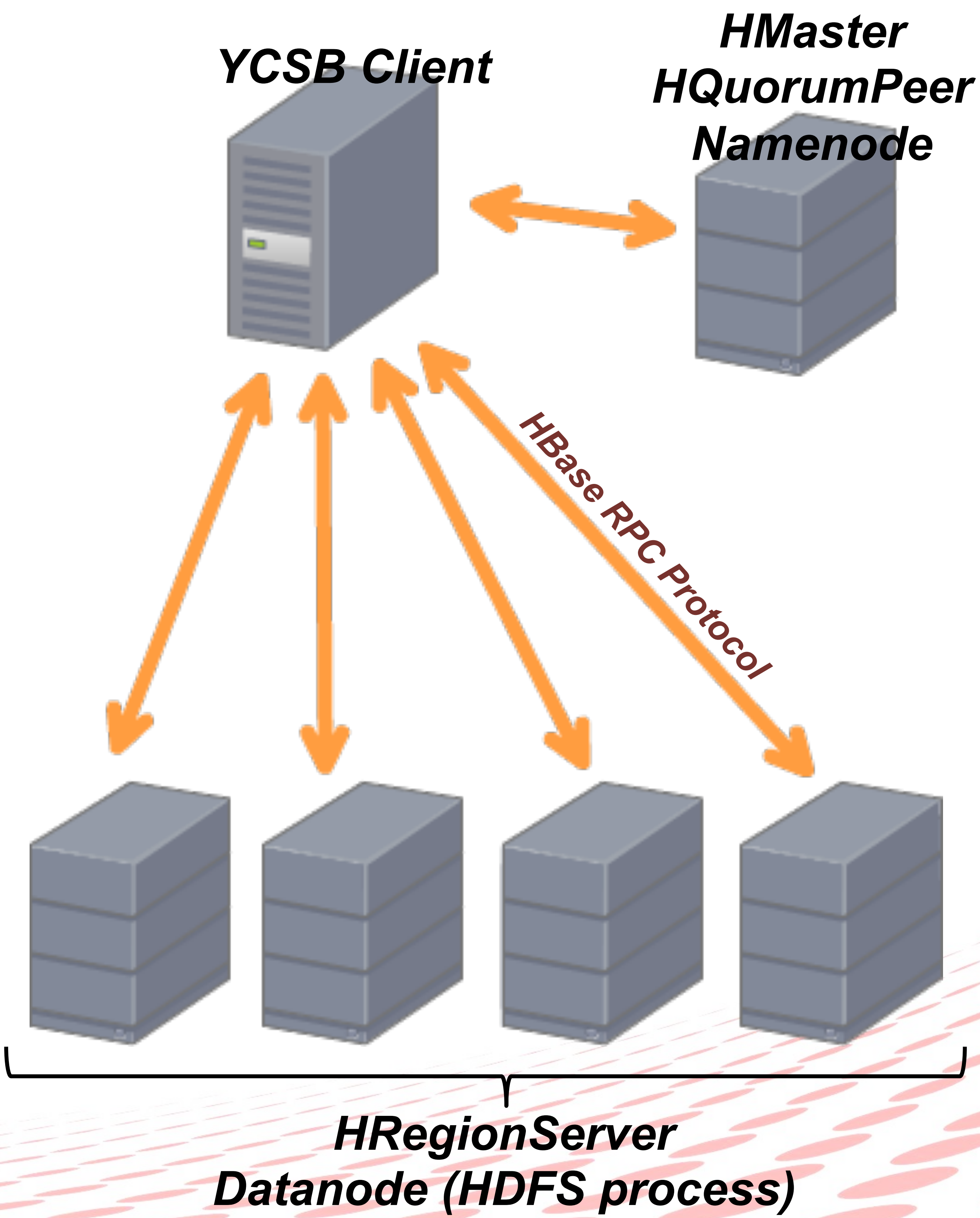
```
CREATE KEYSPACE UserKeyspace WITH
        placement_strategy = 'SimpleStrategy' AND
        strategy_options = {replication_factor:1} AND durable_writes = true;

USE UserKeyspace;

CREATE COLUMN FAMILY UserColumnFamily WITH
        comparator = UTF8Type AND
        key_validation_class = UTF8Type AND
        keys_cached = 100000000 AND
        rows_cached = 1000000 AND
        row_cache_provider = 'SerializingCacheProvider’ AND
        compression_options = {sstable_compression:SnappyCompressor};
```

* Make sure you use off-heap row caching! (it requires JNA library)

HBase 0.92.0

- ✓ Column-oriented
- ✓ Distributed
- ✓ Built on top of Hadoop DFS (Distributed File System)
- ✓ Block cache
- ✓ Bloom filters on per-column family basis
- ✓ Consistent reads/writes

HBase 0.92.0



HBase configuration

Auto flush: OFF

Write buffer: 12M (2M default)

Compression: LZO

JVM tuning

Namenode: 4G

Datanode: 2G

Region server: 6G

Master: 2G

HBase 0.92.0

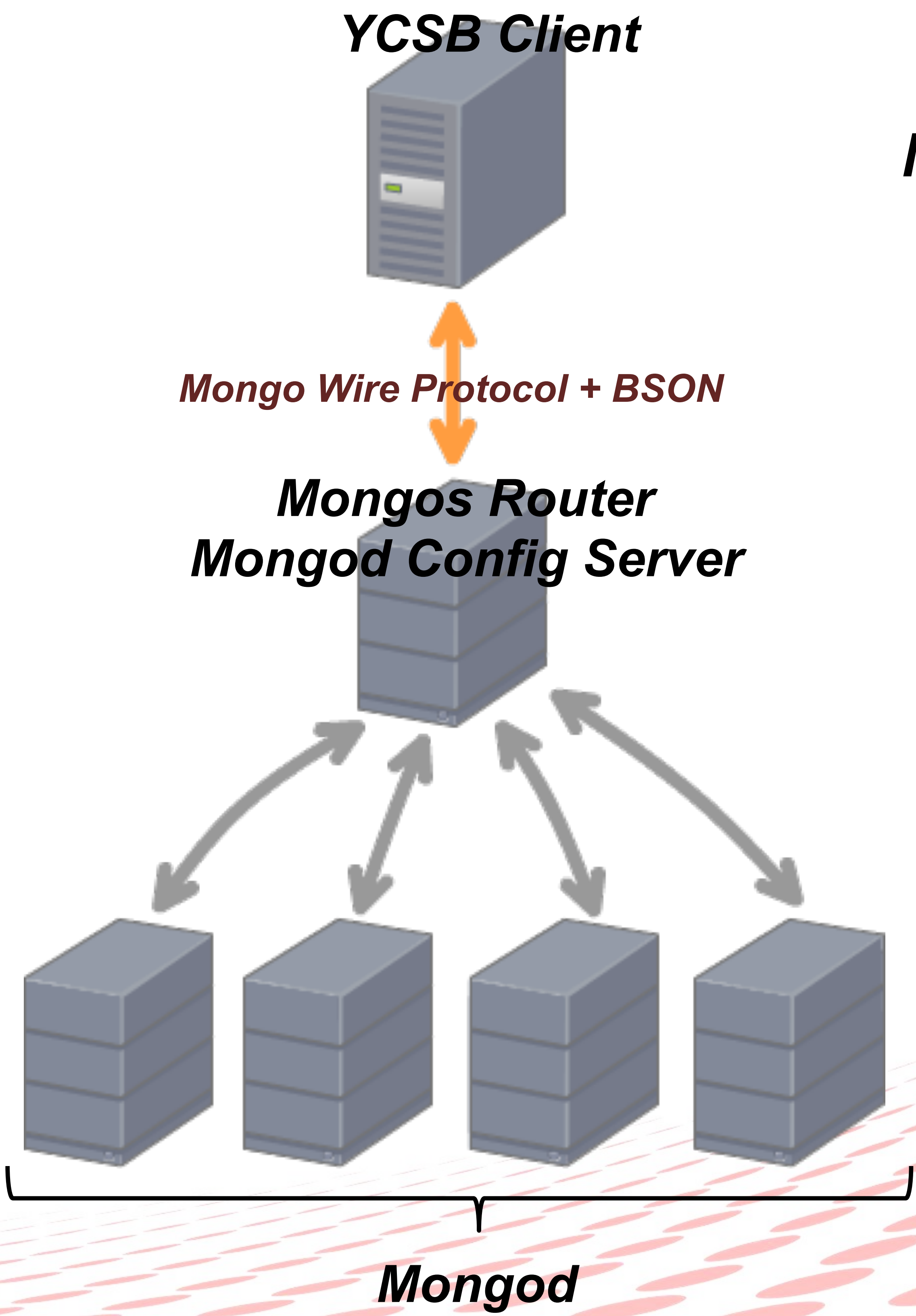
```
CREATE 't1', {
        NAME => 'f1', BLOOMFILTER => 'ROW',
        REPLICATION_SCOPE => '0', COMPRESSION => 'LZO',
        VERSIONS => '3', TTL => '2147483647', BLOCKSIZE => '16384',
        IN_MEMORY => 'true', BLOCKCACHE => 'true',
        DEFERRED_LOG_FLUSH => 'true’
}
DISABLE 't1'
ALTER 't1', METHOD => 'table_att', MAX_FILESIZE => '1073741824'
ENABLE 't1'
```

* Make column family name short (several chars)

MongoDB 2.0.5

- ✓ Document-oriented
- ✓ Distributed
- ✓ Load balancing by sharding
- ✓ Relies on memory mapped files for caching

MongoDB 2.0.5



Mongo configuration

Sharding enabled on database
Collection is sharded by _key (PK)

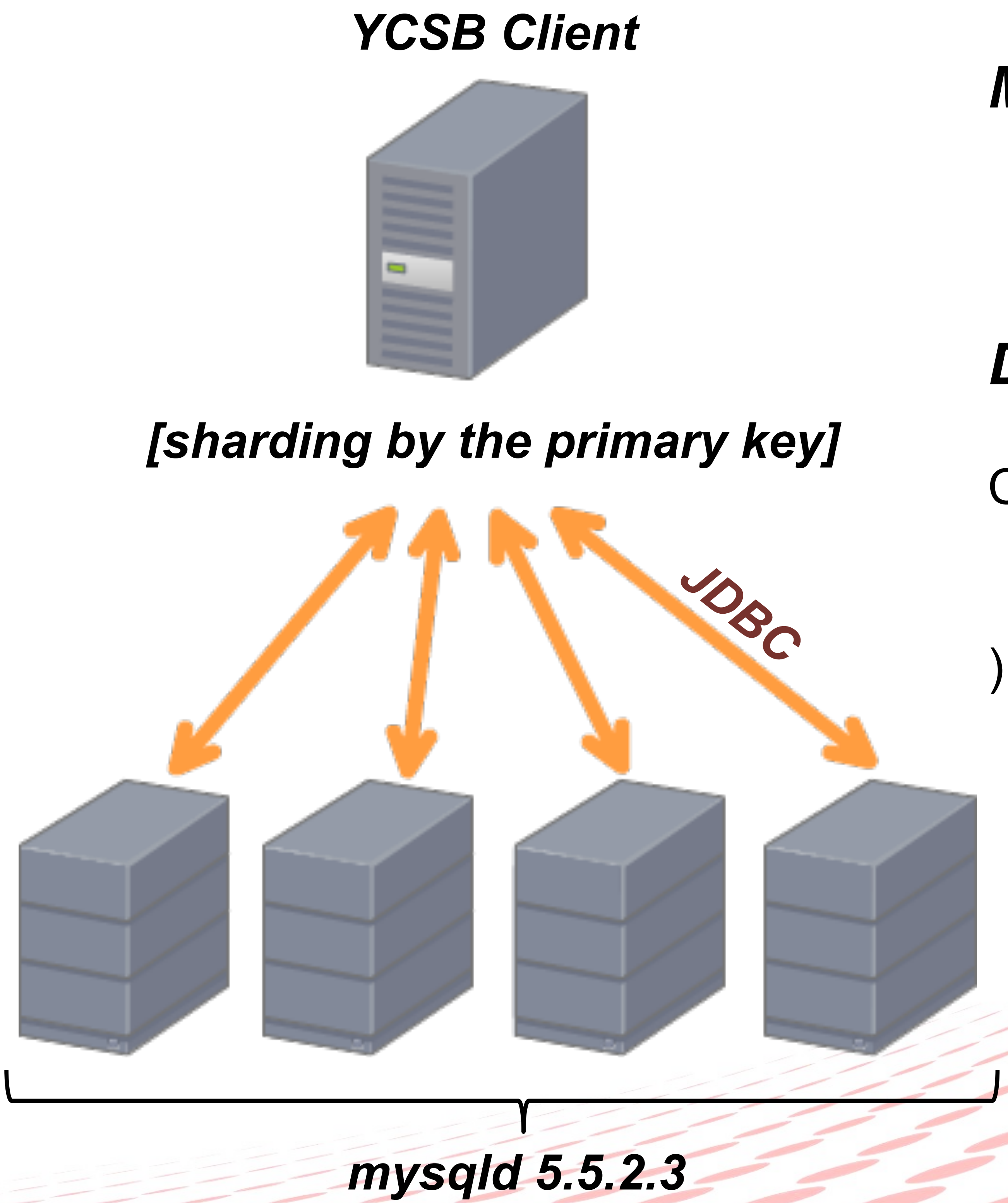
MongoDB 2.0.5

```
// use hostnames instead of IPs
var shards = […];
// adding shards
for (var i = 0; i < shards.length; i++) {
    db.runCommand({ addshard : shards[i] });
}
// enabling sharding
db.runCommand({ enablesharding : "UserDatabase” });
// sharding the collection by key
db.runCommand({ shardcollection : "UserDatabase.UserTable", key : { _id : 1 } });
```

MySQL Sharded 5.5.2.3

- ✓ RDBMS (no surprises here)
- ✓ Sharding is done on the client (YCSB) side
- ✓ Not scalable

MySQL Sharded 5.5.2.3



MySQL Configuration

Storage engine: MyISAM

key_buffer_size: 6G

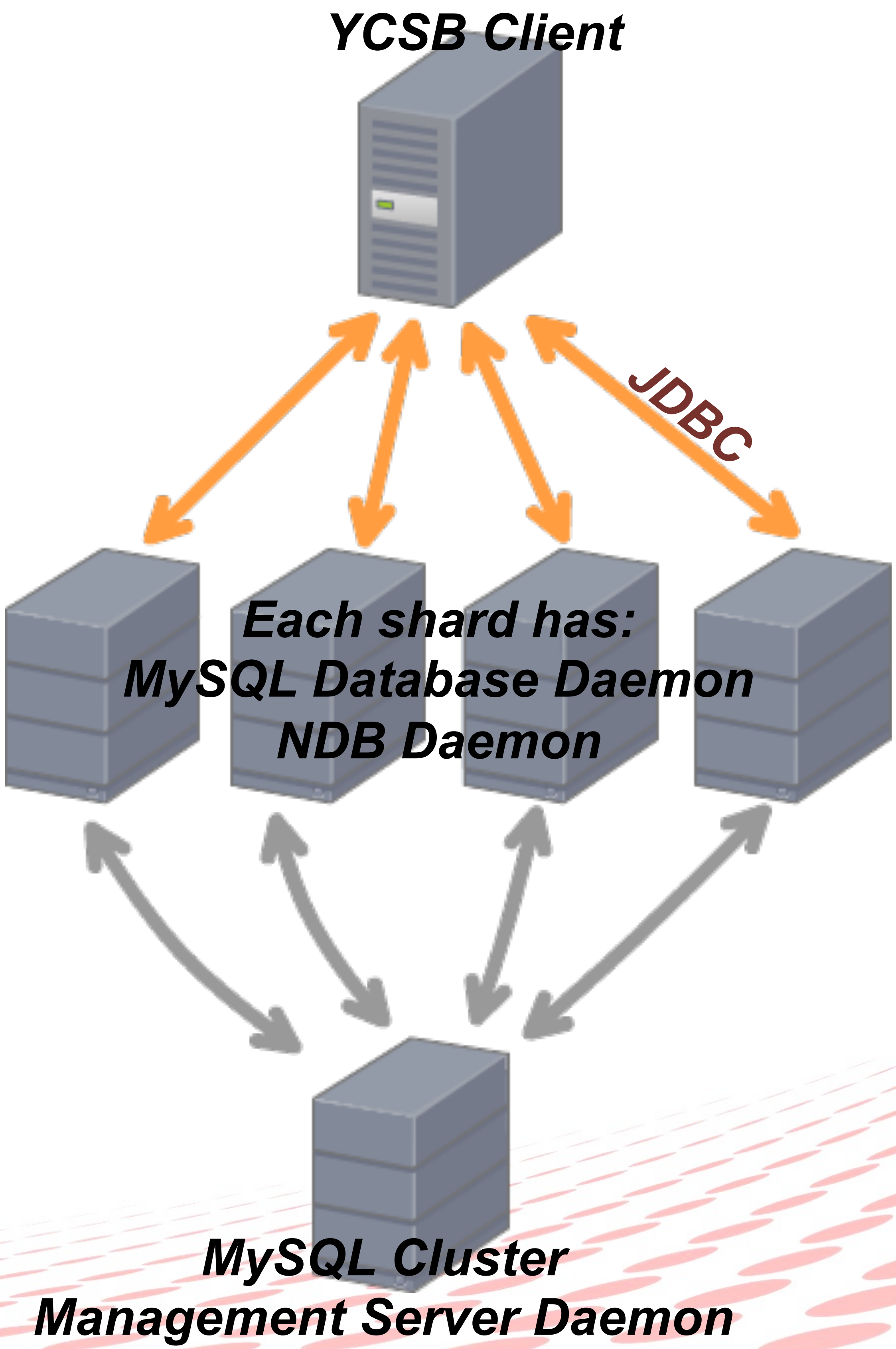
DDL for table creation

```
CREATE TABLE user_table(
  ycsb_key VARCHAR(32) PRIMARY KEY,
  // specify 10 table columns (100 bytes each)
) ENGINE=MYISAM;
```

MySQL Cluster 7.2.5

- ✓ Relational
- ✓ Not really relational
  - No foreign keys
  - ACID: read committed transactions only
- ✓ Shared-nothing
- ✓ In-memory database
- ✓ Can be persistent (non-indexed columns)

MySQL Cluster 7.2.5



MySQL Cluster Configuration

DataMemory: 3G
IndexMemory: 5G
DiskPageBufferMemory: 2G

MySQL Cluster 7.2.5

CREATE TABLE user_table(
 ycsb_key VARCHAR(32) PRIMARY KEY,
 … // columns
 MAX_ROWS=200000000 **ENGINE**=NDBCLUSTER **PARTITION BY KEY**(ycsb_key);

CREATE LOGFILE …; // log files are created

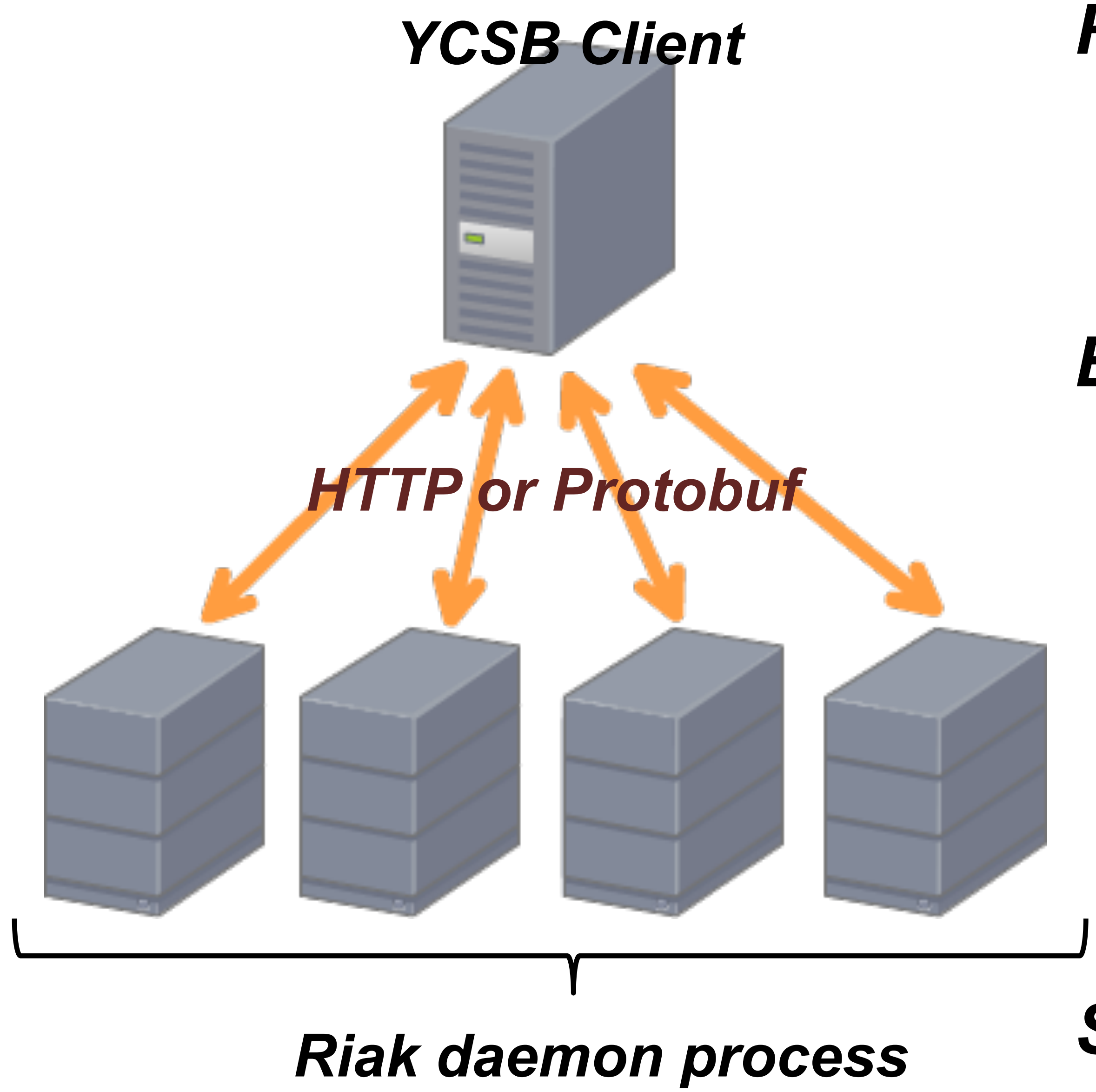
CREATE TABLESPACE …; // table space for disk persistence

ALTER TABLE user_table
  TABLESPACE user_table_space
  STORAGE DISK
  ENGINE=NDBCLUSTER; // assigning table space with target table

Riak 1.1.1

- ✓ Key-value storage (Amazon's Dynamo inspired)
- ✓ Distributed
- ✓ Scalable
- ✓ Schema free
- ✓ Decentralized (no single point of failure)

Riak 1.1.1



Riak Configuration

storage_backend: bitcask

// eleveldb backend was slow

Erlang (vm.arg) Configuration

// number of threads in async thread pool

+A 256

// kernel poll enabled

+K true

// number of concurrent ports and sockets

-env ERL_MAX_PORTS 4096

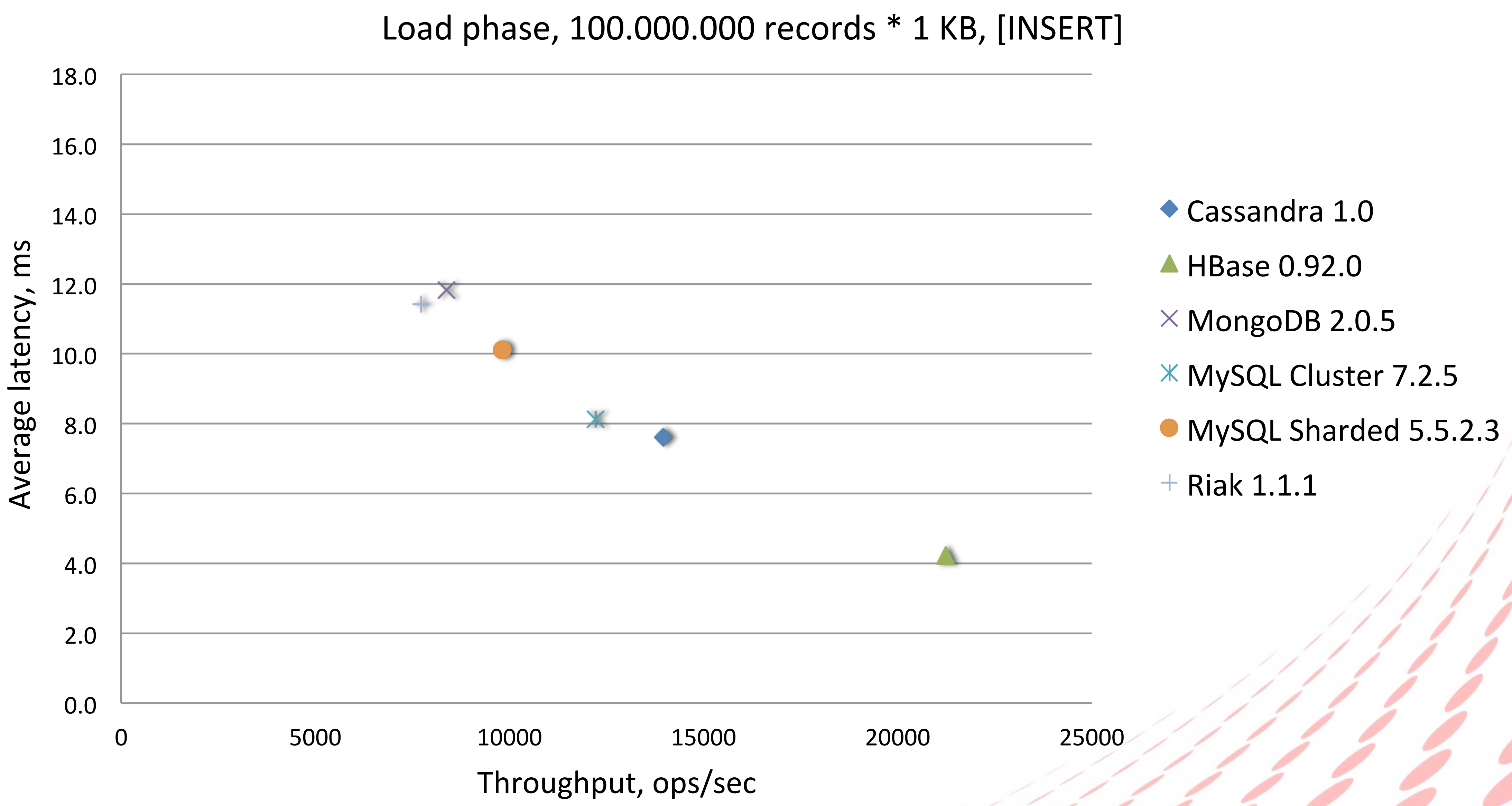
Schema\DDL

Is not required, just a bucket name

Load phase, [INSERT]

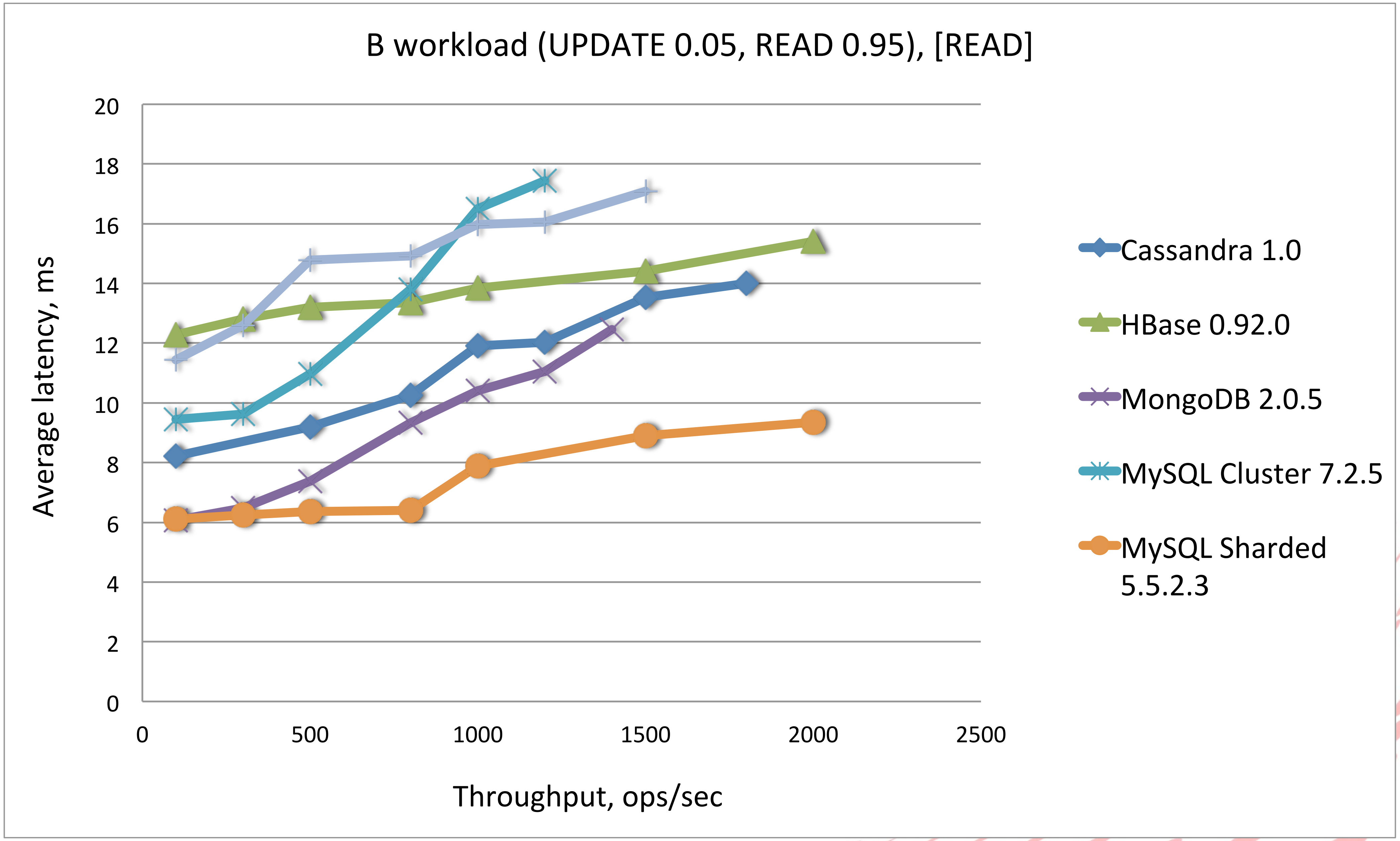
copenhagen goto; conference

ALTOROS



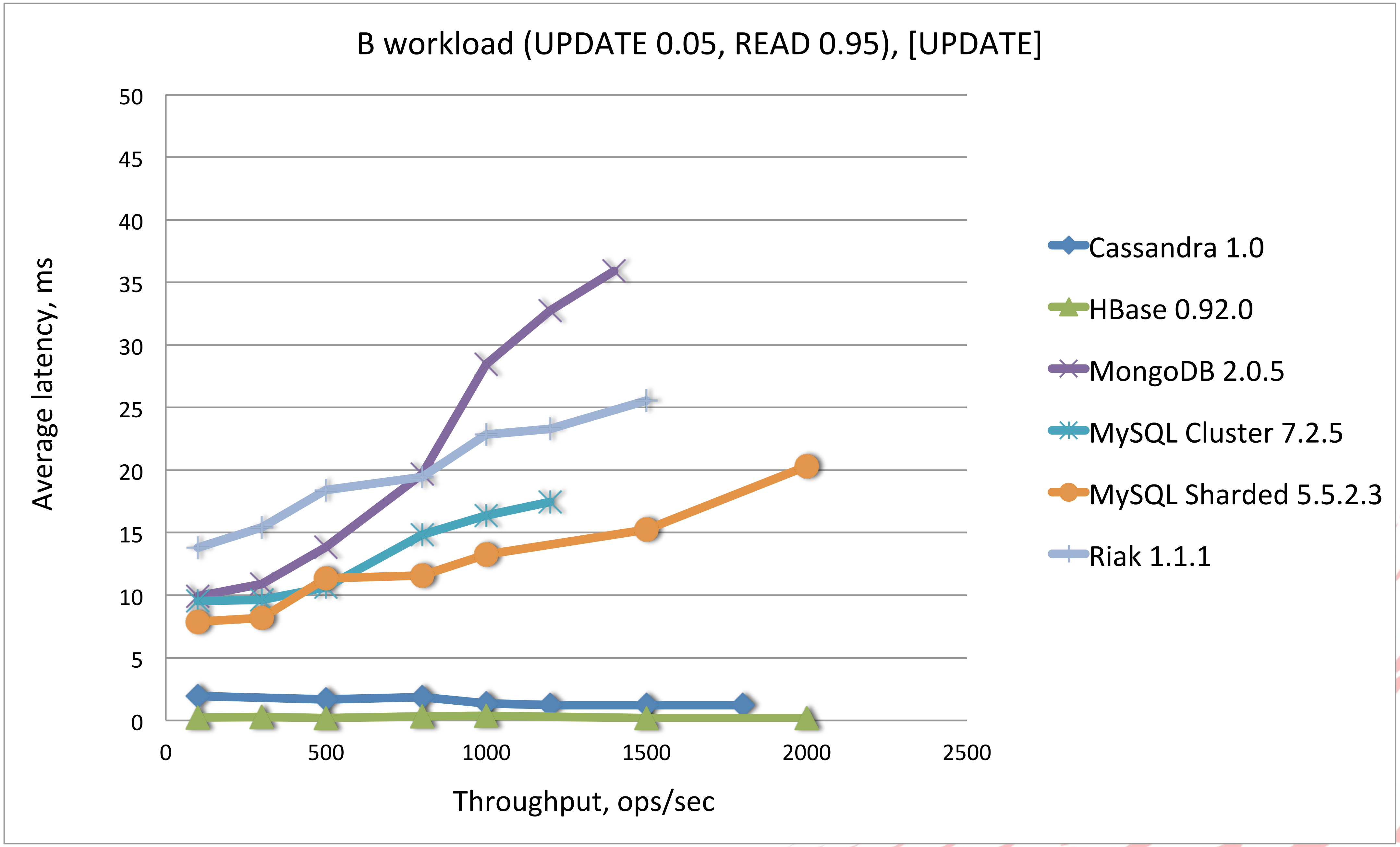
HBase has unconquerable superiority in writes, and with a pre-created regions it showed us up to 40K ops/sec. Cassandra also provides noticeable performance during loading phase with around 15K ops/sec. MySQL Cluster can show much higher numbers in “just in-memory” mode.

Read heavy workload (B), [READ]



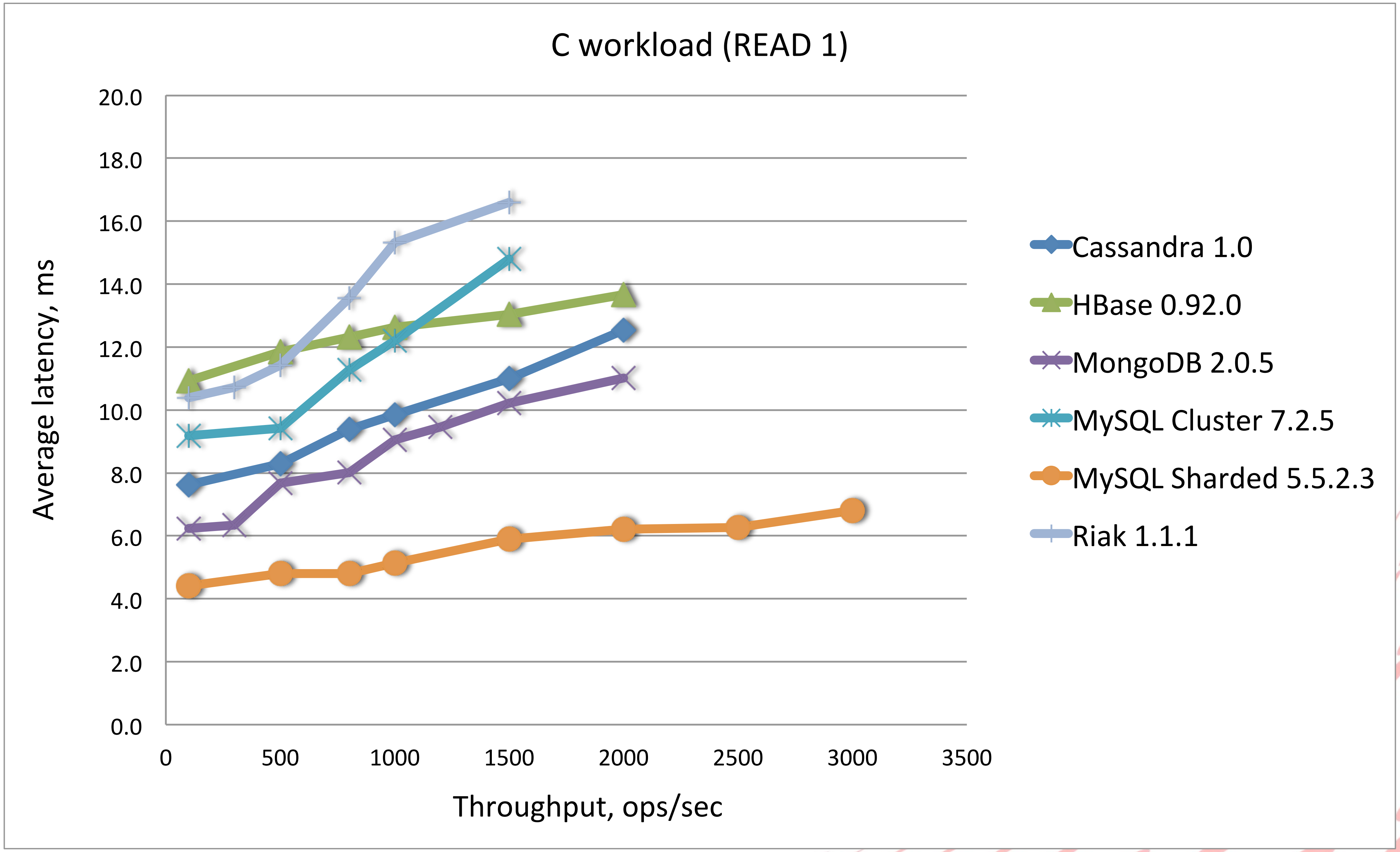
MySQL Sharded is a performance leader in reads. MongoDB is close to it accelerated by the "memory mapped files" type of cache. MongoDB uses memory-mapped files for all disk I/O. Cassandra's key and row caching allows very fast access to frequently requested data. Random read performance is slower in HBase.

Read heavy workload (B), [UPDATE]



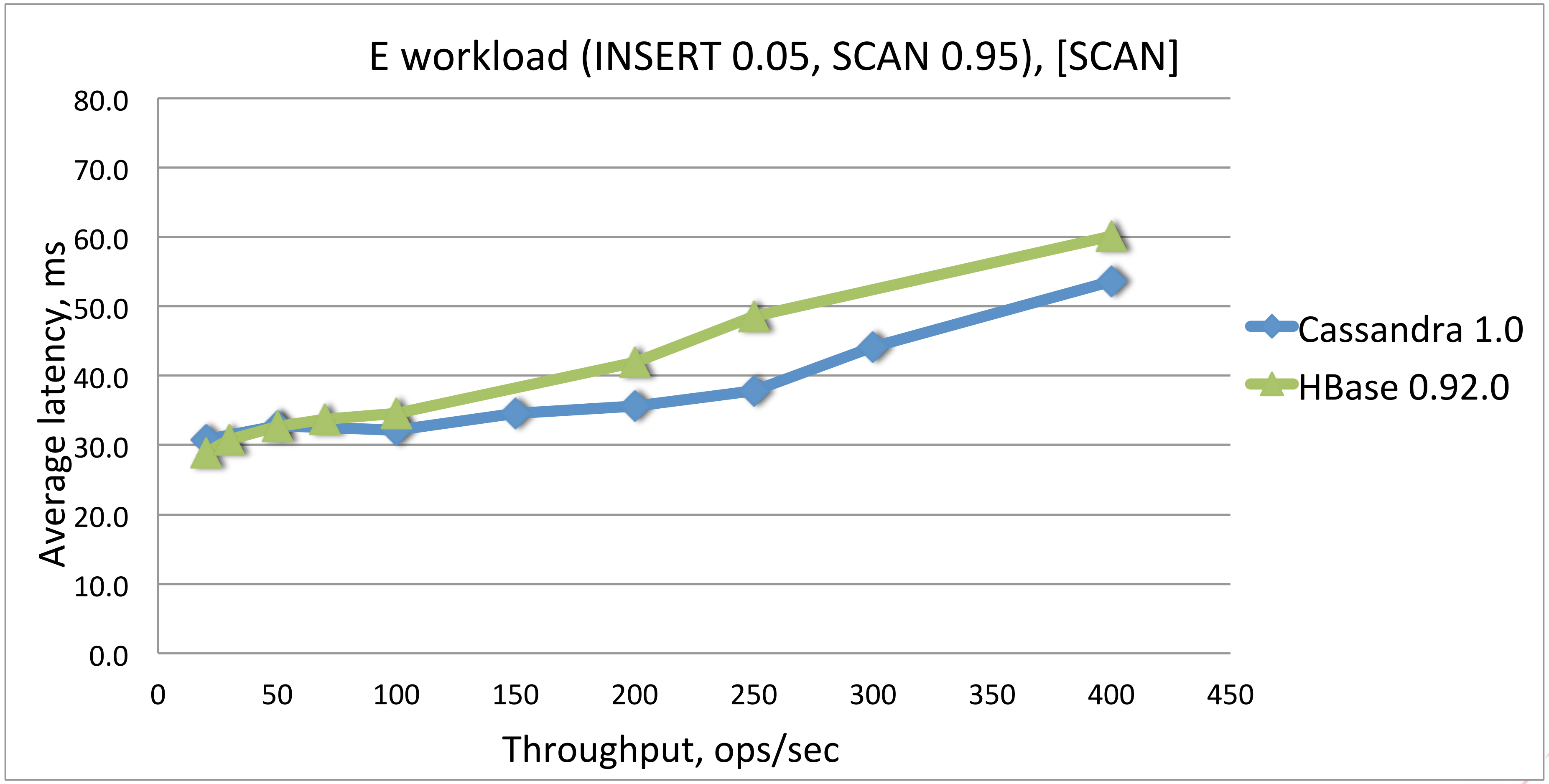
Deferred log flush does the right job for HBase during mutation ops. Edits are committed to the memstore firstly and then aggregated edits are flushed to HLog asynchronously. Cassandra has great write throughput since writes are first written to the commit log with append method which is fast operation. MongoDB's latency suffers from global write lock. Riak behaves more stably than MongoDB.

Read only workload (C)



Read only workload simulates data caching system, where data itself is constructed elsewhere. Application just reads the data. B-Tree indexes make MySQL Sharded a notable winner in this competition.

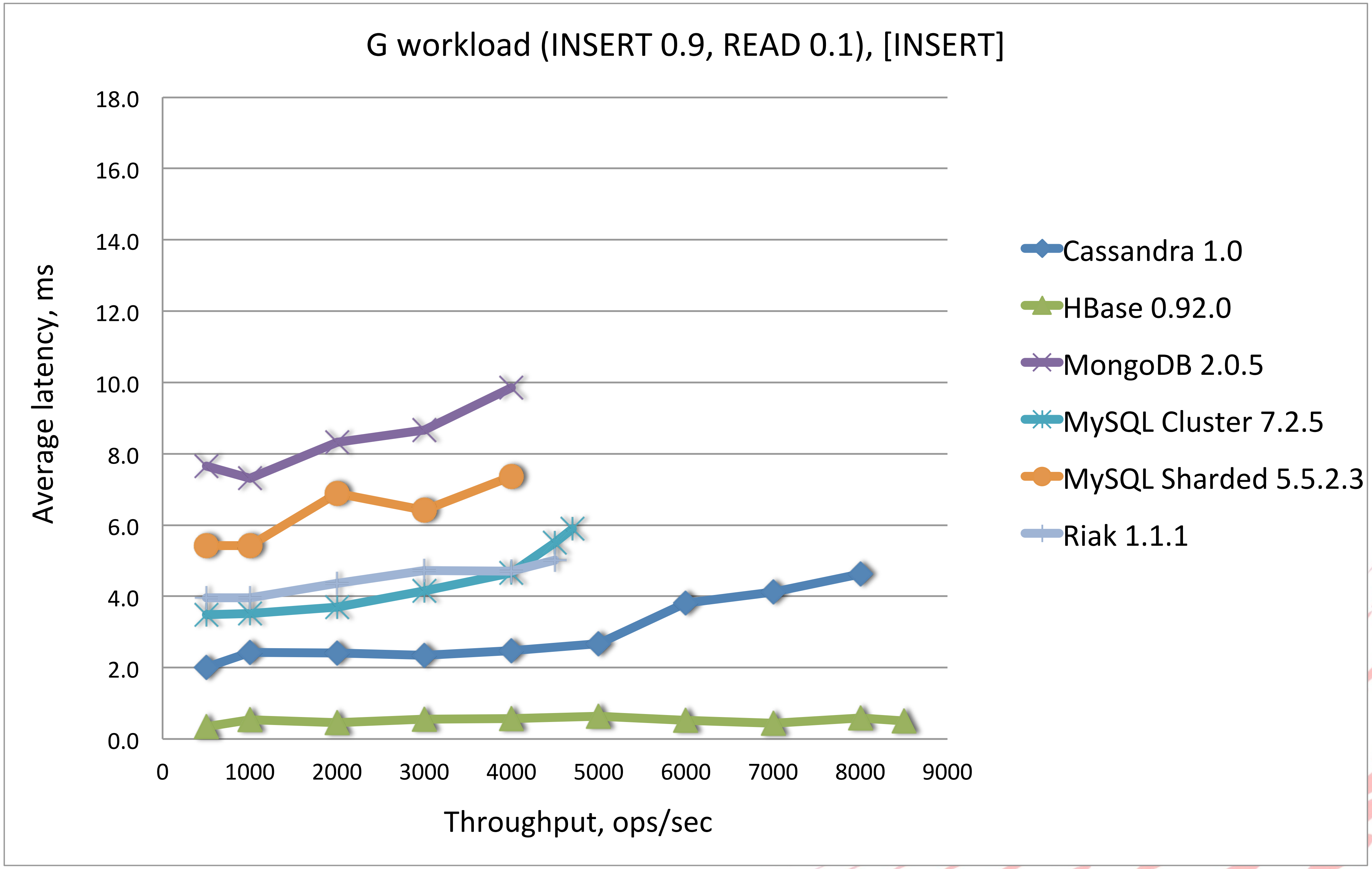
Scan ranges workload (E), [SCAN]



HBase performs a bit better than Cassandra in range scans, though Cassandra range scans improved noticeably from the 0.6 version presented in YCSB slides.

MongoDB 2.5 max throughput 20 ops/sec, latency >≈ 1 sec
MySQL Cluster 7.2.5 <10 ops/sec, latency ≈400 ms.
MySQL Sharded 5.5.2.3 <40 ops/sec, latency ≈400 ms.
Riak's 1.1.1 bitcask storage engine doesn't support range scans (eleveldb was slow during load)

Insert mostly workload (G), [INSERT]



Workload with high volume inserts proves that HBase is a winner here, closely followed by Cassandra. MySQL Cluster’s NDB engine also manages perfectly with intensive writes.

Before conclusions

- Is there a single winner?
- Who is worth the prize?

Summary

Answers

- You decide who is a winner
- NoSQL is a “different horses for different courses”
- Evaluate before choosing the “horse”
- Construct your own or use existing workloads
  - Benchmark it
  - Tune database!
  - Benchmark it again

Amazon EC2 observations

- Scales perfectly for NoSQL
- EBS slowes down database on reads
- RAID0 it! Use 4 disk in array (good choice), some reported performance degraded with higher number (6 and >)
- Don’t be sparing of RAM!

Thank you for attention

mailto:
sergey.bushik@altoros.com

skype:
siarhei_bushyk

linkedin:
http://www.linkedin.com/pub/sergey-bushik/25/685/199