Improving NoSQL Database Benchmarking

Lessons Learned

Steffen Friedrich

University of Hamburg
Department of Informatics
Databases and Information Systems

friedrich@informatik.uni-hamburg.de

March 23, 2017 @ UIOP 2017, DKRZ Hamburg
Part 1

RDBMS => TPC
VS
NoSQL => YCSB
Relational DBMS Benchmarking

Relational DBMS

~50 years database theory

=> high degree of standardization greatly simplified the development of

TPC * 1988
Transaction Processing Performance Council
Standard Benchmarks
T P C B e n c h m a r k s:  
M u l t i p l e d o m a i n s p e c i f i c b e n c h m a r k s:

- The TPC-C O L T P benchmark
  - workload consists of five transaction types simulating activities of a wholesale supplier
  - requires ACID transactions.
  - only requirements specification
    => vendors may implement and run TPC-C
    => TPC consortium approve result reports
  - Metrics: transactions per minute (tpmC), price / tpmC

- TPC-DI, TPC-DS, TPC-E, TPC-H, ...

- Obsolete: TPC-A, TPC-B, TPC-W, ...
NoSQL Databases

> "NoSQL" term coined in 2009

> Interpretation: "Not Only SQL"

> Development driven by large web companies

> Main motivation: Scalability

Large user-generated data / Request load
RDBMS VS NoSQL DB

Scaling

Scale-Up (vertical scaling)
- More RAM
- More CPU
- More HDD

Scale-Out (horizontal scaling)
- Commodity hardware
- Connected by network

Specialized DB hardware
RDBMS | VS | NoSQL DB

Relational data model

Different data models:
- Key-Value,
- Document,
- Wide-Column,
- Graph

SQL query language

Many query languages / APIs

Explicit schema

Schema free => implicit schema

Normalization

denormalization

ACID-Transactions

No transactions & eventual consistency

...
RDBMS VS NoSQL DB

One Size Fits All VS Polyglot Persistence

Specialized Databases for special requirements
More About NoSQL Databases?


NoSQL Toolbox

NoSQL Decision tree:

- **Access**
  - Fast Lookups
  - Volume
  - Unbounded
  - HDD: Size

- **Complex Queries**
  - Volume
  - Unbounded

- **Capacity**
  - CAP
  - CP
  - ACID

- **Consistency**
  - Availability
  - Ac-Fluc

- **Query Pattern**
  - Analytics

Example Applications:

- Cache
- Shopping-basket
- Order History
- OLTP
- Website
- Social Network
- Big Data

Steffen Friedrich | Improving NoSQL Database Benchmarking: Lessons Learned 9
NoSQL Performance Evaluation?

Heterogeneous NoSQL landscape

De facto standard benchmarking framework

YCSB!

Yahoo Cloud Serving Benchmark!

=> User perspective on web app. performance
=> Not only throughput => response times / latencies

Cooper et al.:
Benchmarking Cloud Serving Systems with YCSB, SoCC‘10, ACM, 2010
https://github.com/brianfrankcooper/YCSB/wiki
Limited to the functionality all NoSQL systems have in common:

- **Key-Value interface of CRUD-operations**
- **No domain driven workloads**

=> **configurable mix of operations**

- **No Transactions**
- **No Joins**
- **No Complex Queries**

Databases only have to implement the simple **Database Interface Layer**

- **Command line properties**
  - Database to use
  - Workload Class to use
  - Workload File to use
  - Target throughput
  - Number of client threads
  - ...
Part 1

The Coordinated Omission Problem

“a conspiracy we’re all a part of”

Quote

Gil Tene, CTO @ Azul Systems:

How NOT to Measure Latency, QCon, 2013 - 2016

infoq.com/presentations/latency-response-time

Steffen Friedrich, Wolfram Wingerath & Norbert Ritter:

"Coordinated Omission in NoSQL Database Benchmarking",
YCSBs load generation

```java
_targetOpsTickNanos = (long) (1 000 000 000 / target)
long overallStartTime = System.nanoTime();

while (_opsdone < _opcount) {
    long startTime = System.nanoTime();
    Status status = _db.read( table, key, fields, result );
    long endTime = System.nanoTime();

    _measurements.measure("READ", (int)( (endTime - startTime) / 1000));

    _opsdone++;

    long deadline = overallStartTime + _opsdone * _targetOpsTickNanos;
    long now = System.nanoTime();
    while((now = System.nanoTime()) < deadline) {
        LockSupport.parkNanos( deadline - now );
    }
}
```
while (_opsdone < _opcount) {
    long startTime = System.nanoTime();
    Status status = _db.read( table, key, fields, result );
    long endTime = System.nanoTime();

    _measurements.measure("READ", (int)((endTime - startTime) / 1000));
}
YCSBs load generation

```java
_targetOpsTickNanos = (long) (1000000000 / targetThroughput);
long overallStartTime = System.nanoTime();

while (_opsdone < _opcount) {

    _opsdone++;

    long deadline = overallStartTime + _opsdone * _targetOpsTickNanos;
    long now = System.nanoTime();
    while ((now = System.nanoTime()) < deadline) {
        LockSupport.parkNanos( deadline - now );
    }
}
```
YCSB’s load generation

```java
_targetOpsTickNanos = (long) (1_000_000_000 / targetThroughput);
long overallStartTime = System.nanoTime();

while (_opsdone < _opcount) {
    long startTime = System.nanoTime();
    Status status = _db.read(table, key, fields, result);
    long endTime = System.nanoTime();
    _measurements.measure("READ", (int)( (endTime - startTime) / 1000));
    _opsdone++;
    long deadline = overallStartTime + _opsdone * _targetOpsTickNanos;
    long now = System.nanoTime();
    while ((now = System.nanoTime()) < deadline) {
        LockSupport.parkNanos(deadline - now);
    }
}
```
The Coordinated Omission Problem

Example

System easily handles 10 ops/sec

latency < 1 ms
The Coordinated Omission Problem Example

- 1s database hiccup

Steffen Friedrich | Improving NoSQL Database Benchmarking: Lessons Learned
The Coordinated Omission Problem

Example

Database is able to influence the request rate!

$\Rightarrow$ coordinated omission of relevant measurement points

1s database hiccup
The Coordinated Omission Problem

Example

- proper measurement
- 1s database hiccup
## The Coordinated Omission Problem

**Example**

### The Results:

<table>
<thead>
<tr>
<th></th>
<th>AVG.</th>
<th>90%ile</th>
<th>99%ile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Hiccup</td>
<td>0.92</td>
<td>1.133</td>
<td>1.649</td>
<td>8.423</td>
</tr>
<tr>
<td>Hiccup</td>
<td>17.43</td>
<td>7.539</td>
<td>603.647</td>
<td>903.679</td>
</tr>
<tr>
<td>Hiccup YCSB</td>
<td>4.39</td>
<td>4.711</td>
<td>6.599</td>
<td>902.143</td>
</tr>
</tbody>
</table>

---

**Steffen Friedrich | Improving NoSQL Database Benchmarking: Lessons Learned**
The Coordinated Omission Problem

Example

The Results:

<table>
<thead>
<tr>
<th></th>
<th>AVG.</th>
<th>90%ile</th>
<th>99%ile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Hiccup</td>
<td>0.92</td>
<td>1.133</td>
<td>1.649</td>
<td>8.423</td>
</tr>
<tr>
<td>Hiccup</td>
<td>17.43</td>
<td>7.539</td>
<td>603.647</td>
<td>903.679</td>
</tr>
<tr>
<td>Hiccup YCSB</td>
<td>4.39</td>
<td>4.711</td>
<td>6.599</td>
<td>902.143</td>
</tr>
</tbody>
</table>

Do not just look at average latencies (+ StdDeviation), because latencies are not normally distributed!
Coordinated Omission Correction
since YCSB Version 0.2.0 RC 1 June 2015

=> intended measurement interval

while (_opsdone < _opcount) {

startTime = _deadline
(computed after previous request)

_measurement.measure("INTENDED_READ", (int)((endTime - _deadline) / 1000));

_opsdone++;

_deadline = overallStartTime + _opsdone * _targetOpsTickNanos;

...
Coordinated Omission Correction
since YCSB Version 0.2.0 RC 1 June 2015

=> intended measurement interval

while (_opsdone < _opcount) {
  startTime = _deadline
  (computed after previous request)

  _measurements.measure("INTENDED_READ", (int)((endTime - _deadline) / 1000));
  _opsdone++;
  _deadline = overallStartTime + _opsdone * _targetOpsTickNanos;
}

=> but still influenceable request rate!
Closed VS. Open System Model
for load generation

Closed System Model

Think → Send → Receive

Open System Model

Leave

New Arrivals

Scalable NoSQL-Benchmarking

nosqlmark.informatik.uni-hamburg.de
Scalable NoSQL-Benchmarking

nosqlmark.informatik.uni-hamburg.de

> built to implement our consistency measurement approach

> Scales YCSB compatible workloads to multiple benchmarking nodes => Automatically aggregates results

> Compatible to the YCSB database interface layer

> Closed and Open System Model
Coordinated Omission Avoidance in NoSQLMark

```scala
implicit val ec = context.system.dispatchers.lookup("blocking-io-dispatcher")

case DoOperation => {
  val operation = workload.nextOperation
  val startTime = System.nanoTime
  val future = Future {
    sendRequest(operation)
  }
  future.onComplete {
    case Success(status) => {
      val endTime = System.nanoTime
      measurementActor ! Measure(operation.name, (endTime - startTime) / 1000)
    }
    case Failure(ex) => {
      log.error(ex, "Error occurred during operation ", operation.name)
    }
  }
}
...
```
Coordinated Omission Validation with **SickStore**

**Single-node inconsistent key-value Store**

Originally developed to validate consistency measurement approaches

Lesson we have learned: **Validate your tools!**


github.com/steffenfriedrich/SickStore
New Feature: Simulation of maximum throughput and database hiccups

1. Compute theoretical waiting time $T_i$ of request $i$ in the database system

2. Calling client thread has to sleep for $T_i$
Experimental Validation: SickStore

Benchmark: 90,000 ops, target = 1,000 ops/sec,
SickStore: 1 second hiccup, max throughput = 1250 ops/sec,

80% of max throughput

YCSB:
- Intended: 80 ms
- AVG.: 2 ms

NoSQLMark:
- AVG.: 29 ms

Graph showing latency over time and percentiles for different benchmarks.
Experimental Validation: SickStore

Benchmark: 90,000 ops, target = 1,000 ops/sec,
SickStore: 1 second hiccup, max throughput = 1,250 ops/sec,

80% of max throughput

AVG.: YCSB 6 ms, NoSQLMark 29 ms, YCSB Intended 49 ms
**Experimental Validation: SickStore**

Benchmark: 90,000 ops, target = 1,000 ops/sec,

*SickStore*: 1 second hiccup, max throughput = 1250 ops/sec,

<table>
<thead>
<tr>
<th></th>
<th>YCSB</th>
<th>NoSQLMark</th>
<th>YCSB Intended</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG.</td>
<td>9 ms</td>
<td>29 ms</td>
<td>44 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>YCSB</th>
<th>NoSQLMark</th>
<th>YCSB Intended</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG.</td>
<td>49 ms</td>
<td>29 ms</td>
<td>54 ms</td>
</tr>
</tbody>
</table>
Experimental Validation: SickStore

Different max throughputs

~10% load

~70% load

~90% load

~95% load
Elasticy Benchmark with Cassandra

- One Cassandra node loaded with 10 million records
- After 5 min add a second node
  => it starts serving after ~5 min
  => roughly the time it takes latency to stabilize
- Run each experiment for max 15 min on a fresh cluster

Kuhlenkamp et al.: Benchmarking Scalability and Elasticity of Distributed Database Systems, VLDB, 2014

YCSB without intended measurement interval
Elasticity Benchmark with Cassandra

Target throughput = 10,000 ops/sec

[Graph showing latency vs. YCSB threads with markers for 99.9%, 99%, 90%, and mean values.]
Elasticity Benchmark with Cassandra

YCSB: 48 threads

[Graph showing latency (ms) vs. throughput (ops/sec) for different benchmarks and percentiles.]
Benchmarking is hard and your latency values are probably lying to you!

> The coordinated omission problem can't be corrected!

=> Wisely implement / choose your load generator's system model!

> Do not just look at average latencies (+StdDeviation), because latencies are not normally distributed!

> Validate your tools!
Further Improvement in NoSQL Mark

More realistic distributions for request rate

- User requests => Poisson process
  => exponential inter-request/arrival times

- Many authors consider Pareto or hyper-exponential distributed inter-arrival times


Neil J. Gunther: Load Testing Think Time Distributions, blogpost, 2010

perfdynamics.blogspot.de/2010/05/load-testing-think-time-distributions.html