Coordinated Omission
in NoSQL Database Benchmarking

Steffen Friedrich, Wolfram Wingerath, Norbert Ritter

University of Hamburg
Department of Informatics
Databases and Information Systems

March 6, 2017
NoSQL Performance Evaluation?

YCSB!

Cooper et al.:
Benchmarking Cloud Serving Systems with YCSB, SoCC'10, ACM, 2010

Friedrich, Wingerath, Ritter | Coordinated Omission in NoSQL Database Benchmarking
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

_targetOpsTickNanos = (long) (1 000 000 000 / 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 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 );
    }
}
```

What if latency > _targetOpsTickNanos ? now > deadline ?
The Coordinated Omission Problem

“a conspiracy we’re all a part of”

Gil Tene, Azul Systems:
How NOT to Measure Latency, QCon, 2016

infoq.com/presentations/latency-response-time
The Coordinated Omission Problem

Example 1

System easily handles 10 ops/sec

latency < 1 ms

proper measurement
The Coordinated Omission Problem

Example II

![Graph showing a scatter plot with 'proper measurement' and '1s database hiccup' annotations.](image-url)
The Coordinated Omission Problem

Example III

![Graph showing latency over time with a database hiccup at 11 seconds]

- YCSB

Friedrich, Wingerath, Ritter | Coordinated Omission in NoSQL Database Benchmarking
Database is able to influence the request rate!

=> coordinated omission of relevant measurements
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>
The Coordinated Omission Problem

Real World GC Event Chart

80 - 20 seconds GC pauses @ Netflix Cassandra Cluster 2012

http://techblog.netflix.com/2013/05/garbage-collection-visualization.html

Friedrich, Wingerath, Ritter | Coordinated Omission in NoSQL Database Benchmarking
The Coordinated Omission Problem

Real World GC Event Chart

After fixing the problem: max 5 seconds GC pauses

http://techblog.netflix.com/2013/05/garbage-collection-visualization.html
Closed VS. Open System Model

Closed System Model
- Think
- Receive

Open System Model
- Send
- Leave

New Arrivals

Closed VS. Open System Model

Schröder et al. surveyed system models in various web related workload generators:

• "Most of these generators/benchmarks assume a closed system model”

• "For many of these workload generators, it was quite difficult to figure out which system model was being assumed”

• "Principle (i): For a given load, mean response times are significantly lower in closed systems than in open systems.”
while (_opsdone < _opcount) {

=> intended measurement interval

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

_opsdone++;

_deadline = overallStartTime + _opsdone * _targetOpsTickNanos;

...
while (_opsdone < _opcount) {

=> intended measurement interval

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

    _opsdone++;

    _deadline = overallStartTime + _opsdone * _targetOpsTickNanos;

=> but still influences the request rate!

}
Scalable NoSQL-Benchmarking

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

nosqlmark.informatik.uni-hamburg.de

• Scaling YCSB compatible workloads to multiple benchmarking nodes
  => Automatically aggregated results
• Compatible to YCSB database interface layer
• Closed and Open System Model

YCSB! Scala
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

Originally developed to validate consistency measurement approaches

- consistent single-node backend
- multi-node behaviour
- tunable staleness
- globally consistent logfile

[github.com/steffenfriedrich/SickStore](https://github.com/steffenfriedrich/SickStore)

Coordinated Omission Validation with 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 = 1000 ops/sec,
SickStore: 1 second hiccup, max throughput = 1250 ops/sec,

80% of max throughput
Experimental Validation: SickStore

Benchmark: 90,000 ops, target = 1000 ops/sec,
SickStore: 1second hiccup, max throughput = 1250 ops/sec,

AVG.: 6 ms 29 ms 49ms
Experimental Validation: SickStore

Benchmark: 90 000 ops, target = 1000 ops/sec,
SickStore: 1 second hiccup, max throughput = 1250 ops/sec,

YCSB NoSQLMark YCSB Intended
AVG.: 49 ms 29 ms 54 ms

YCSB NoSQLMark YCSB Intended
AVG.: 3 ms 29 ms 44 ms
Experimental Validation: SickStore

Different max throughputs

~10% load

~70% load

~90% load

~95% load
Elasticity Benchmark with Cassandra

- one Cassandra node loaded with 10 million records

- after 5 min add 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

YCSB without intended measurement interval

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

target throughput = 10,000 ops/sec
Elasticity Benchmark with Cassandra
YCSB: 48 threads

![Graph showing latency vs. throughput for Cassandra and YCSB benchmarks.]

- NoSQLMark
- YCSB
- YCSB Intended
- 99.9\%ile
- 99\%ile
- 90\%ile
- Mean

Throughput (ops/sec): 10000 to 15000
Latency (ms): 2 to 20000
Summary

Benchmarks are hard and your latency values are probably lying to you!

Be aware of the system model underlying your load generators.
Further Improvement in NoSQLMark

More realistic distributions for request rate

- Exponential inter-request/arrival time $\Rightarrow$ poisson distributed request/arrival rate

  \[
  \begin{array}{cccccccc}
  | & | & | & | & | & | & | & | & | & | \\
  \hline
  t
  \end{array}
  \]

  Inter-arrival Times

  \[
  \begin{array}{ccccccc}
  3 & 2 & 3 & 2 & 2 & 4 & x \\
  \hline
  \end{array}
  \]

  Count per Interval

- Some authors consider Perato or hyper-exponential distributed inter-arrival time


Neil J. Gunther: Load Testing Think Time Distributions, blogpost, 2010
perfdynamics.blogspot.de/2010/05/load-testing-think-time-distributions.html