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

RDBMS

~ 50 years database theory

=> high degree of standardization greatly simplified the development of

TPC * 1988

Transaction Processing Performance Council

Standard Benchmarks

TPC Benchmarks:

Multiple domain specific benchmarks:

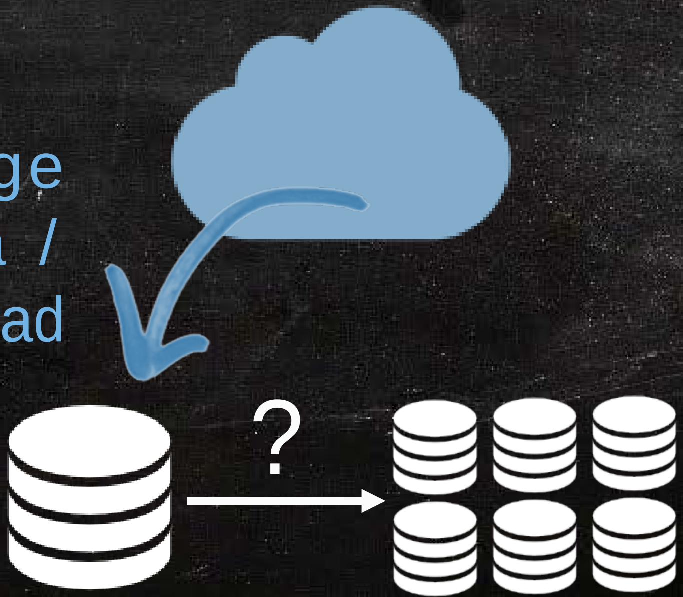
- The TPC-C OLTP benchmark
  - workload consists of five transaction types simulating activities of a wholesale supplier
  - requires ACID transactions.
  - only requirements specification
=> vendors may implement and run TPC-C
=> TPC consortium approve result reports
  - Metrics: transactions per minute (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

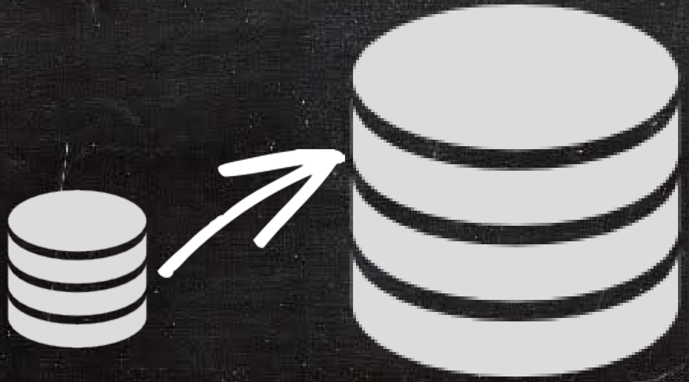
Large
user-generated data /
Request load



Scalability

Scale-Up

(*vertical* scaling)



Specialized DB hardware

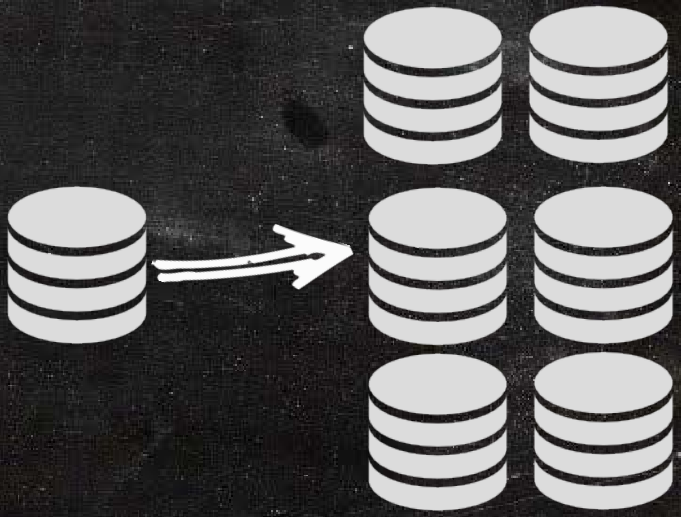
More RAM

More CPU

More HDD

Scale-Out

(*horizontal* scaling)



Commodity hardware

Connected by network

RDBMS	VS	NoSQL DB
Relational data model		Different data models: › Key-Value, › Document, › Wide-Cloumn, › 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

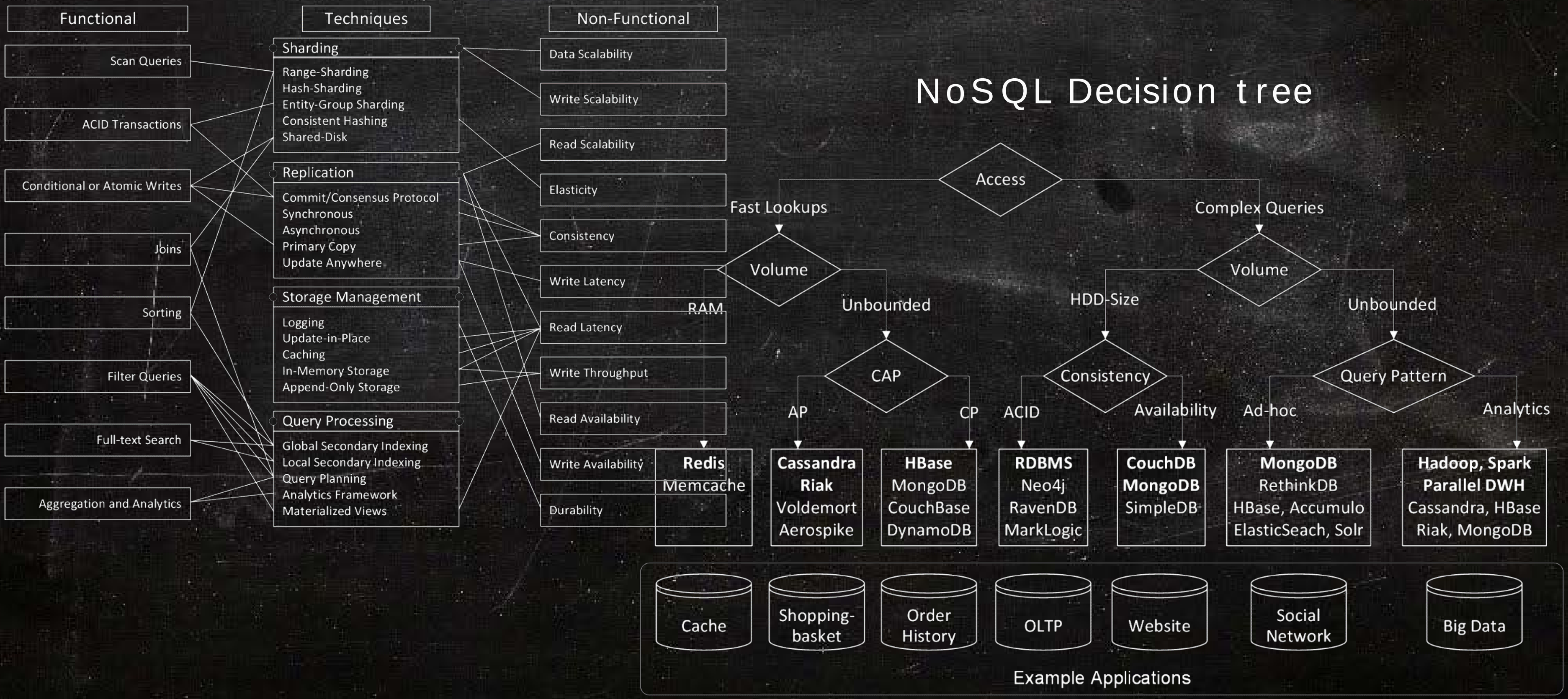
Specialized Databases for special requirements

More About NoSQL Databases?

Felix Gessert, Wolfram Wingerath, Steffen Friedrich & Norbert Ritter: "NoSQL database systems: a survey and decision guidance", *Computer Science - Research and Development* 1-13 (2016)

NoSQL Toolbox

NoSQL Decision tree



Functional

Scan Queries

ACID Transactions

Conditional or Atomic Writes

Joins

Sorting

Filter Queries

Full-text Search

Aggregation and Analytics

Techniques

Sharding
- Range-Sharding
- Hash-Sharding
- Entity-Group Sharding
- Consistent Hashing
- Shared-Disk

Replication
- Commit/Consensus Protocol
- Synchronous
- Asynchronous
- Primary Copy
- Update Anywhere

Storage Management
- Logging
- Update-in-Place
- Caching
- In-Memory Storage
- Append-Only Storage

Query Processing
- Global Secondary Indexing
- Local Secondary Indexing
- Query Planning
- Analytics Framework
- Materialized Views

Non-Functional

Data Scalability

Write Scalability

Read Scalability

Elasticity

Consistency

Write Latency

Read Latency

Write Throughput

Read Availability

Write Availability

Durability

Access

Fast Lookups

Complex Queries

Volume

RAM

Unbounded

Volume

HDD-Size

Unbounded

CAP

AP

CP

Consistency

ACID

Availability

Query Pattern

Ad-hoc

Analytics

Redis
Memcache

Cassandra
Riak
Voldemort
Aerospike

HBase
MongoDB
CouchBase
DynamoDB

RDBMS
Neo4j
RavenDB
MarkLogic

CouchDB
MongoDB
SimpleDB

MongoDB
RethinkDB
HBase, Accumulo
ElasticSeach, Solr

Hadoop, Spark
Parallel DWH
Cassandra, HBase
Riak, MongoDB

Cache

Shopping-basket

Order History

OLTP

Website

Social Network

Big Data

Example Applications

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

YCSB!

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

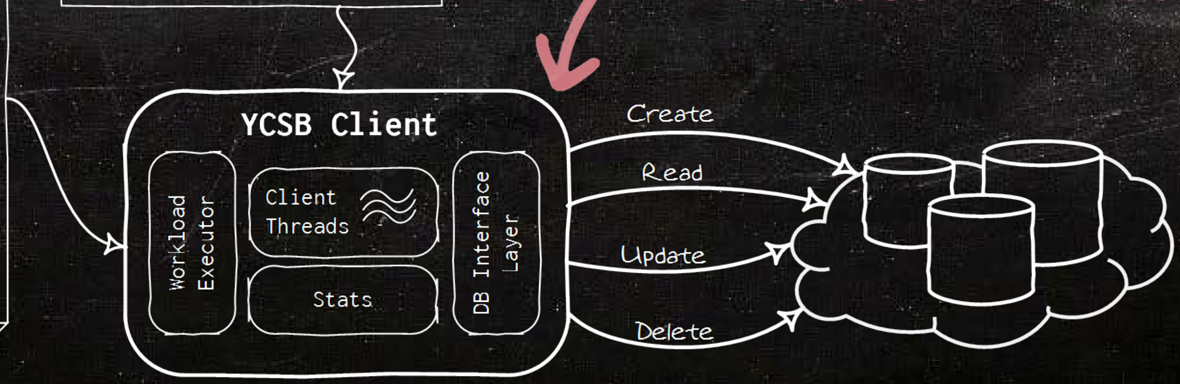
Workload File

```
recordcount=1000
operationcount=1000

readproportion=0.5
updateproportion=0.5
insertproportion=0
readmodifywriteproportion=0
scanproportion=0

requestdistribution=zipfian

fieldcount=10
fieldlength=100
readallfields=true
writeallfields=false
fieldlengthdistribution=constant

insertorder=hashed
```

Command line properties

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

Databases only have to impliment the simple Database Interface Layer



Part II

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", BTW 2017, Workshopband, p. 215-225, 2017

YCSBs load generation

```
_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 );
  }
}
```

YCSBs load generation

```
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 );
    }
}
```

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 );
  }
}
```

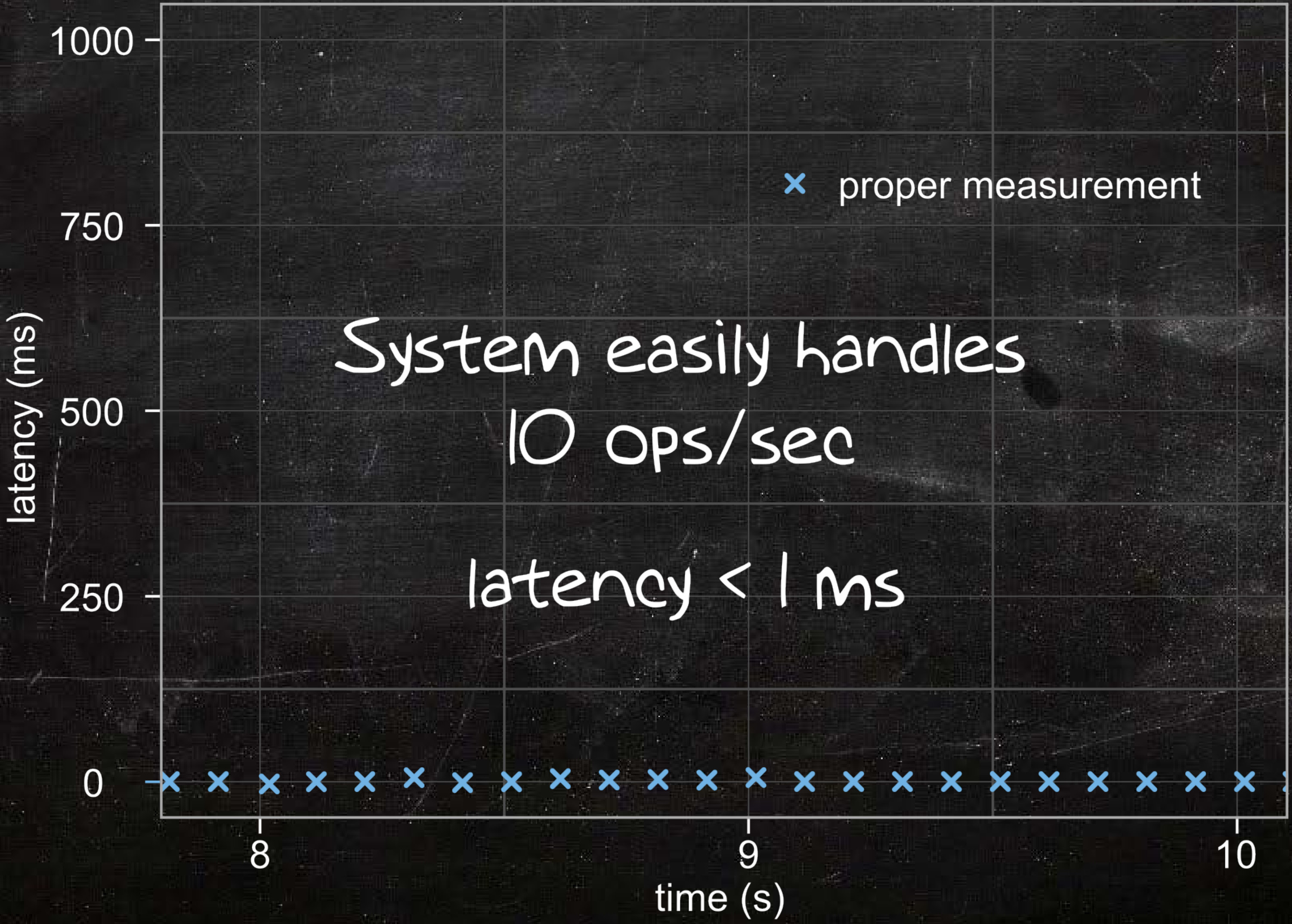
What if

latency >> _targetOpsTickNanos ?

=> now >> deadline ?

The Coordinated Omission Problem

Example

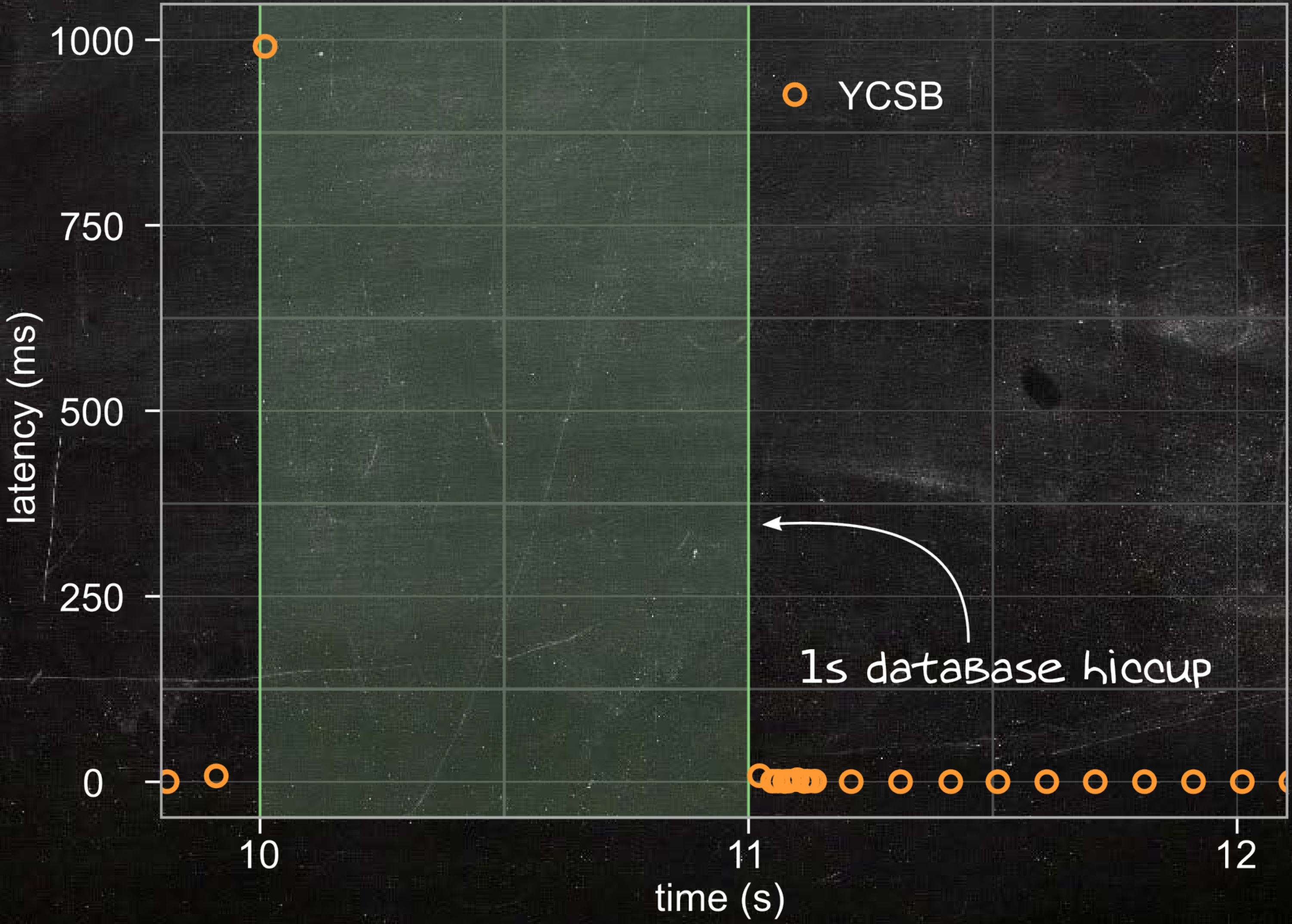


System easily handles 10 ops/sec

latency < 1 ms

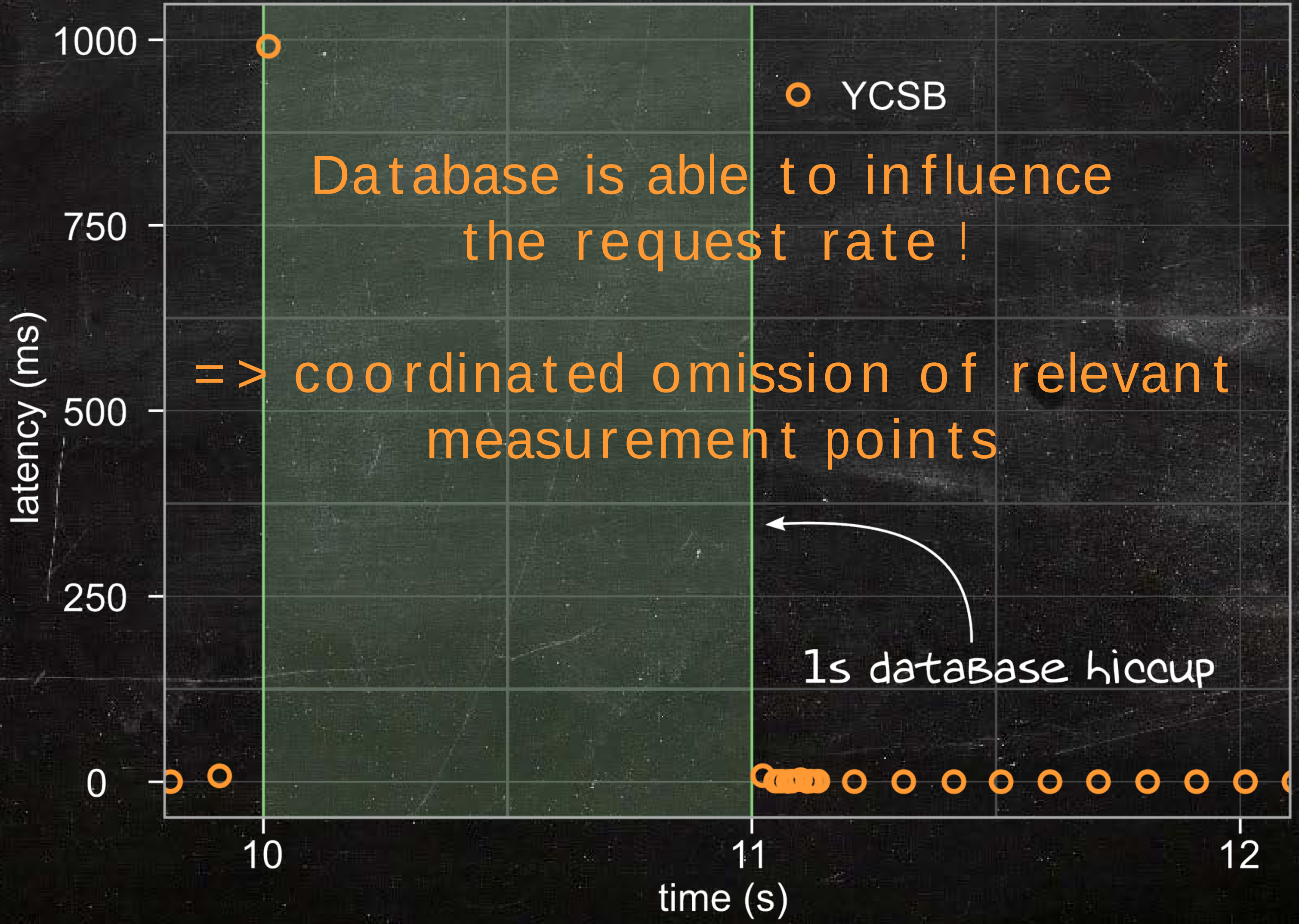
The Coordinated Omission Problem

Example



The Coordinated Omission Problem

Example

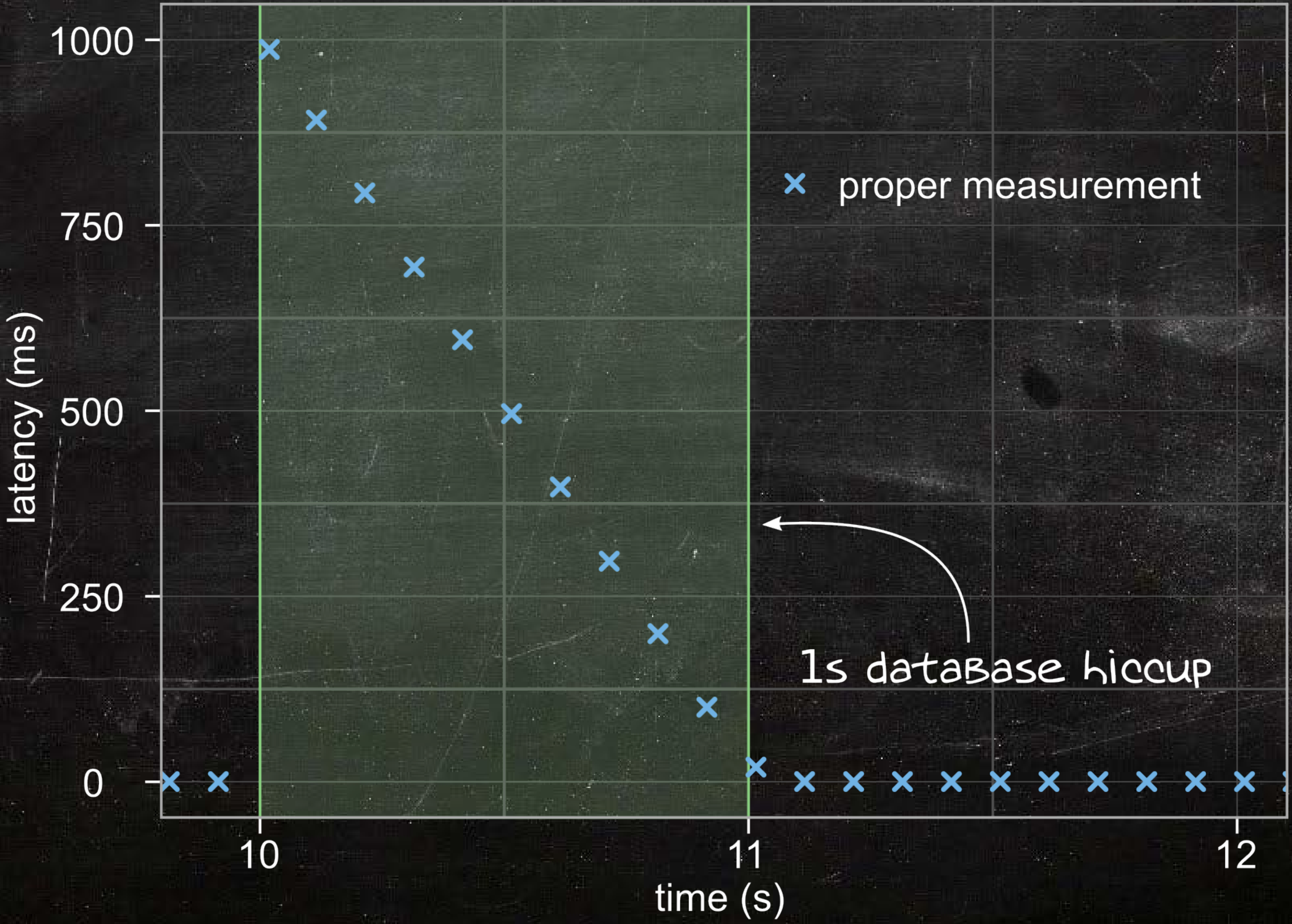


Database is able to influence the request rate !

=> coordinated omission of relevant measurement points

The Coordinated Omission Problem

Example



The Coordinated Omission Problem

Example

The Results:

	AVG.	90%ile	99%ile	Max
No Hiccup	0.92	1.133	1.649	8.423
Hiccup	17.43	7.539	603.647	903.679
Hiccup YCSB	4.39	4.711	6.599	902.143

The Coordinated Omission Problem

Example

The Results:

	AVG.	90%ile	99%ile	Max
No Hiccup	0.92	1.133	1.649	8.423
Hiccup	17.43	7.539	603.647	903.679
Hiccup YCSB	4.39	4.711	6.599	902.143

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) {



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

   _opsdone++;

   _deadline = overallStartTime + _opsdone * _targetOpsTickNanos;

                                 ...

}
```

startTime = _deadline
(computed after previous request)

Coordinated Omission Correction

since YCSB Version 0.2.0 RC 1, June 2015

=> intended measurement interval

```
while (_opsdone < _opcount) {

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

   _opsdone++;

   _deadline = overallStartTime + _opsdone * _targetOpsTickNanos;
```

startTime = _deadline

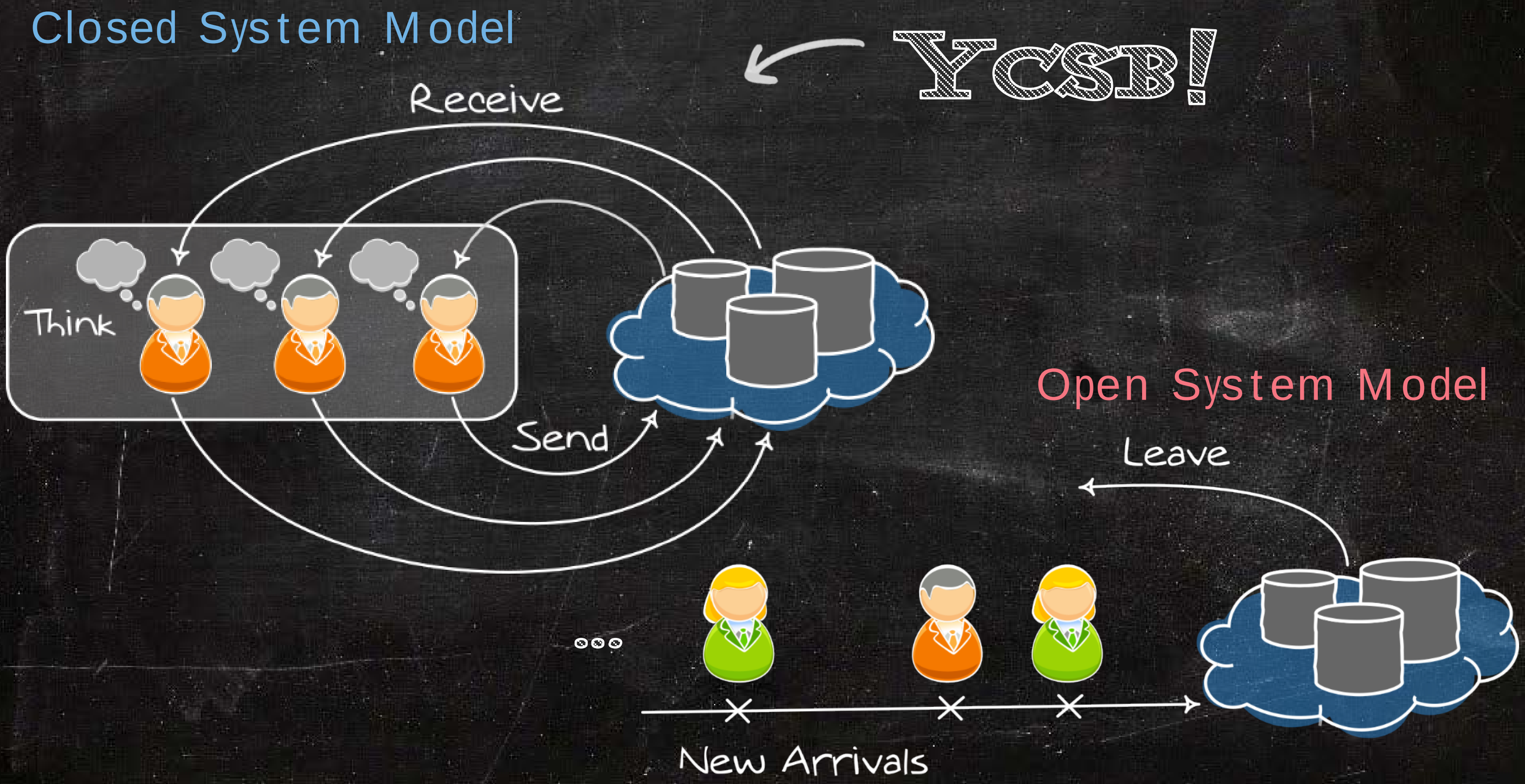
(computed after previous request)

=> but still influenceable request rate !

```
}
```

Closed VS. Open System Model

for load generation



Schröder et al. Open Versus Closed: A Cautionary Tale, 2006

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

YCSB! Scala akka

Coordinated Omission Avoidance

in NoSQLMark

```
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 occured during operation {}", operation.name)
    }
  }
...
```

Asynchronous load generation !

Coordinated Omission Validation with

SickStore

Single-node inconsistent key-value Store

Originally developed to validate consitency measurement approaches

Lesson we have learned:

Validate your tools!

Wingerath, Friedrich, Gesser, Ritter:
Who Watches the Watchmen?
On the Lack of Validation in NoSQL Benchmarking, BTW 2015

github.com/steffenfriedrich/SickStore

Coordinated Omission Validation with SickStore

Single-node inconsistent key-value Store

New Feature:
Simulation of maximum throughput and database hiccups

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

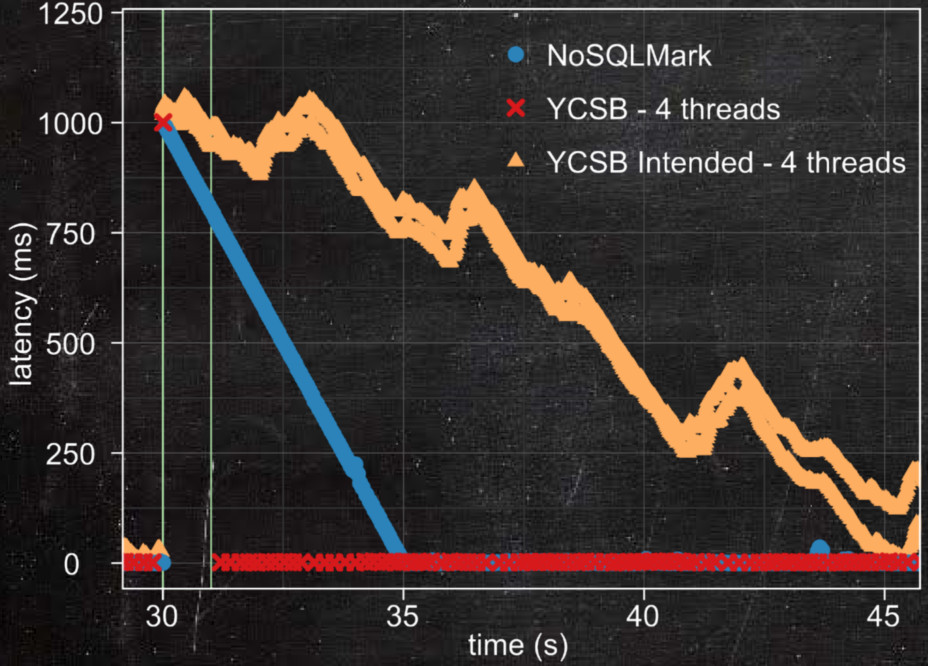
2. Calling client thread has to sleep for T_i

Experimental Validation: SickStore

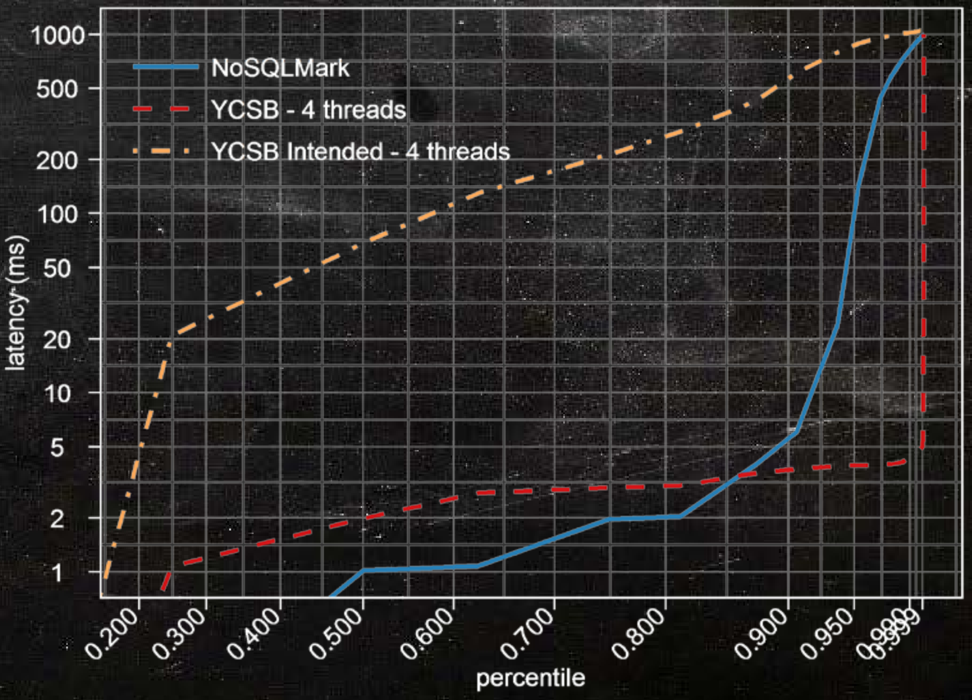
Benchmark: 90 000 ops, target = 1000 ops/sec,

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

80% of max throughput



	YCSB	NoSQLMark	YCSB Intended
AVG.:	2 ms	29 ms	180 ms

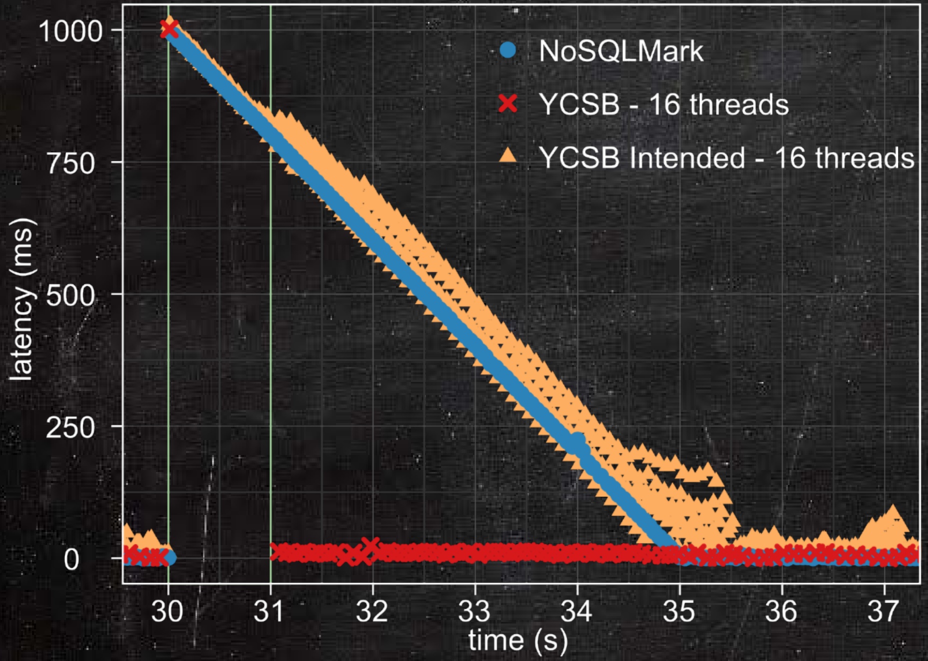


Experimental Validation: SickStore

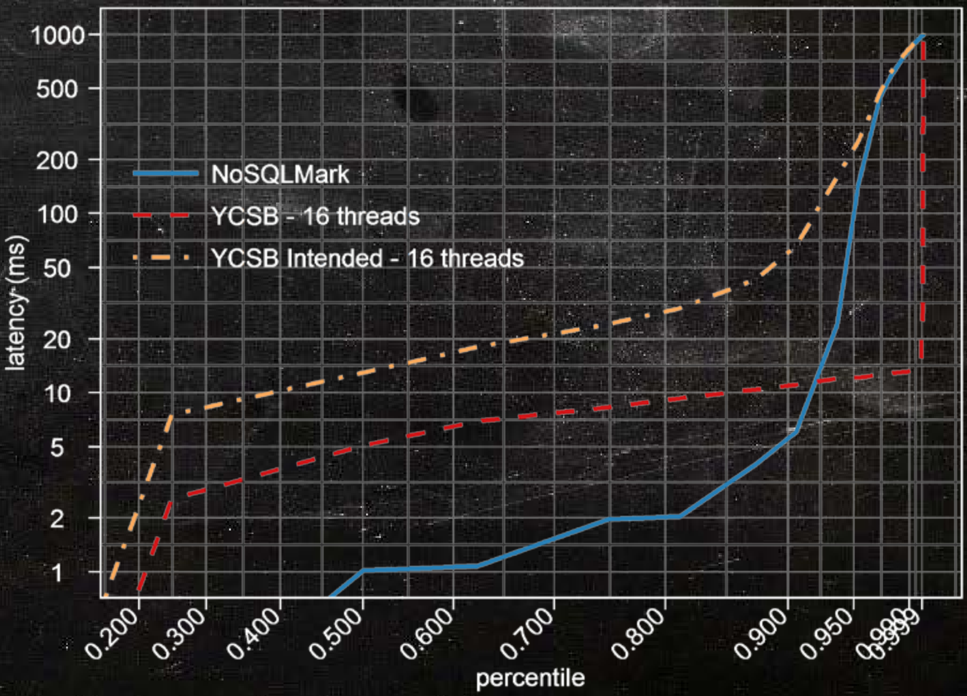
Benchmark: 90 000 ops, target = 1000 ops/sec,

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

80% of max throughput



	YCSB	NoSQLMark	YCSB Intended
AVG.:	6 ms	29 ms	49ms

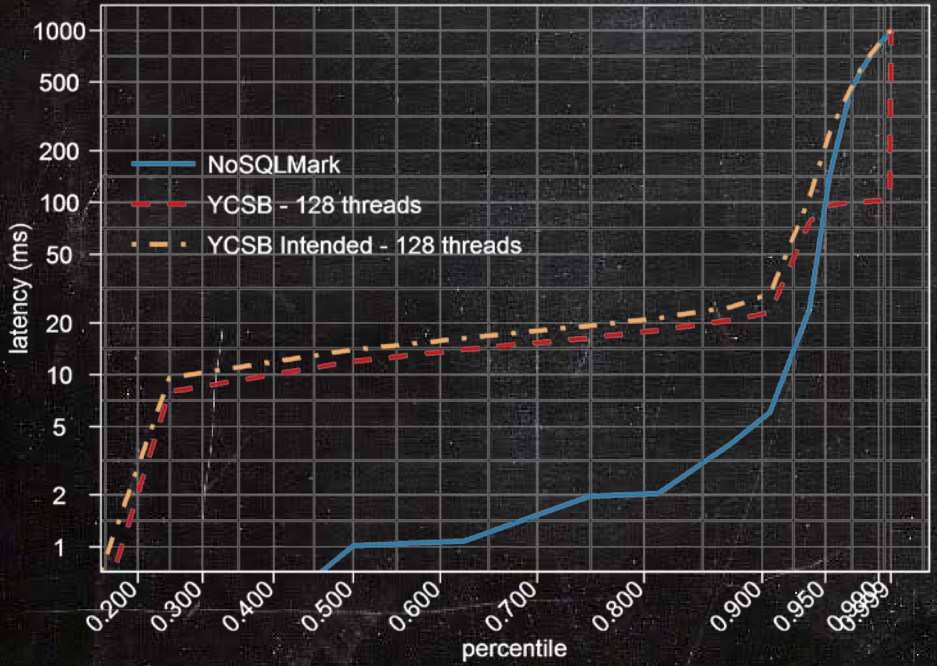


Experimental Validation: SickStore

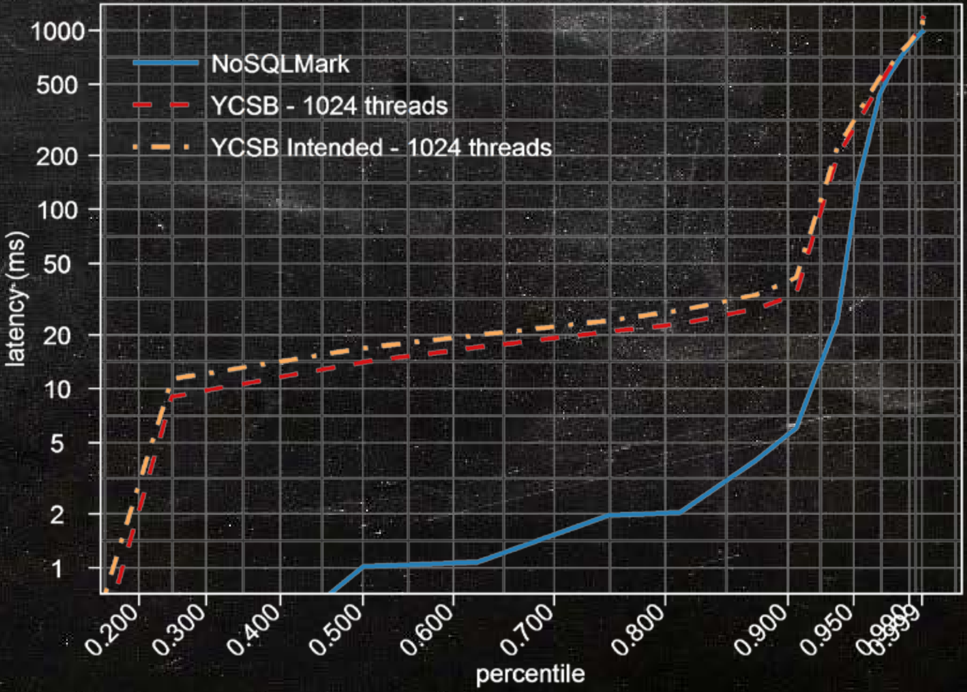
Benchmark: 90 000 ops, target = 1000 ops/sec,

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

	YCSB	NoSQLMark	YCSB Intended
AVG.:	19 ms	29 ms	44ms

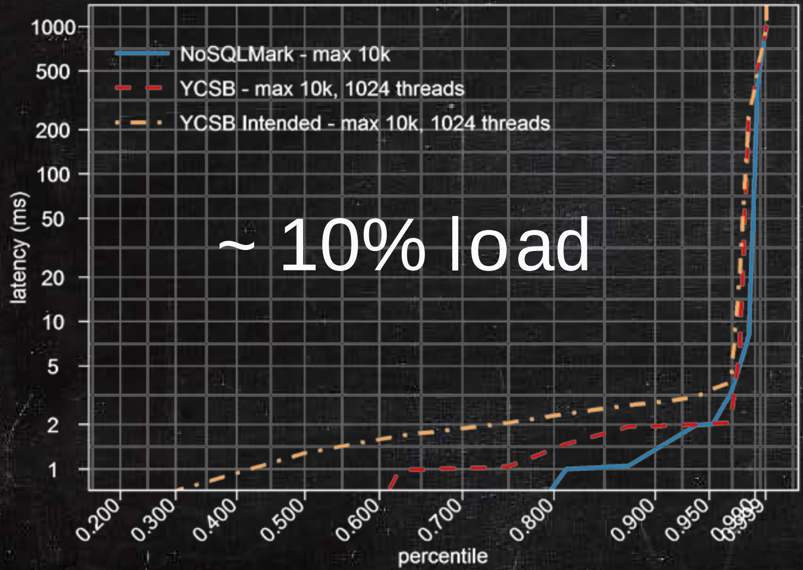


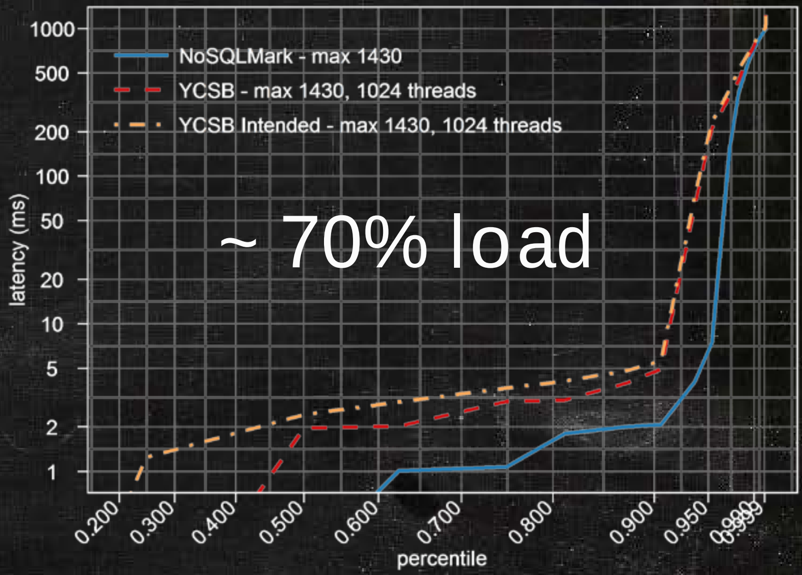
	YCSB	NoSQLMark	YCSB Intended
AVG.:	49 ms	29 ms	54ms

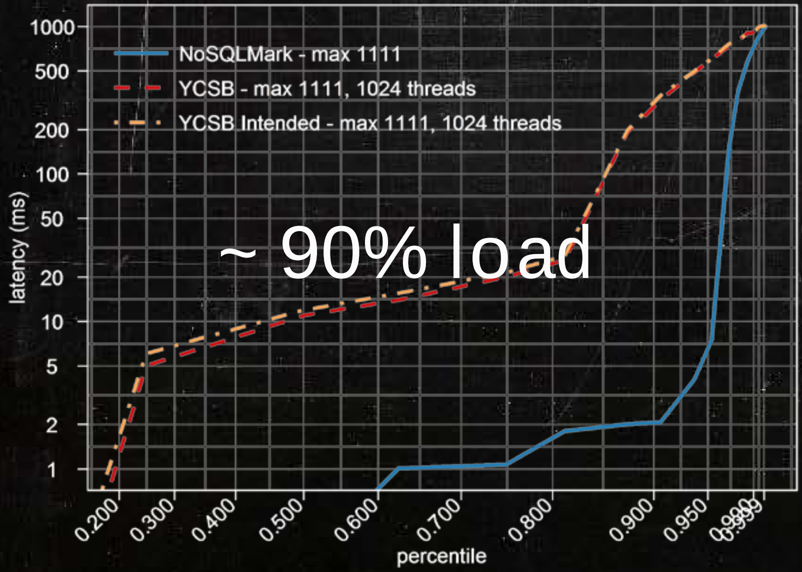


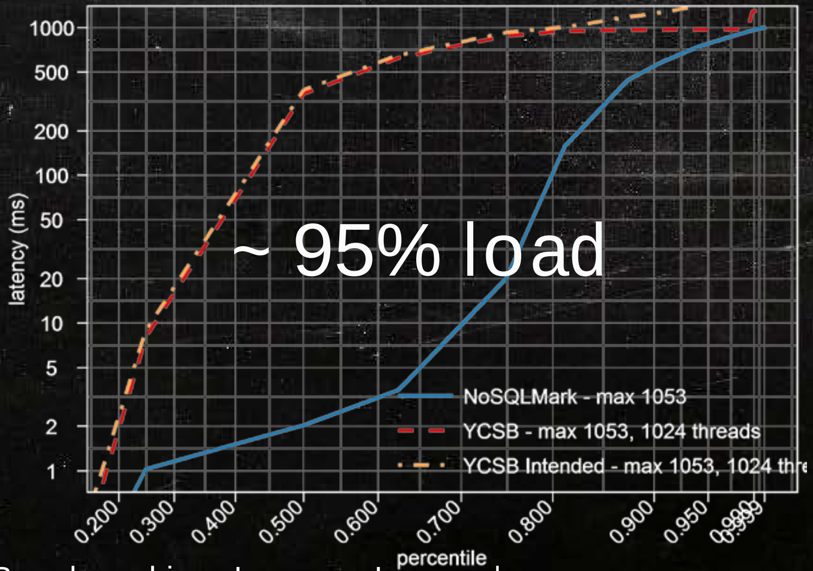
Experimental Validation: SickStore

Differrent max throughputs









Elasticity 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

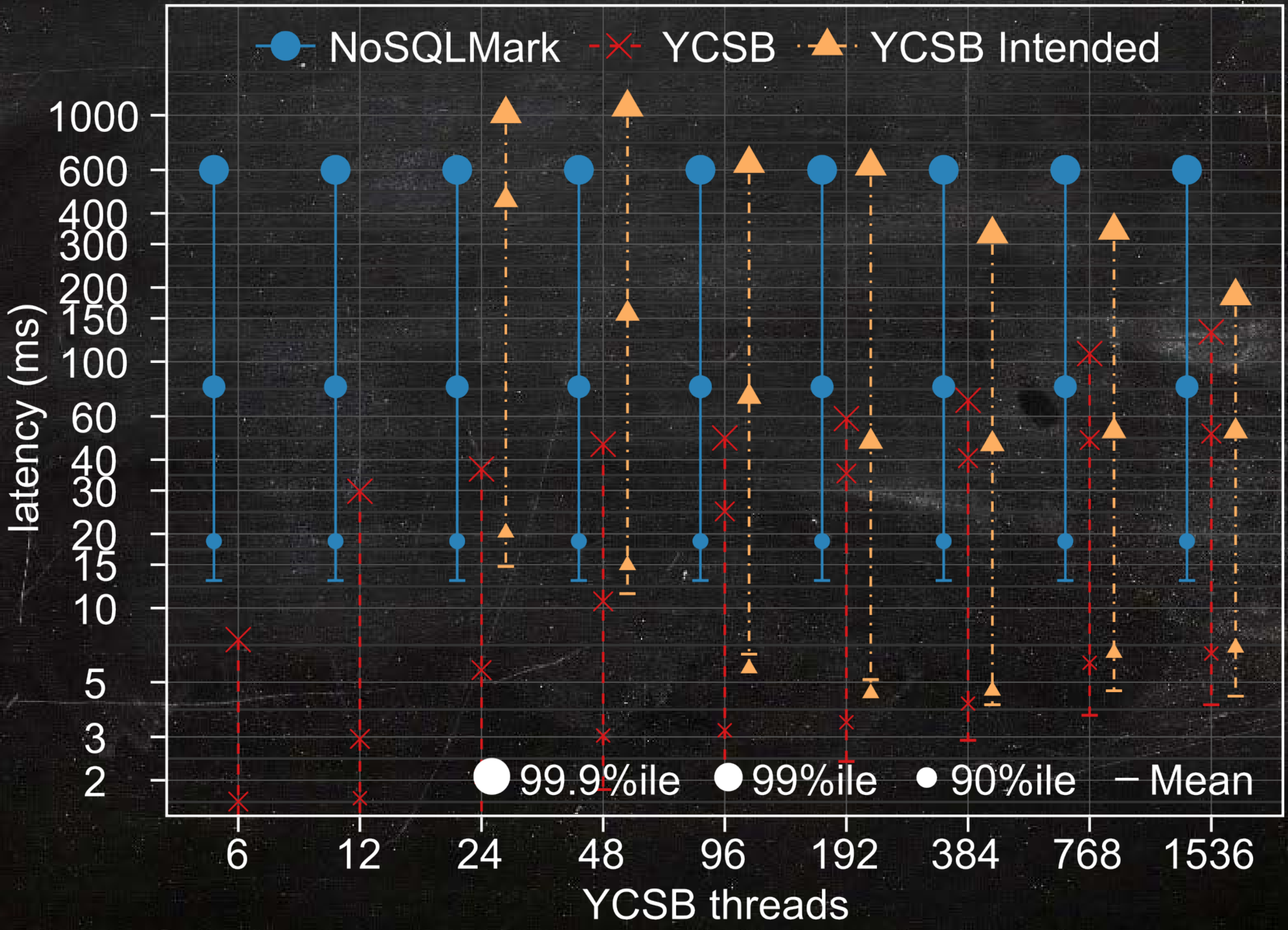
- Run each experiment for max 15 min on a fresh cluster

YCSB without intended measurement interval

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

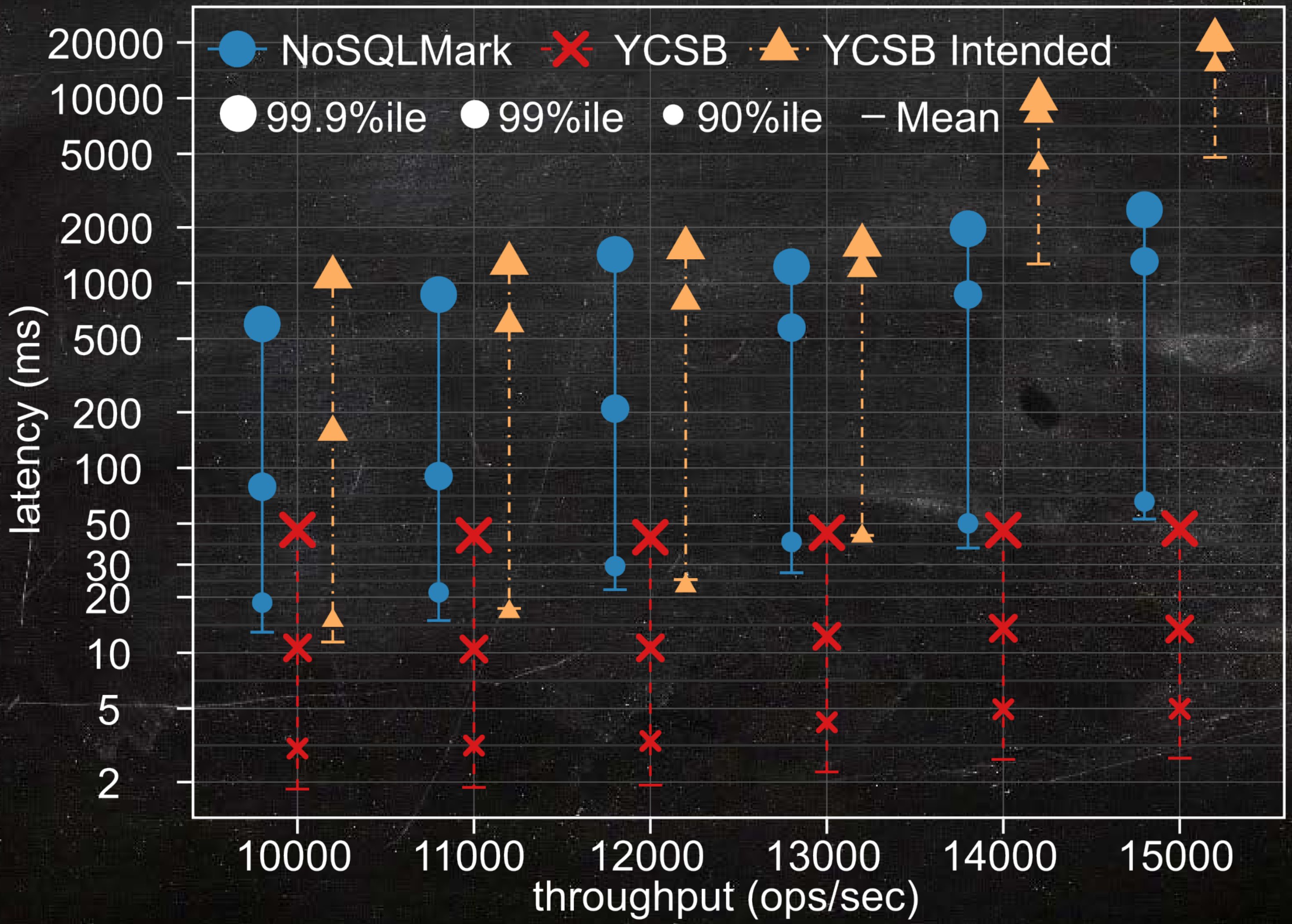
Elasticity Benchmark with Cassandra

Target throughput = 10 000 ops/sec



Elasticity Benchmark with Cassandra

YCSB: 48 threads



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 generators system model !

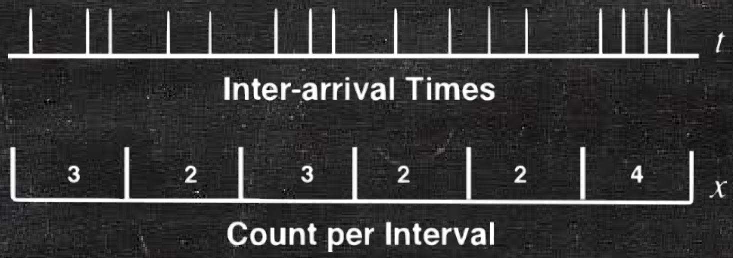
> **Do not just look at average latencies (+StdDeviation), because** latencies are not normally distributed!

> Validate your tools!

Further Improvement in NoSQLMark

More realistic distributions for request rate

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



Already Implemented In NoSQLMark

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

James F. Brady & Neil J. Gunther: How to Emulate Web Traffic Using Standard Load Testing Tools, CoRR, 2016

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