Better a Microbenchmark on a Cluster than a User at the Office: Flink Cluster Benchmarking

David Georg Reichelt [1]
Lars-Peter Meyer [1]
Stefan Kühne [1]

[1]Universität Leipzig, University Computing Centre, Research and Development

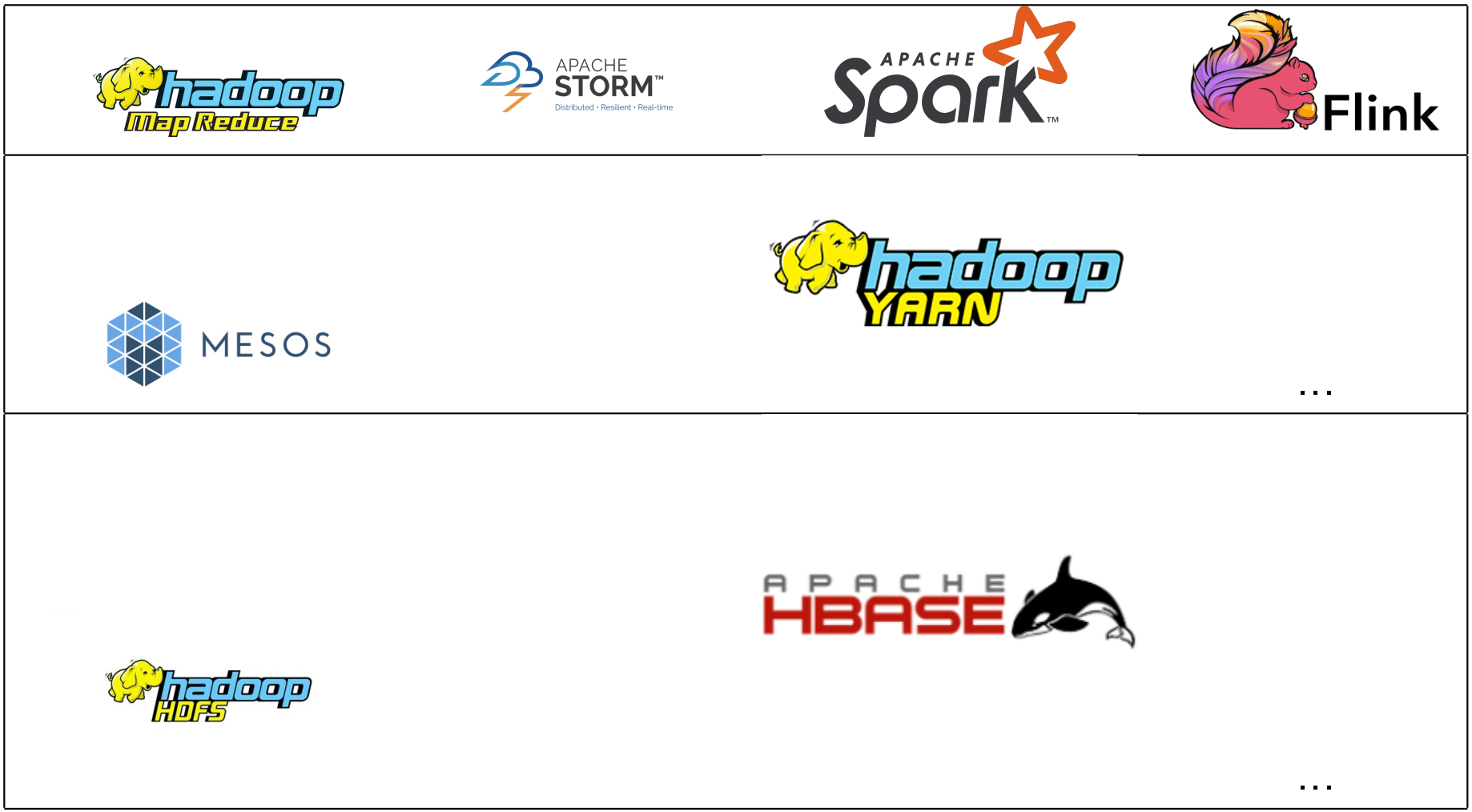
November 11, 2018

ScaDS DRESDEN LEIPZIG

ScaDS
DRESDEN LEIPZIG

Apache Flink

Why Flink?


hadoop MapReduce
APACHE STORM
Distributed • Resilient • Real-time
APACHE Spark
Flink
MESOS
hadoop YARN
...
hadoop HDFS
APACHE HBASE
...

ScaDS DRESDEN LEIPZIG

Usage of Flink

```
1  DataStream<WordWithCount> windowCounts = text
2    .flatMap(new FlatMapFunction<String,WordWithCount>()
3    {
4    @Override
5    public void flatMap(String value,
6                        Collector<WordWithCount> out) {
7       for (String word : value.split("\\s")) {
8          out.collect(new WordWithCount(word, 1L));
9       }
10   } } )
11   .keyBy("word")
12   .timeWindow(Time.seconds(5))
13   .reduce(new ReduceFunction<WordWithCount>() {
14   @Override
15   public WordWithCount reduce(WordWithCount a,
16                               WordWithCount b) {
17     return new WordWithCount(a.word,
18                              a.count + b.count);
19   } } );
```

ScaDS DRESDEN LEIPZIG

Usage of Flink

```
1  DataStream<WordWithCount> windowCounts = text
2    .flatMap(new FlatMapFunction<String,WordWithCount>()
3    {
4    @Override
5    public void flatMap(String value,
6                        Collector<WordWithCount> out) {
7       for (String word : value.split("\\s")) {
8          out.collect(new WordWithCount(word, 1L));
9       }
10   } } )
11   .keyBy("word")
12   .timeWindow(Time.seconds(5))
13   .reduce(new ReduceFunction<WordWithCount>() {
14   @Override
15   public WordWithCount reduce(WordWithCount a,
16                               WordWithCount b) {
17     return new WordWithCount(a.word,
18                              a.count + b.count);
19   } } );
```

Usage of Flink

```
1  DataStream<WordWithCount> windowCounts = text
2    .flatMap(new FlatMapFunction<String,WordWithCount>()
3    {
4    @Override
5    public void flatMap(String value,
6                        Collector<WordWithCount> out) {
7        for (String word : value.split("\\s")) {
8            out.collect(new WordWithCount(word, 1L));
9        }
10   } } )
11   .keyBy("word")
12   .timeWindow(Time.seconds(5))
13   .reduce(new ReduceFunction<WordWithCount>() {
14   @Override
15   public WordWithCount reduce(WordWithCount a,
16                               WordWithCount b) {
17     return new WordWithCount(a.word,
18                              a.count + b.count);
19   } } );
```

Usage of Flink

```
1  DataStream<WordWithCount> windowCounts = text
2    .flatMap(new FlatMapFunction<String,WordWithCount>()
3    {
4    @Override
5    public void flatMap(String value,
6                        Collector<WordWithCount> out) {
7       for (String word : value.split("\\s")) {
8          out.collect(new WordWithCount(word, 1L));
9       }
10   } } )
11   .keyBy("word")
12   .timeWindow(Time.seconds(5))
13   .reduce(new ReduceFunction<WordWithCount>() {
14   @Override
15   public WordWithCount reduce(WordWithCount a,
16                               WordWithCount b) {
17     return new WordWithCount(a.word,
18                              a.count + b.count);
19   } } );
```

ScaDS DRESDEN LEIPZIG

Usage of Flink

```
1  DataStream<WordWithCount> windowCounts = text
2    .flatMap(new FlatMapFunction<String,WordWithCount>()
3    {
4    @Override
5    public void flatMap(String value,
6                        Collector<WordWithCount> out) {
7       for (String word : value.split("\\s")) {
8          out.collect(new WordWithCount(word, 1L));
9       }
10   } } )
11   .keyBy("word")
12   .timeWindow(Time.seconds(5))
13   .reduce(new ReduceFunction<WordWithCount>() {
14   @Override
15   public WordWithCount reduce(WordWithCount a,
16                               WordWithCount b) {
17     return new WordWithCount(a.word,
18                              a.count + b.count);
19   } } );
```

Usage of Flink

```
1  DataStream<WordWithCount> windowCounts = text
2    .flatMap(new FlatMapFunction<String,WordWithCount>()
3    {
4    @Override
5    public void flatMap(String value,
6                        Collector<WordWithCount> out) {
7       for (String word : value.split("\\s")) {
8          out.collect(new WordWithCount(word, 1L));
9       }
10   } } )
11   .keyBy("word")
12   .timeWindow(Time.seconds(5))
13   .reduce(new ReduceFunction<WordWithCount>() {
14   @Override
15   public WordWithCount reduce(WordWithCount a,
16                               WordWithCount b) {
17     return new WordWithCount(a.word,
18                              a.count + b.count);
19   } } );
```

Usage of Flink

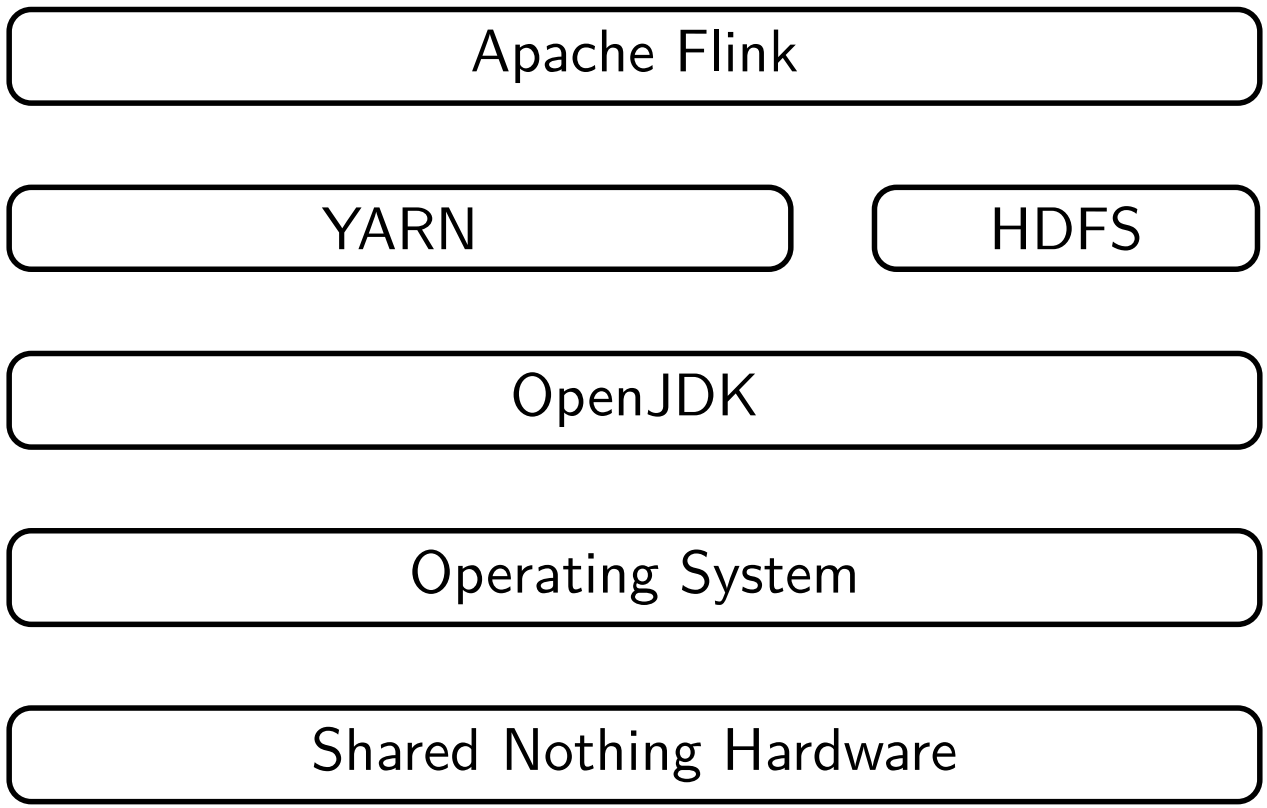


Figure: Tools and our Cluster Setup of Flink

ScaDS DRESDEN LEIPZIG

Microbenchmark Suite

ScaDS DRESDEN LEIPZIG

Microbenchmark Suite

- Goal
  - Identify performance problems in Flink clusters
  - Usage from operators perspective
  - Short-running benchmarks

Microbenchmark Suite

- Goal
  - Identify performance problems in Flink clusters
  - Usage from operators perspective
  - Short-running benchmarks

- Makrobenchmark: Measure the performance of a whole application
- Micro-Benchmark: Measure the performance of a small code unit

ScaDS DRESDEN LEIPZIG

Microbenchmarks

- Each workload: Job executed on cluster
- Parameters: *size* and *parallelism*

- Computing performance: Execute $size * factor$ additions
- RAM performance: Reserve $size * factor$ bytes
- HDFS write performance
  - Write $size * factor$ bytes
  - return throughput
- HDFS read performance
  - Read $size * factor$ bytes from previously created files
  - return throughput

ScaDS DRESDEN LEIPZIG

Assuring Statistic Rigor

- Steady state needs to be assured

- Assumption: Normal distribution
  $\Rightarrow$ Steady state reached if t-test does not reject that values are equal
- Setup: 100 Jobs with 10.000, 20.000, 40.000 size
- Measurements 15 to 25 not unequal 5 to 15
  $\Rightarrow$ 25 Measurements, use last 10

Assuring Statistic Rigor

- Steady state needs to be assured

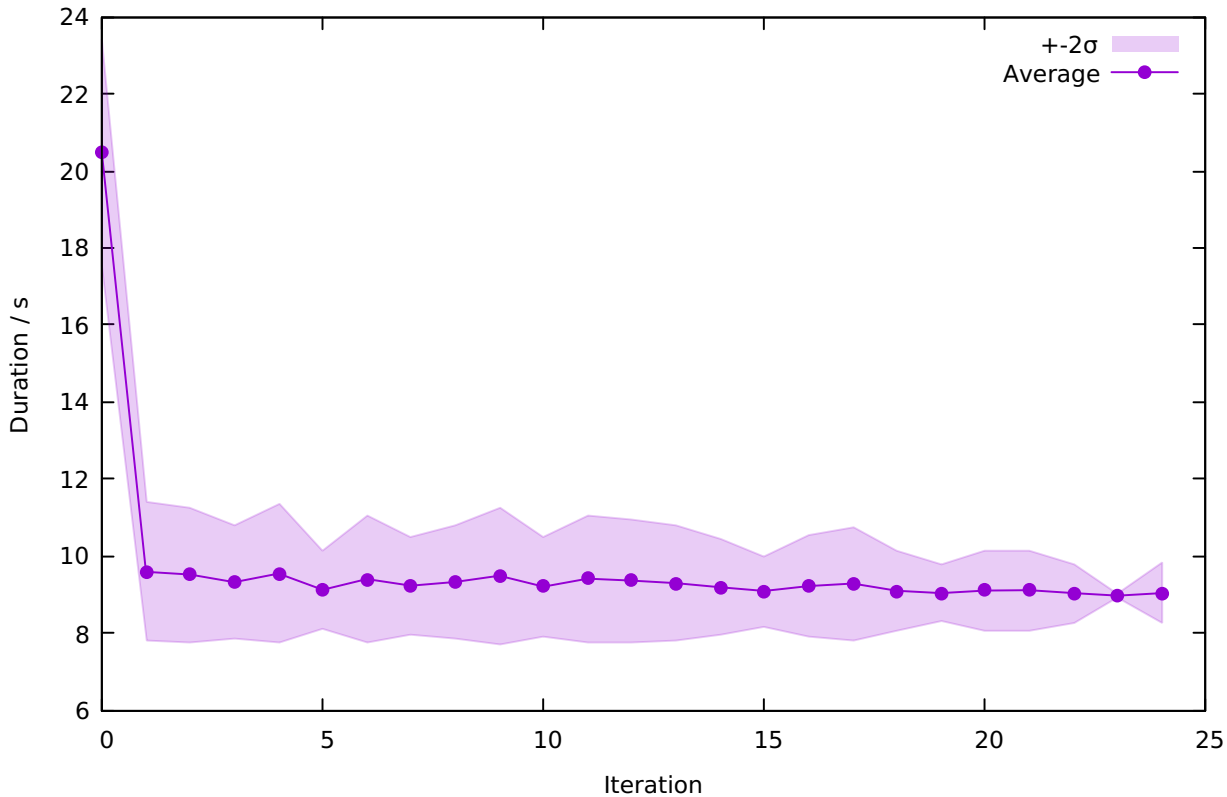


Figure: Average Execution Times (Size 10.000)

ScaDS DRESDEN LEIPZIG

Case Study

ScaDS DRESDEN LEIPZIG

Clusters Setup

- 2 Clusters á 16 Workers
- Both Flink 1.5 with YARN, OpenJDK 1.8

Clusters Setup

- 2 Clusters á 16 Workers
- Both Flink 1.5 with YARN, OpenJDK 1.8

	Worker@Cluster1	Worker@Cluster2
CPUs	2 Xeon®, 2.4GHz	1 Xeon®, 2.5GHz
Model	E5-2620v3	E5-2430v2
RAM	128 GByte	48 GByte
HDDs	5x4TB SATA	2x4TB SATA
Ethernet	10 GBit/s	1 GBit/s
OS	CentOS 7.6	openSUSE 13.2

Table: Hardware of the Clusters Used

Results: Comparison with Local Execution

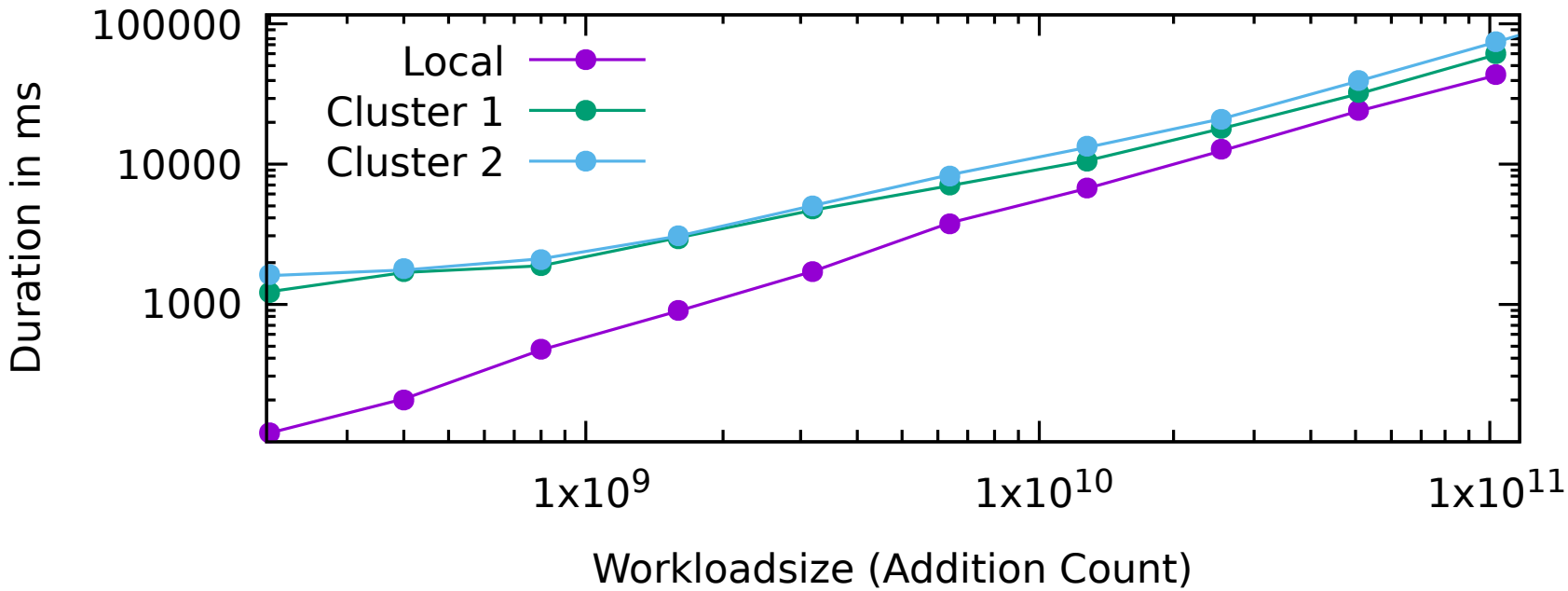


Figure: Add-Benchmark Execution Time

Results: Usage of Parallel Processing

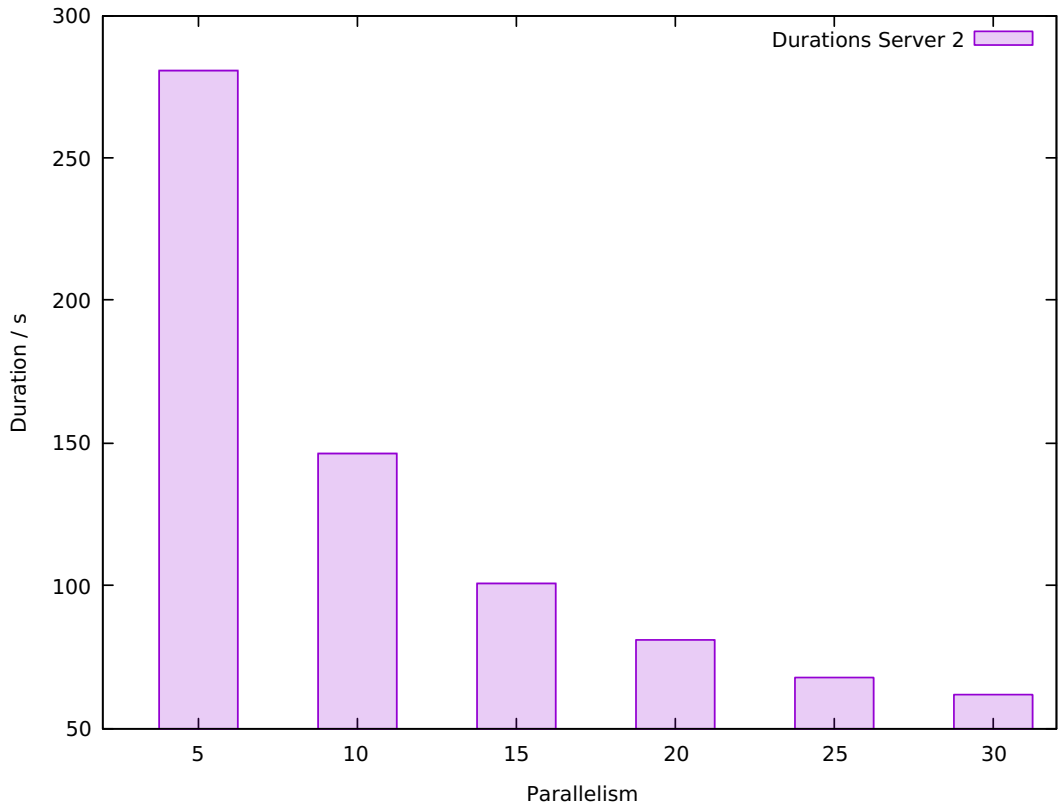


Figure: Histogram of Parallel Execution Durations

Results: I/O Benchmark

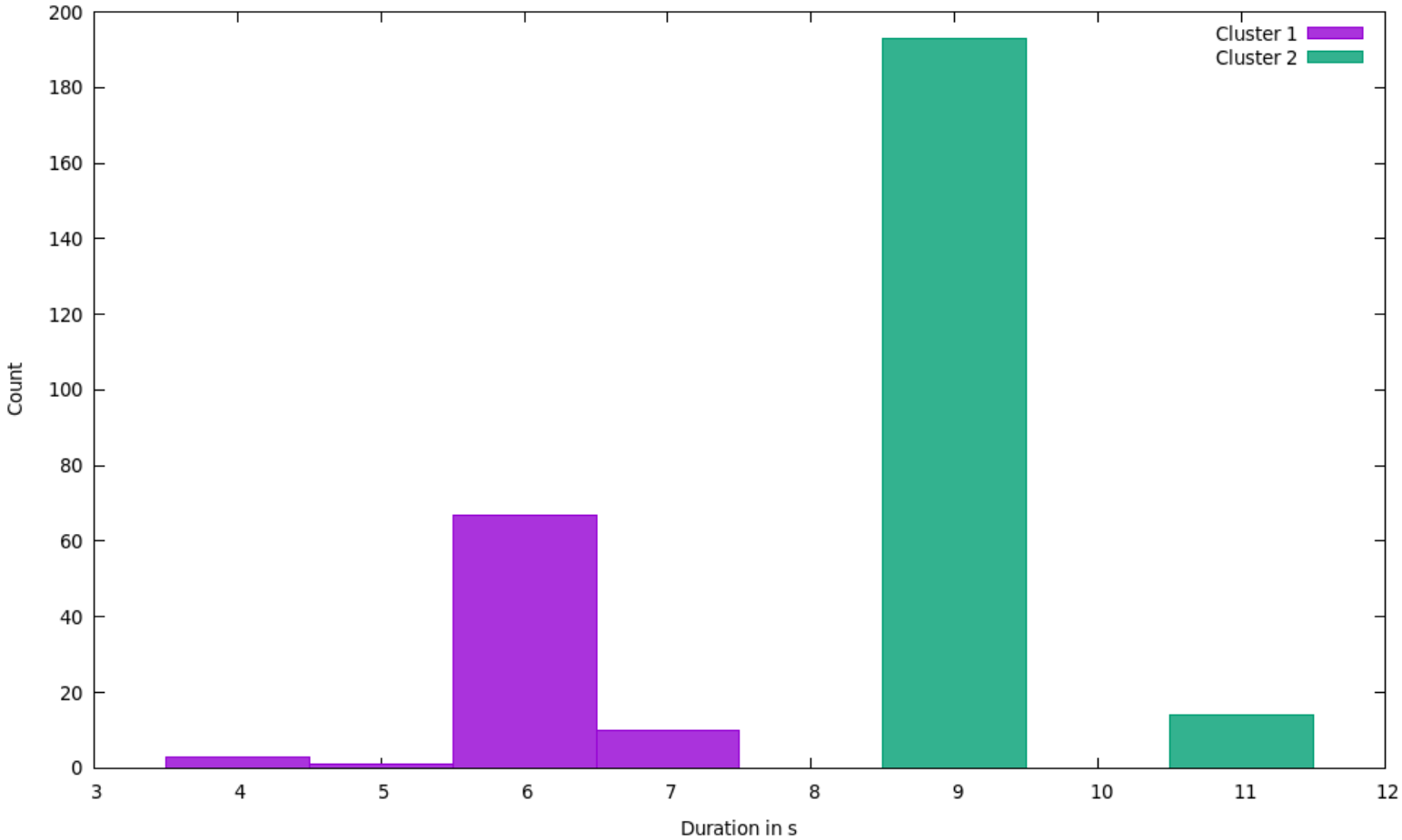


Figure: Write-Benchmark Execution Time

Results: Add Benchmark

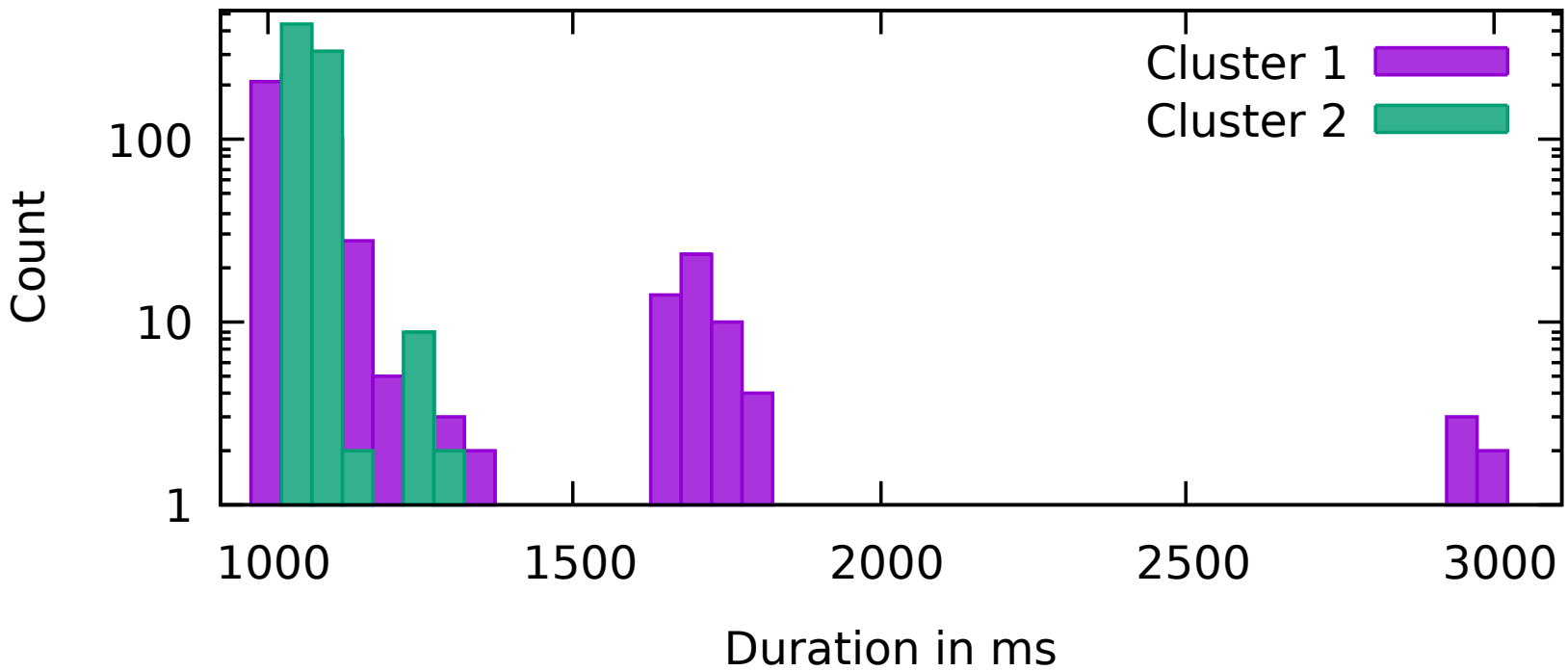


Figure: Histogram of Execution Durations

ScaDS DRESDEN LEIPZIG

Related Work

Related Work

- Benchmarking from users perspective
  - HiBench suite
  - BigBench
  - Production scenarios
- Benchmarking HDFS
  - TestDFSIO
  - Enhanced TestDFSIO

ScaDS DRESDEN LEIPZIG

Summary

ScaDS DRESDEN LEIPZIG

Summary

- Built benchmark from operators perspective
- Goal: Reproduce problems with microbenchmarks
- Case study: Problems reproducible

- Next step: Facilitate root cause analysis
  - Run on standalone cluster
  - Search logs
  - Add workloads

Thanks for your attention!

David Georg Reichelt
Universitätsrechenzentrum
Universität Leipzig
reichelt@informatik.uni-leipzig.de

Funded by:

Hanns Seidel Stiftung

ScaDS DRESDEN LEIPZIG

FKZ 01IS14014B

SPONSORED BY THE
Federal Ministry of Education and Research