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

¹Universität Leipzig, University Computing Centre, Research and Development
1. Apache Flink

2. Microbenchmark Suite

3. Case Study

4. Related Work

5. Summary
Apache Flink
Why Flink?

- Hadoop
- Apache Flink
- Spark
- Mesos
- YARN
- HBase
- Storm

Other options:
- ...
Usage of Flink

```java
DataStream<WordWithCount> windowCounts = text
    .flatMap(new FlatMapFunction<String, WordWithCount>() {
        @Override
        public void flatMap(String value,
                            Collector<WordWithCount> out) {
            for (String word : value.split(\"\s\")) {
                out.collect(new WordWithCount(word, 1L));
            }
        }
    })
    .keyBy(\"word\")
    .timeWindow(Time.seconds(5))
    .reduce(new ReduceFunction<WordWithCount>() {
        @Override
        public WordWithCount reduce(WordWithCount a,
                                    WordWithCount b) {
            return new WordWithCount(a.word,
                                      a.count + b.count);
        }
    });
```
Usage of Flink

```java
DataStream<WordWithCount> windowCounts = text
    .flatMap(new FlatMapFunction<String, WordWithCount>() {
        @Override
        public void flatMap(String value, Collector<WordWithCount> out) {
            for (String word : value.split("\s")) {
                out.collect(new WordWithCount(word, 1L));
            }
        }
    })
    .keyBy("word")
    .timeWindow(Time.seconds(5))
    .reduce(new ReduceFunction<WordWithCount>() {
        @Override
        public WordWithCount reduce(WordWithCount a, WordWithCount b) {
            return new WordWithCount(a.word, a.count + b.count);
        }
    });
```
Usage of Flink

```java
DataStream<WordWithCount> windowCounts = text
   .flatMap(new FlatMapFunction<String, WordWithCount>() {
     @Override
     public void flatMap(String value, Collector<WordWithCount> out) {
       String[] words = value.split("\s");
       for (String word : words) {
         out.collect(new WordWithCount(word, 1L));
       }
     }
   })
   .keyBy("word")
   .timeWindow(Time.seconds(5))
   .reduce(new ReduceFunction<WordWithCount>() {
     @Override
     public WordWithCount reduce(WordWithCount a, WordWithCount b) {
       return new WordWithCount(a.word, a.count + b.count);
     }
   });
```
Usage of Flink

```java
DataStream<WordWithCount> windowCounts = text
  .flatMap(new FlatMapFunction<String, WordWithCount>() {
    @Override
    public void flatMap(String value, Collector<WordWithCount> out) {
      for (String word : value.split("\\s")) {
        out.collect(new WordWithCount(word, 1L));
      }
    }
  })
  .keyBy("word")
  .timeWindow(Time.seconds(5))
  .reduce(new ReduceFunction<WordWithCount>() {
    @Override
    public WordWithCount reduce(WordWithCount a, WordWithCount b) {
      return new WordWithCount(a.word, a.count + b.count);
    }
  });
```
Usage of Flink

```java
DataStream<WordWithCount> windowCounts = text
 .flatMap(new FlatMapFunction<String, WordWithCount>() {
   @Override
   public void flatMap(String value,
                        Collector<WordWithCount> out) {
     for (String word : value.split("\s")) {
       out.collect(new WordWithCount(word, 1L));
     }
   }
 } )
 .keyBy("word")
 .timeWindow(Time.seconds(5))
 .reduce(new ReduceFunction<WordWithCount>() {
   @Override
   public WordWithCount reduce(WordWithCount a,
                                WordWithCount b) {
     return new WordWithCount(a.word,
                              a.count + b.count);
   }
 } );
```
Usage of Flink

```java
DataStream<WordWithCount> windowCounts = text
    .flatMap(new FlatMapFunction<String, WordWithCount>() {
        @Override
        public void flatMap(String value, Collector<WordWithCount> out) {
            for (String word : value.split(">\\s")) {
                out.collect(new WordWithCount(word, 1L));
            }
        }
    })
    .keyBy("word")
    .timeWindow(Time.seconds(5))
    .reduce(new ReduceFunction<WordWithCount>() {
        @Override
        public WordWithCount reduce(WordWithCount a, WordWithCount b) {
            return new WordWithCount(a.word, a.count + b.count);
        }
    });
```
Usage of Flink

Figure: Tools and our Cluster Setup of Flink
Microbenchmark Suite
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
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
Steady state needs to be assured

Assumption: Normal distribution
⇒ 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
⇒ 25 Measurements, use last 10
Assuring Statistic Rigor

- Steady state needs to be assured

Figure: Average Execution Times (Size 10,000)
Case Study
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

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

**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

Apache Flink
Microbenchmark Suite
Case Study
Related Work
Summary

November 11, 2018
Results: Add Benchmark

![Histogram of Execution Durations](image)

**Figure:** Histogram of Execution Durations
Related Work
Related Work

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

FKZ 01IS14014B

November 11, 2018