DPS Assignment 1

Reproducibility Report of Spark and Flink for Stream Processing

Egecan Karakas
s2914123
Leiden University

Tristan Kattenberg
s2508907
Leiden University

1 INTRODUCTION

Distributed Stream Data Processing Systems are the backbone of many modern systems that works on real-time streaming data. Such systems are build with the help of clusters. Clusters are a combination of servers, usually called nodes, that are orchestrated using systems such as Spark [9] or Flink [2] to run applications on. The systems are usually tested in confined testing spaces creating a mismatch between real performance and measured performance [8]. Providing consistent performance is not a given due to the high variability in systems [7]. Furthermore the lack of standard benchmark methodology of parallel computing systems makes reproducibility and accurate measuring a challenge on its own [5]. In this paper we will challenge ourselves by reproducing the performance of the Spark and Flink System as in [6] with Yahoo Streaming Benchmark [3] and report on the claims given by this system.

Our paper is organised as follows. In Section 2 we take a closer look at the original research papers of Spark and FLink to give the necessary background knowledge. In Section 3 we explain the authors methodology and explain what and why we made adjustment to the authors methodology for our duplication. We report our results in Section 4. We conclude with Section 5, 6 containing the discussion and conclusion.

2 BACKGROUND

In this section we will give the general background. This section will start with some previous research on Streaming Data Processing Systems (SDPSs) and high level introduction of the Spark and Flink Distributed Data Processing Systems (DDPS) and move onto benchmark of these Data Processing Systems with a real world use case of Yahoo Streaming Benchmark.

2.1 Flink

Streaming Data Processing (such as Complex Event Processing) and Batch (static) Data Processing have been considered as distinct ends of Data Processing Systems. However, Flink tries to unify these systems in a middle ground via an architectural pattern of "lambda architecture". It uses event based streaming and adopts a stream-first approach such that treats batches as a stream with a limit. In this experiment we focused on performance of Flink from a streaming applications use-case perspective.

Flink Process Model in Figure 1 shows that data sharing in flink happens between task managers and job managers collect heartbeats, task statuses etc. from task managers. To do so, Job Manager calculates the DAG dataflow of applications and deploys it accordingly to the Task Managers. The communication happens through Akka actor models deployed in managers.

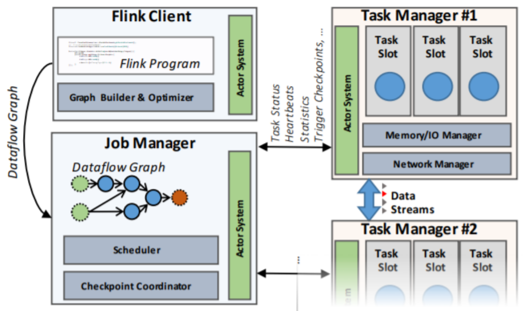

Figure 1: Flink Process Model

2.2 Spark and Spark Streaming

"Resilient Distributed Datasets"[9] introduced RDDs in 2012 to utilise coarse-grained transformations and easy data recovery using lineage. Later, Spark Streaming was introduced in 2014 to allow processing of streaming data by turning it into mini batches 2 to be processed by the Spark System. Even though it seems to have an architecture designed for batch workloads, With the introduction of Spark Streaming, it can still be used for scaling stream processing applications.

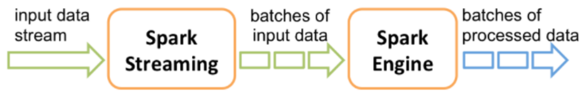

Figure 2: Streaming Data transformed to Mini Batches with Spark Streaming

Yahoo Streaming Benchmark Yahoo Streaming Benchmark[3] is developed at Yahoo to compare their use of Storm and understand its strengths and weaknesses against other applications in streaming landscape. The benchmark consists of 6 steps in a flow, as in Figure 3. First, System Under Test (SUT) reads an event from Kafka persistent message queue. Then it deserializes the JSON string, filter out irrelevant events based on event_type field and take a projection of the relevant fields of ad_id and event_time. Then SUT, joins each event by ad_id with its associated campaign_id which is found in distributed key-value store Redis. Finally, SUT takes a windowed count of events per campaign and store each window in Redis along with a timestamp of the time the window was last updated in Redis. This step is expected to handle late events.

The decision of using this benchmark slightly differs from the experiments in the assignment paper [6]. However, it resembles a widely usable and more acknowledge work in the literature. The reason behind this choice is that data from the paper is not accessible and it's design doesn't include overheads which can occur in real-world scenarios. On the other hand, we acknowledge that YSB

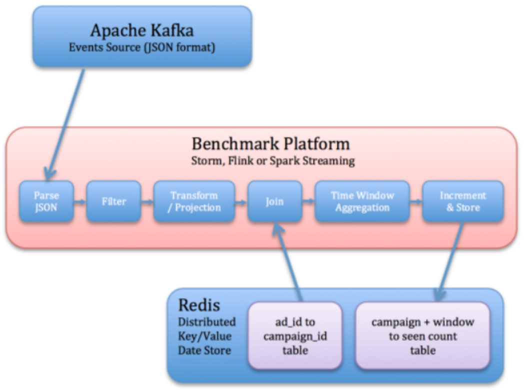
Apache Kafka
Events Source (JSON format)
Benchmark Platform
Storm, Flink or Spark Streaming
Parse JSON
Filter
Transform / Projection
Join
Time Window Aggregation
Increment & Store
Redis
Distributed Key/Value Date Store
ad_id to campaign_id table
campaign + window to seen count table

Figure 3: Design of YSB Benchmark

benchmark has its flaws too. It fails to fully cover DSPS functionalities which is mentioned in ESPBench paper [4] that use more comprehensive applications with business data based on TPC-C. Nonetheless we decided to proceed with YSB benchmark in this reproducibility assignment.

3 METHODOLOGY/DESIGN

This section explains our experiments and motivate our approach. We start by explaining the infrastructure. Secondly we extract the relevant experiments from the paper on RDDs and Spark. In the experiments we explain how to original author proposed the setup and how we tweaked the setup to fit our system. Next we propose our setup for the same experiment. The goal is to duplicate and verify the result or the original paper. Our experiments are publicly available on Github [1].

3.1 DAS-5 Cluster System

The Distributed ASCI Supercomputer 5 (DAS-5) is a distributed system used for research by computer scientists [1]. Using SLURM resource management system[2] we can allocate different amounts of nodes for our experiments. DAS-5 has multiple clusters. We use the Leiden University cluster which has access to 24 cpu nodes, each consisting of: dual 8-cores with 2.4Ghz speed, 64GB Memory, 128 TB storage, 2*4TB local HDD's and using network IB and GBe.

Cluster setup Reproduced paper [6] had a setup with N(2,4,8) nodes for SUT and N(2,4,8) nodes for running the driver (data generator) and queues between driver and SUT. To replicate we used standalone clusters for both Flink and Spark. In our setup, we followed N+N+1 to mediate between YSB Benchmark and reproducability paper where first N nodes are used for SUT, second N is used for kafka message queues and extra node holds data generator, zookeeper master, and redis node. This decision is made to make up the difference from both sides. Because paper worked argues YSB for Kafka and Redis bottleneck but when we exclude these it doesn't really replicate the real-world use cases and no persistence. To reduce the message load of high availability we ran zookeeper with only 1 node even if it is arguable to single point of failure, at the extra node. On the other hand, thankfully, we already have

[1]https://github.com/egecankarakas/Streaming-Benchmark
[2]https://slurm.schedmd.com/documentation.html

	count	mean	std	min	max
Spark 2 Nodes	32100	10329	1892	487	25271
Spark 4 Nodes	51486	10118	1675	253	11416
Spark 8 Nodes	40400	10204	1653	427	11369
Flink 2 Nodes	33531	29243	17539	11156	68821
Flink 4 Nodes	62104	14211	4062	10517	28197
Fink 8 Nodes	68875	10665	1355	1078	13712

Table 1: Latency statistics, mean, standard deviation ,min, max

servers synced in DAS5 cluster with NTP. Finally, we used openjdk version "1.8.0_161" and python version 3.6 on the Centos 7 servers of DAS-5.

3.2 Experiments

In this section we explain what experiments we are running, how they are relevant, what benchmarks we use and go into more detail about the experiments we run.

Experiment choice YSB benchmark comes as a bundle, so it is hard to say that it correctly replicates any experiment discussed in the paper by itself. However, we still wanted to observe it and ran it with 20 repetitions. YSB benchmark both have a join and aggregation phase in contrast to assignment paper which have these separated. Also, sources in assignment papers are both real-time streams compared to redis lookups in the YSB benchmark. Therefore with running YSB benchmark we are running experiment 1 and experiment 2 in the paper with the addition of lookups but abondoning the collision of 2 flowing data sources.

To come up with the missing experiment, we looked for a way to reproduce an experiment in the paper without compromising from the YSB benchmark design. Therefore, as the 2nd experiment we decided to replicate experiment 5 - Fluctuating workloads. For this experiment, the paper starts with a workload of 0.84M/s then reduce it to 0.28M/s and increase it back to 0.84M/s in the end. However, as our benchmark problem is different then theirs, we checked ESPBench [4] where similar comparison is made with 10,000 and 1,000 events per second. Therefore, in this experiment, we use a load of 10k events at first, then lower it down to 1k events, and raise it back 10k events in 40seconds splitted time frames. In this experiments we also use an additional node to generate the reduced workload on a separate machine to work around cold start and finish of the data generator.

4 EVALUATION

In this section we present the results from the paper using K-means; comparing these results with our results. In the discussion we will report on the notable results and give our explanation.

4.1 Experiment 1: Node Number comparison

For experiment 1 we compare the latency Times for both spark and flink. YSB benchmark was used to test both system. Each run of the experiment began by setting up a new Zookeper, Kafka and Spark or flink. Each test was repeated a minimum of 20 times resulting in at least 33,531 separate measurements of latency Time per test.

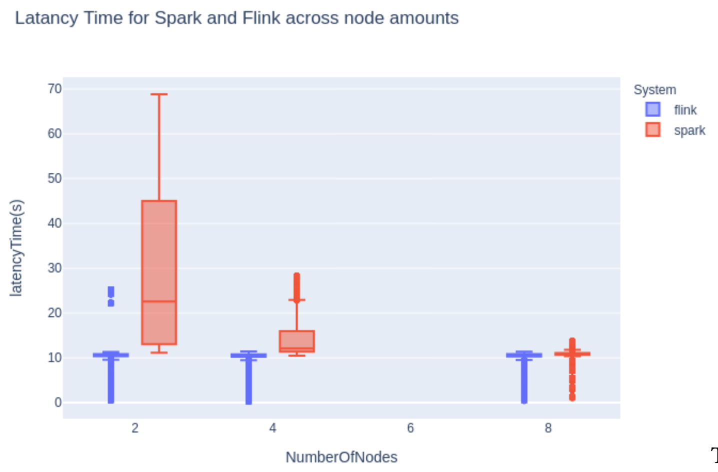

Figure 4: Spark and Flink comparison of latency-Times in seconds

	sum squares	df	F	PR(>F)
Nodes	4.338245e+12	2.0	53202.672626	0.0
System	2.830014e+12	1.0	69412.538631	0.0
Nodes:System	3.695492e+12	2.0	45320.176665	0.0
Residual	1.184355e+13	290490.0	NaN	NaN

Table 2: 2 Sided Anova Test for latency Time Number of Nodes + System + Number of Nodes and System

In Figure 4 our experiment is able to duplication the finding in [6]. The paper says scaling Flink has little effect of the Latency Time and this is support by out data. Looking at 4, Flink has the same, 10 s latency time independent on number of nodes. Spark, on the other hand, benefits from scaling. Comparing a spark cluster with 2 nodes we a latency of approximately 23 second is observed vs a Spark cluster of 8 where spark is has very similarly latency to 8,4 or 2 nodes cluster of Flink.
In order to analyze further what is show in 4 an 2 sided Anova test was performed results are show in **??**. To see if there is a statistically signification different in Mean Latency Time in ms between Number of Nodes, System and Number of Node and System. At a alpha value of .05 we are able to reject the null hypothesis. In order to determine if our observation from 4 are current a Multiple Comparison of Mean Tukey HSD was performed.

The Tukey results 3 show Flink's Mean latency Times between running 2,4, or 8 nodes are not significantly different at an alpha value of 0.05, Sparkk does. For Spark as you increase the amount of nodes the Mean latency Times is significantly from each other. These is a large mean difference between these run again as shown graphically in 4.
Finally examining our best Spark cluster of 8 Nodes never matches the latency Times provided by Flink. Combaring spark 8 with Flink 2,4, and 8; Flink is statically different from spark and the mean difference show Flink has a lower Latency Time in each of these groupings.
Spark improves with scaling and Flink improving is not statistically significant.

group1	group2	meandiff	reject
2flink	2spark	18914.6737	True
2flink	4flink	-210.826	True
2flink	4spark	3881.9217	True
2flink	8flink	-124.8858	False
2flink	8spark	336.3272	True
2spark	4flink	-19125.4997	True
2spark	4spark	-15032.7521	True
2spark	8flink	-19039.5595	True
2spark	8spark	-18578.3465	True
4flink	4spark	4092.7476	True
4flink	8flink	85.9402	False
4flink	8spark	547.1532	True
4spark	8flink	-4006.8074	True
4spark	8spark	-3545.5944	True
8flink	8spark	461.213	True

Table 3: Multiple Comparison of Means - Tukey HSD,FWER=0.05

4.2 Experiment 2

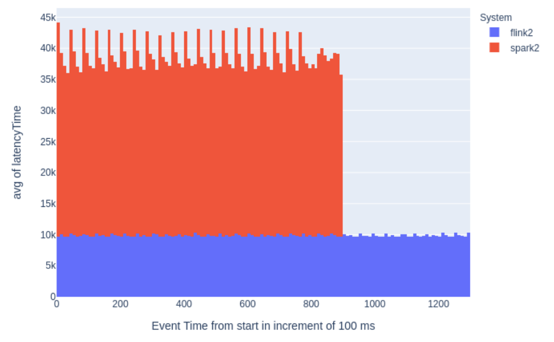

Figure 5: Spark and Flink with 2 nodes, Average Latancy Time per 100 (ms)

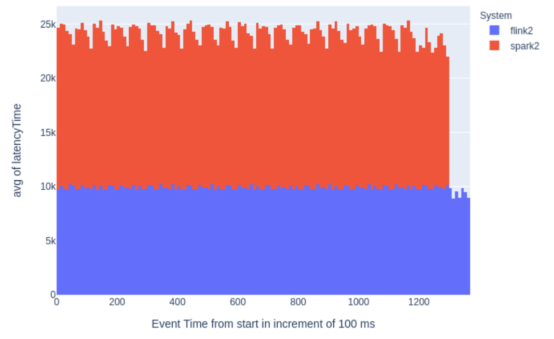

Figure 6: Spark and Flink with 4 nodes, Average Latancy Time per 100 (ms)

These 3 graphs show a comparision between Flink and Spark Average latency Times. At 400 and 800 the input rate is changed from 10k to 1k and then at 800 set back to 10k.

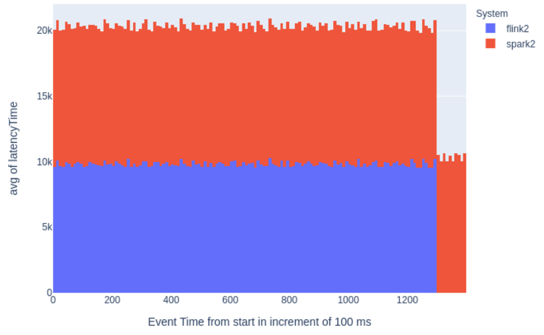

Figure 7: Spark and Flink with 8 nodes, Average Latancy Time per 100 (ms)

5 DISCUSSION

In this section we discuss the results reported in the previous section.

5.1 Experiment 1

Here we discuss a few of the note worthy points our results show. In general we note that similar to the original paper K-means in Spark outperforms Hadoop 2-3 times.

Observable scaling in Spark One of our first noticeable trends is the increase of the time when we increase the nodes. The trend holds for both our approach and the original approach. We think that the computation time necessary is actually small and throwing more computational power, a.k.a more nodes, increases computation time due to the overhead of distributing the systems resources.

Variability and Outliers In Figure **??** the 10 node have a higher variance. We think the variance could a byproduct of the higher server usage of the cluster during our experiment by other users. Otherwise the variance stays unexplainable, although there is variance there are almost no outliers which seem to indicate our results are reliable. In the 15 node setup we see two measurements that are 40 seconds faster than the remaining measurements. Locality of the data processing could be one of the reasons or the system has some caching implemented that increased the performance.

5.2 Experiment 2

In general we only have two points to discuss which are mentioned below, we do see that 10x fold increase of data does increase the amount of time necessary for Spark to compute, however the performance is better than we expected. The original paper shows similar results; as their the data size increases the time computations remain feasible.

6 CONCLUSION

Reproducibility is one of the foundations of science. The mismatch between reported performance and actual performance with distributed systems can be noticeable. The paper was designed to push the systems to their limits however YSB benchmarks is more about observing the systems against real world use cases where other ecosystem components matter and cannot be excluded to just push SUT to its limits. However, with the many observations from the paper, Flinks flaws were more apparent which remained to be discovered later in our reproducibility experiment. Another important take-out is that, repeating tests shows the variability of systems much better. If we haven't repeated this many times, it would be really hard to correctly observe differences between min and max performances specifically in spark's case. On the other hand, even it the experiments fail to show true limits of flink, it shows that it can be relied on some SLAs as it shows low latency difference in benchmarks.

Our experiments in this paper ran on the DAS-5 system. We ran two experiments. First K-means which the author claimed performance of 2-3 times that of MapReduce with Hadoop. Secondly the author clams data mining on large datasets is very feasible and showing a linear performance with the increase of data. To reproduce the experiments we fit the original setup as much as possible to our current infrastructure and ran the experiments with our own improvements. The unexplained parameters of the original paper were left to system defaults.

Our final results for K-means show similar performance to the authors original claims both in the original setup and our setup.Increasing the amount of nodes does not improve computation time much for both Hadoop and Spark. Furthermore increasing hte computational power of individual nodes similarly does not signify an noticeable increase in performance. The interactive data mining showed varying outliers and variability in the measurements. This shows how measurements can have high unexplained variability even in closed cluster environments where variability is easier to explain. However even taken the unexplained variability we can still see that the claim of linear increase of computational time holds for the data mining.

In conclusion our paper reproduced the original authors experiments. Despite the ambiguity of specific experimental setups enough details was explained to reproduce the original results of the paper.

7 FUTURE WORK

In case of future work we would take a closer look at the in explainable variability of our current results. With sufficient time we would increase the sample size of our measurements. Furthermore we would perform our experiments on a computationally more expensive K-means as that current setup does not show an performance gained with the increase of nodes. Finally we would tweak the experimental code to perform the experiments with less tweaking.

REFERENCES

[1] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. 2016. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *Computer* 49, 05 (may 2016), 54–63. https://doi.org/10.1109/MC.2016.127

[2] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* 38 (2015), 28–38.

[3] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Peng, and Paul Poulosky. 2016. Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming. *2016 IEEE International Parallel and Distributed*

Processing Symposium Workshops (IPDPSW) (2016), 1789–1792.

[4] Guenter Hesse, Christoph Matthies, Michael Perscheid, Matthias Uflacker, and Hasso Plattner. 2021. ESPBench: The Enterprise Stream Processing Benchmark. *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Apr 2021). https://doi.org/10.1145/3427921.3450242

[5] T. Hoefler and R. Belli. 2015. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1145/2807591.2807644

[6] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. 2018. Benchmarking Distributed Stream Data Processing Systems. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 1507–1518. https://doi.org/10.1109/ICDE.2018.00169

[7] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming Performance Variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 409–425. https://www.usenix.org/conference/osdi18/presentation/maricq

[8] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2019. Is Big Data Performance Reproducible in Modern Cloud Networks? (2019). arXiv:cs.PF/1912.09256

[9] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 15–28.

A TIMETABLE

In Table 4 we show the estimated time spent.

Table 4: Who did what and how much hours spent

Activity	Egecan	Tristan
Github	2	1.5
DAS-5	20	10
Coding	30	25
Read Paper	4	3
Write Report	15	10
Experiments	12	6