Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor's Thesis Nr. 197b

Systems Group, Department of Computer Science, ETH Zurich

Implementation of a Benchmark Suite for Strymon

by

Nicolas Hafner

Supervised by

Dr. John Liagouris
Prof. Timothy Roscoe

November 2017 - May 2018

Informatik
Computer Science

Abstract

A real abstract kind of text

Acknowledgements

Cool people

Contents

1 Introduction

2 Related Work

In this section we analyse and compare a number of papers about stream processors. In particular, we look at the ways in which they evaluate and test their systems in order to get an idea of how benchmarking has so far commonly been done. Each subsection looks at one paper at a time, providing a graph of the data flows and operators used to evaluate the system, if such information was available.

2.1 S4: Distributed stream computing platform[1]

The S4 paper evaluates its performance with two sample algorithms: click-through rate (CTR), and online parameter optimisation (OPO). The system uses typed events between nodes and distributes them to physical nodes based on a "key property" on the event.

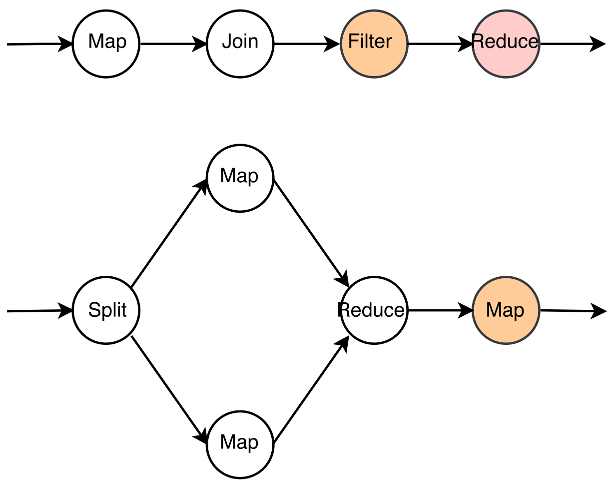

Figure 1: Graphs of the two tests used to evaluate the S4 platform: click-through rate and online parameter optimisation. Orange nodes are stateful. Nodes coloured in red retain state of previous stream events.

The CTR test is implemented by four nodes: the first assigns keys to the initially keyless events coming in. It passes them to a node that combines matching events. From there the events go on to a filter that removes unwanted events. Finally, the events are passed to a node that computes the CTR, and emits it as a new event.

The OPO test consists of five nodes: the first node assigns keys to route the events to either the second or third node. The second and third nodes perform some computations on the events and emit the results as new ones. The fourth node compares the events it gets from nodes two and three, to determine the optimisation parameters. The fifth node runs an adaptation depending on the parameters it receives.

2.2 SPADE: the system s declarative stream processing engine[2]

In order to evaluate the system, a simple algorithm is run to determine bargains to buy. The system uses typed data payloads between nodes. The distribution and allocation of physical nodes is automatically performed by the system.

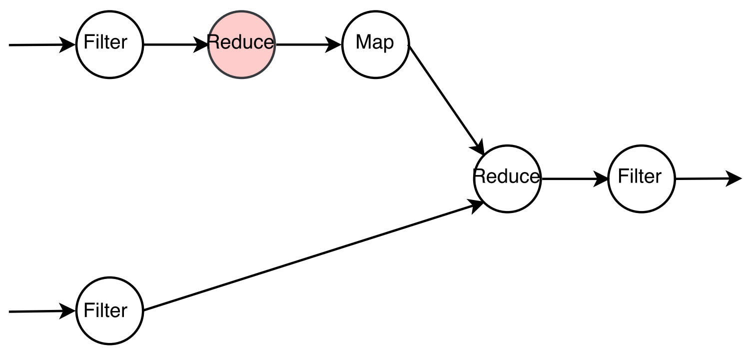

Figure 2: Graph of the example used in the SPADE paper: a bargain index computation. Nodes coloured in red retain state of previous stream events.

The data flow is composed of six nodes: the first filters out trade information and computes its price. It passes its information on to a moving aggregation node, with a window size of 15, and a slide of 1. The aggregate is passed on to a mapping node that computes the volume weighted average price (VWAP). The fourth node filters out quote information. This is then, together with the VWAP, passed to the fifth node, which joins the two together to compute the bargain index. The final node simply filters out the zero indexes.

2.3 Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters[3]

In the Discretized Streams paper, the performance is evaluated through a simple Word Count algorithm. The system operates on tuples of data, and uses a rather simple API of predefined operators to construct a data flow.

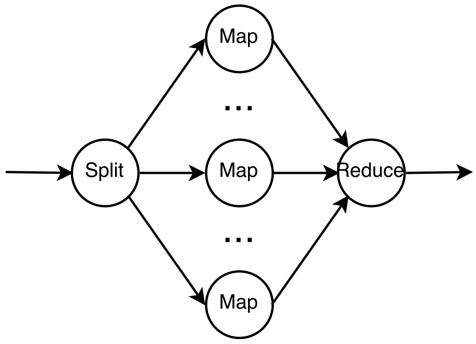

Figure 3: Graph of the Word Count example used to illustrate the discretized streams. Not shown is the incremental reduction done between batches.

The Word Count test is implemented through three operators: a “flat map” that splits an incoming string into words, a map that turns each word into a tuple of the word and a counter, and finally a hopping-window aggregation that adds the counters together grouped by word.

2.4 MillWheel: fault-tolerant stream processing at internet scale[4]

The Millwheel paper unfortunately provides barely any information at all about the tests implemented. The only mention is about how many stages the pipelines have they use to evaluate the system. Two tests are performed: a single-stage test to measure the latency, and a three-stage test to measure the lag of their fault tolerance system.

The system follows a similar model to S4, where nodes handle typed data, and the data is clustered by a key property. Unlike S4 where the key is a direct property of the data, in MillWheel “key extractors” of each node provide the keys, and thus the same data can be separated into multiple clusters.

2.5 Bigdatabench: A big data benchmark suite from internet services[5]

This paper proposes a suite of benchmarks and tests to evaluate Big Data systems. The paper primarily focuses on the generation of suitable testing data sets, and proposes the following algorithms to test the system:

- Sort
- Grep
- Word Count
- Retrieving Data
- Storing Data
- Scanning Data
- Select Query
- Aggregate Query
- Join Query
- Nutch Server
- Indexing
- Page Rank
- Olio Server
- K-means
- Connected Components
- Rubis Server
- Collaborative Filtering
- Naive Bayes

The paper does not propose any particular implementation strategies. They provide performance evaluation for an implementation of different parts of the benchmark suite on the Hadoop, MPI, Hbase, Hive, and MySQL systems, but no particular details of the implementation are discussed.

2.6 Streamcloud: An elastic and scalable data streaming system[6]

In this paper, the system is evaluated by two distinct queries. It is not stated whether either of the queries have any real-world application. The system declares a set of nodes that operate on tuples. These predefined nodes can then be connected together to perform a query.

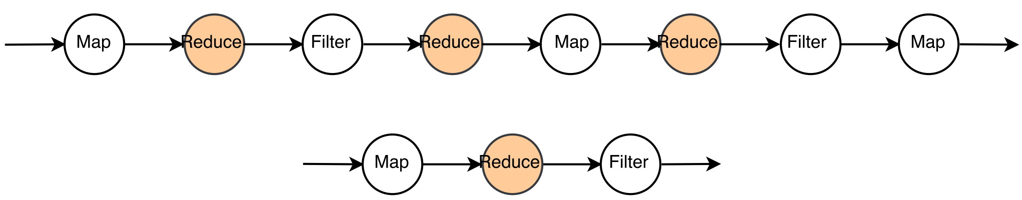

Figure 4: Graph of the query used to evaluate StreamCloud. Nodes coloured in orange retain some state.

Both queries perform a sequence of maps and filters followed by aggregations. The aggregate is based on a window size and slide, which can be configured for each node. However, the configurations used are not provided by the paper.

2.7 Integrating scale out and fault tolerance in stream processing using operator state management[7]

To evaluate their approach for fault tolerance using Operator State Management, two queries were implemented: a linear road benchmark to determine tolls in a network, and a Top-K query to determine the top visited pages. The data flow is composed out of stateless and stateful nodes, where stateful nodes use explicitly declared state variables. Data is exchanged between nodes in the form of tuples.

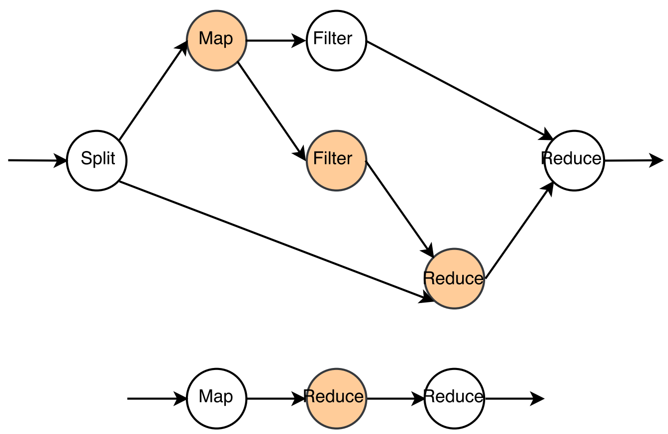

Figure 5: Illustration of the queries for the Linear Road Benchmark and the Top-K tests used to evaluate their system. Orange nodes maintain internal state.

2.8 Timestream: Reliable stream computation in the cloud[8]

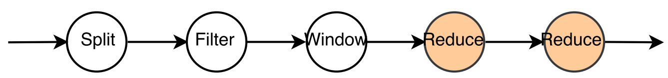

Figure 6: The Distinct Count query used to evaluate the Timestream system. Nodes in orange have internal state.

The sentiment analysis has no description of the graph used and we could thus not replicate it here.

2.9 Adaptive online scheduling in Storm[9]

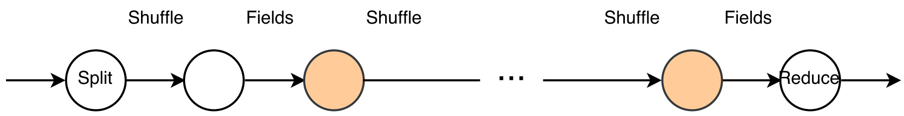

Figure 7: The topology graph used to evaluate the Storm schedulers. Orange nodes are stateful. “Shuffle” connections send the output event to a random destination node. “Fields” connections send the output event to a specific destination node based on a field of the event.

2.10 Big data analytics on high Velocity streams: A case study[10]

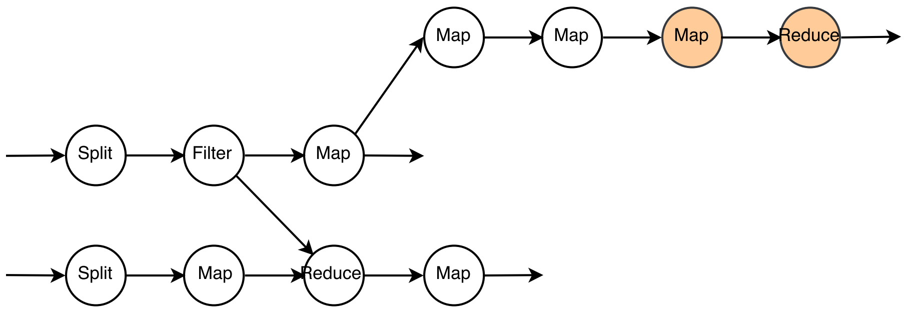

Figure 8: An illustration of the topology used for the Twitter & Bitly link trend analysis. Nodes coloured orange maintain internal state.

2.11 Comparison

In Table 1 and Table 2 we compare the most important features of the tests performed in the various papers. Unfortunately, most of the papers do not supply or use publicly available data, making it difficult to compare them, even if the test data flows were replicated.

Paper	Goal	Application	Dataflow Properties	Dataflow Operators
S4[1]	A practical application of the system to a real-life problem.	Search	Stateful	Map, Filter, Join
SPADE[2]	Sample application, performance study.	Finance	Stateful	Map, Filter, Reduce, Join
D-Streams[3]	Scalability and recovery test.	None	None	Map, Reduce, Window
Millwheel[4]	In-Out Latency.	Ads	Unspecified	Unspecified
BigDataBench[5]	Fair performance evaluation of big data systems.	Search, Social, Commerce	Unspecified	Unspecified
StreamCloud[6]	Evaluation of scalability and elasticity.	Telephony	Stateful	Map, Filter, Reduce, Join
Operator State[7]	Testing dynamic scaling and fault-tolerance.	Road tolls	Stateful	Map, Reduce, Join
TimeStream[8]	Low-latency test for real-world applications.	Search, Social Network	None	Map, Filter, Reduce, Window
Adaptive Scheduling[9]	Evaluating performance of scheduling algorithms.	None	Stateful	None
Analytics on High Velocity Streams[10]	Analysing trends for links on Twitter.	Social Network	Stateful	Map, Filter, Reduce, Window
YSB[11]	Benchmarking streaming systems via Ad analytics.	Ads	Stateful	Map, Filter, Reduce, Join, Window
HiBench[12]	Evaluating big data processing systems.	Big Data	Stateful	Map, Reduce, Window
NEXMark[13]	Adaptation of XMark for streaming systems.	Auctioning	**TODO**	**TODO**

Table 1: Comparison of the test properties of the reference papers.

Paper	Workloads	Testbed	External Systems	Public Data
S4[1]	~1M live events per day for two weeks.	16 servers with 4x32-bit CPUs, 2GB RAM each.	Unspecified	No
SPADE[2]	~250M transactions, resulting in about 20GB of data.	16 cluster nodes. Further details not available.	IBM GPFS	Maybe[1]
D-Streams[3]	~20 MB/s/node (200K records/s/node) for WordCount.	Up to 60 Amazon EC2 nodes, 4 cores, 15GB RAM each.	Unspecified	No
Millwheel[4]	Unspecified.	200 CPUs. Nothing further is specified.	BigTable	No
BigDataBench[5]	Up to 1TB.	15 nodes, Xeon E5645, 16GB RAM, 8TB disks each.	Hadoop, MPI, Hbase, Hive, MySQL	Yes[2]
StreamCloud[6]	Up to 450’000 transactions per second.	100 nodes, 32 cores, 8GB RAM, 0.5TB disks each, 1Gbit LAN.	Unspecified	No
Operator State[7]	Up to 600’000 tuples/s.	Up to 50 Amazon EC2 “small” instances with 1.7GB RAM.	Unspecified	No
TimeStream[8]	~30M URLs, ~1.2B Tweets.	Up to 16 Dual Xeon X3360 2.83GHz, 8GB RAM, 2TB disks each, 1Gbit LAN.	Unspecified	No
Adaptive Scheduling[9]	Generated.	8 nodes, 2x2.8GHz CPU, 3GB RAM, 15GB disks each, 10Gbit LAN.	Nimbus, Zookeper	Yes[3]
Analytics on High Velocity Streams[10]	~1’600GB of compressed text data.	4 nodes, Intel i7-2600 CPU, 8GB RAM each.	Kafka, Cassandra	Maybe[4]
YSB[11]	Generated	Unspecified	Kafka, Redis	Yes[5]
HiBench[12]	Generated	Unspecified	Kafka	Yes[6]
NEXMark[13]	Generated	Unspecified	Firehose Stream Generator	Yes[7]

Table 2: Comparison of the test setups of the reference papers.

[1] The data was retrieved from the IBM WebSphere Web Front Office for all of December 2005.
[2] Obtainable at http://prof.ict.ac.cn/BigDataBench/
[3] The data is generated on the fly, the algorithm of which is specified in the paper.
[4] Data stems from Twitter and Bit.ly for June of 2012, but is not publicly available.
[5] Generated by YSB: https://github.com/yahoo/streaming-benchmarks
[6] Generated by HiBench.
[7] Generated by the “Firehose Stream Generator”.

3 Timely Dataflow

4 Yahoo Streaming Benchmark (YSB)[11]

4.1 Implementation

4.2 Evaluation

5 HiBench: A Cross-Platforms Micro-Benchmark Suite for Big Data[12]

5.1 Implementation

5.2 Evaluation

6 NEXMark Benchmark[13]

6.1 Implementation

6.2 Evaluation

7 Conclusion

References

[1] Leonardo Neumeyer et al. "S4: Distributed stream computing platform". In: *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE. 2010, pp. 170–177.
URL: https://pdfs.semanticscholar.org/53a8/7ccd0ecbad81949c688c2240f2c0c321cdb1.pdf.

[2] Bugra Gedik et al. "SPADE: the system s declarative stream processing engine". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pp. 1123–1134.
URL: http://cs.ucsb.edu/~ckrintz/papers/gedik_et_al_2008.pdf.

[3] Matei Zaharia et al. "Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters". In: *HotCloud* 12 (2012), pp. 10–10.
URL: https://www.usenix.org/system/files/conference/hotcloud12/hotcloud12-final28.pdf.

[4] Tyler Akidau et al. "MillWheel: fault-tolerant stream processing at internet scale". In: *Proceedings of the VLDB Endowment* 6.11 (2013), pp. 1033–1044.
URL: http://db.cs.berkeley.edu/cs286/papers/millwheel-vldb2013.pdf.

[5] Lei Wang et al. "Bigdatabench: A big data benchmark suite from internet services". In: *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. IEEE. 2014, pp. 488–499.
URL: https://arxiv.org/pdf/1401.1406.

[6] Vincenzo Gulisano et al. "Streamcloud: An elastic and scalable data streaming system". In: *IEEE Transactions on Parallel and Distributed Systems* 23.12 (2012), pp. 2351–2365.
URL: http://oa.upm.es/16848/1/INVE_MEM_2012_137816.pdf.

[7] Raul Castro Fernandez et al. "Integrating scale out and fault tolerance in stream processing using operator state management". In: *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. ACM. 2013, pp. 725–736.
URL: http://openaccess.city.ac.uk/8175/1/sigmod13-seep.pdf.

[8] Zhengping Qian et al. "Timestream: Reliable stream computation in the cloud". In: *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM. 2013, pp. 1–14.
URL: https://pdfs.semanticscholar.org/9e07/4f3d1c0e6212282818c8fb98cc35fe03f4d0.pdf.

[9] Leonardo Aniello, Roberto Baldoni, and Leonardo Querzoni. "Adaptive online scheduling in Storm". In: *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM. 2013, pp. 207–218.
URL: http://midlab.diag.uniroma1.it/articoli/ABQ13storm.pdf.

[10] Thibaud Chardonnens et al. "Big data analytics on high Velocity streams: A case study". In: *Big Data, 2013 IEEE International Conference on*. IEEE. 2013, pp. 784–787.
URL: https://www.researchgate.net/profile/Philippe_Cudre-Mauroux/publication/261281638_Big_data_analytics_on_high_Velocity_streams_A_case_study/links/5891ae9592851cda2569ec2b/Big-data-analytics-on-high-Velocity-streams-A-case-study.pdf.

[11] Sanket Chintapalli et al. "Benchmarking streaming computation engines: Storm, Flink and Spark streaming". In: *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE. 2016, pp. 1789–1792.
URL: http://ieeexplore.ieee.org/abstract/document/7530084/.

[12] Intel. *HiBench is a big data benchmark suite*.
URL: https://github.com/intel-hadoop/HiBench.

[13] Pete Tucker et al. *NEXMark–A Benchmark for Queries over Data Streams (DRAFT)*. Tech. rep. Technical report, OGI School of Science & Engineering at OHSU, Septembers, 2008.
URL: http://datalab.cs.pdx.edu/niagara/pstream/nexmark.pdf.