Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Yangjun Wang

Stream Processing Systems Benchmark: StreamBench

Master's Thesis
Espoo, Nov 20, 2015

DRAFT! — January 10, 2016 — DRAFT!

Supervisors: Assoc. Prof. Aristides Gionis
Advisor: D.Sc. Gianmarco De Francisci Morales

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Yangjun Wang

Title:
Stream Processing Systems Benchmark: StreamBench

Date: Nov 20, 2015 **Pages:** 26

Major: Foundations of Advanced Computing **Code:** T-110

Supervisors: Assoc. Prof. Aristides Gionis

Advisor: D.Sc. Gianmarco De Francisci Morales

Batch processing technologies(Such as MapReduce, Hive, Pig) have matured and been widely used in the industry. These systems solved the issue processing big volumes of data successfully. However, first big data need to be collected and stored in a database or file system. Then it takes time to finish batch processing analysis job before get any results. While there are many cases that need analysed results from streaming data in seconds or sub-seconds. To satisfy the increasing demand of processing stream data, several streaming processing systems are implemented and widely adopted, such as Apache Storm, Apache Spark, IBM InfoSphere Streams and Apache Flink. They all support online stream processing, high scalability, and tasks monitoring. While how to evaluate a stream processing system before choosing it to use in production development is a open question.

In this thesis, we introduce StreamBench, a benchmark framework to facilitate performance comparisons of stream processing systems. A common API component and a core set of workloads are defined. We implement the common API and run benchmarks for three widely used open source stream processing systems: Apache Storm, Spark Streaming and Flink Streaming. Therefore, a key feature of the StreamBench framework is that it is extensible – it supports easy definition of new workloads, in addition to making it easy to benchmark new stream processing systems.

Keywords: Big Data, Stream, Benchmark, Storm, Flink, Spark

Language: English

Acknowledgements

I want to thank Professor Aristides Gionis and my advisor Gianmarco De Francisci Morales for their good guidance.

Espoo, Nov 20, 2015

Yangjun Wang

Abbreviations and Acronyms

Symbols

$\mathbf{B}$	magnetic flux density
c	speed of light in vacuum $\approx 3 \times 10^8$ [m/s]
ω_{D}	Debye frequency
ω_{latt}	average phonon frequency of lattice
$\uparrow$	electron spin direction up
$\downarrow$	electron spin direction down

Operators

$\nabla \times \mathbf{A}$	curl of vectorin $\mathbf{A}$
$\frac{\mathrm{d}}{\mathrm{d}t}$	derivative with respect to variable t
$\frac{\partial}{\partial t}$	partial derivative with respect to variable t
$\sum_i$	sum over index i
$\mathbf{A} \cdot \mathbf{B}$	dot product of vectors $\mathbf{A}$ and $\mathbf{B}$

Abbreviations

DFS	Distribute File System
HDFS	Hadoop Distribute File System
GFS	Google File System
YCSB	Yahoo Cloud Serving Benchmark

Contents

List of Figures

Chapter 1

Introduction

Along with the rapid development of information technology, the speed of data generation increases dramatically. To process and analysis such large amount of data, the so-called Big Data, cloud computing technologies get a quick development, especially after these two papers related to MapReduce and BigTable published by Google [5, 10].

1.1 Stream Processing and Evaluation

In theory, Big Data don't only mean "big" **v**olume. Besides volume, Big Data still have two other properties: **v**elocity and **v**ariety [12]. Velocity means the amount of data is growing at high velocity. Variety refers to the various data formats. They are called three ***V***s of Big Data. When deal with Big Data, there are two types of processing model, batch processing and stream processing. A big data architecture contains several parts. For batch processing, masses of structured and semi-structured historical data are stored in HDFS (**V**olume + **V**ariety). On the other side, stream processing is used for fast data requirements (**V**elocity + **V**ariety)[36].

Batch processing is generally more concerned with throughput than latency of individual components of the computation. In batch processing, data is collected and stored in file system. When the size of data reaches a tradeoff, batch jobs could be configured to run without manual intervention, trained against entire dataset at scale in order to produce output in the form of computational analyses and data files. Because of time consume in data collection and processing stage, depending on the size of the data being processed and the computational power of the system, output can be delayed significantly. Generally, latency could be range from minutes to hours.

Streaming processing is required for many practical cases which need

analysed results from streaming data in a very short latency. For example, a online shopping website would want give a customer accurate recommendations as soon as possible after he/she scan the website for a while. Several streaming processing systems are implemented and widely adopted, such as Apache Storm, Apache Spark, IBM InfoSphere Streams and Apache Flink. They all support real-time stream processing, high scalability, and awesome monitoring.

How to evaluate a real time stream processing system before choosing it to use in production development is a open question. Before these real time stream processing systems are implemented, Michael demonstrated the 8 requirements[34] of real-time stream processing, which gives us a standard to evaluate whether a real time stream processing system satisfies these requirements. A very common and traditional approach to verify whether the performance of a system meets the requirements is benchmarking. Published benchmarking results from industry standard benchmark systems could help users compare products and understand features of a system easily. In this thesis, we introduce a benchmark framework called StreamBench to facilitate performance comparisons of stream processing systems.

1.2 Structure of the Thesis

The main topic of this thesis is stream processing systems benchmark. First, cloud computing and benchmark technology background is introduced in Chapter 2. Chapter 3 presents architecture and main features of three widely used stream processing systems. In Chapter 4, we demonstrate the design of our benchmark framework – StreamBench, including the whole architecture, test data generator and extensibility of StreamBench. Experiments and results are discussed in Chapter 5. At last, conclusions are given in Chapter 6.

Chapter 2

Background

This thesis focuses on building a benchmark framework for stream processing systems, which aim to solve issues related to **V**elocity and **V**ariety of big data. [36] Therefore, it is easy to know that the stream processing technology belongs to cloud computing technologies and it has many common features of cloud computing technologies. In the first section of this chapter, we will discuss some background knowledge about distributed computing. Widely accepted benchmark systems of other computer science area and an existing benchmark of stream processing systems are demonstrated in the second chapter.

2.1 Cloud Computing

Cloud computing, also known as 'on-demand computing', is a kind of Internet-based computing, where shared resources, data and information are provided to computers and other devices on-demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources. [28, 30] Cloud computing and storage solutions provide users and enterprises with various capabilities to store and process their data in third-party data centers. [22] Users could use computing and storage resources as need elastically and pay according to the amount of resources used. In another way, we could say cloud computing technologies is a collection of technologies to provide elastic "pay as you go" computing. That includes scalable file system and data storage such as Amazon S3 and Dynamo, scalable processing such as MapReduce and Spark, visualization of computing resources and distributed consensus etc.

2.1.1 Parallel Computing

Parallel computing is a computational way in which many calculations participate and simultaneously solve a computational problem, operating on the principle that large problems could be divided into smaller ones and smaller problems could be solved at the same time. Base on the level of parallelism, parallel computing could be separated as bit-level, instruction-level, and task parallelism. In the case of bit-level and instruction-level parallelism, parallelism is transparent to the programmer. Parallelism discussed in this thesis is task parallelism. Compared to serial computation, parallel computing has the following features: multiple CPUs, distributed parts of the problem, concurrent execution on each compute node. Because of these features, parallel computing obtains better performance than serial computing.

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple computers to work on the same task. Therefore, when the need for parallelism arises, there are two different ways to do that. The first way is "Scaling Up", in which a single powerful computer is added with more CPU cores, more memory, and more hard disks. The other way is dividing task between a large number of less powerful machines with (relatively) slow CPUs, moderate memory amounts, moderate hard disk counts, which could be called "Scaling out". Scalable cloud computing is trying to exploiting "Scaling Out" instead of "Scaling Up".

2.1.2 Computing Cluster

A computer cluster consists of a set of computers which are connected to each other and work together so that, in many respects, they can be viewed as a single system. The components of a cluster are usually connected to each other through fast local area networks ("LAN"), with each node running its own instance of an operating system. In most circumstances, all of the nodes use the same hardware and the same operating system and are set to perform the same task, controlled and scheduled by software. The large number of less powerful machines mentioned above is a computer cluster.

Most computing clusters have two different type of nodes, master nodes and slave nodes. Generally, users only interact with the master node which is a specific computer managing slaves and scheduling tasks. Slave nodes are not available to users which makes the whole cluster as a single system. It is possible that each nodes in a cluster could be failed with some probability. To avoid master node be the single point of failure, master server may be

further divided to active master and standby master (passive master) in order to meet the requirement of fault tolerance. When a slave node failed, master node will reassign the running task in the failed node to another slave node.

As the features of computing cluster demonstrated above, it is usually used to improve performance and availability over that of a single computer. In most cases, the average computing ability of a node is less than a single computer as scheduling and communication between nodes consume resources.

2.1.3 Batch Processing and Stream Processing

According to the size of data processed per unit, processing model could be classified to two categories: batch processing and stream processing. Batch processing is very efficient in processing high **V**olume data. Where data is collected, entered to the system, processed as a unit and then results are produced in batches. The output is another batch that can be reused for computation. Batch jobs are configured to run without manual intervention, trained against entire dataset at scale in order to produce output in the form of computational analyses and data files. Depending on the size of the data being processed and the computational power of the computing cluster, the latency of a task could be measured in minutes or more. MapReduce and Spark are two widely used batch processing model.

In contrast, stream processing emphasizes on the **V**elocity of big data. It involves continual input and outcome of data. Each records in the data stream is processed as a unit. Therefore, data could be processed within small time period or near real time. Streaming processing gives decision makers the ability to adjust to contingencies based on events and trends developing in real-time. Storm is one representative stream processing model.

Except batch processing and stream processing, between them there is another processing model called mini-batch processing. Instead of processing the streaming data one record at a time, mini-batch processing model discretizes the streaming data into tiny, sub-second mini-batches. Each mini-batch is processed as a batch task. As each batch is very small, mini-batch processing obtains much better latency performance than batch processing.

	Batch	Stream
Latency	mins - hours	milliseconds - seconds
Throughput	Large	Small(relatively)
Prior **V**	Volume	Velocity

Table 2.1: Comparison of batch processing and stream process

2.1.4 MapReduce

MapReduce is a parallel programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster of commodity hardware in a reliable, fault-tolerant manner. [10] To achieve the goal, there are two primitive parallel methods (map and reduce) predefined in MapReduce programming model. A MapReduce job usually executes map tasks first to split the input data-set into independent chunks and perform map operations on each chuck in a completely parallel manner. In this step, MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted. The final outputs of map stage are shuffled as input of reduce tasks which performs a summary operation.

Usually, the outputs of map stage are a set of key/value pairs. Then the outputs are shuffled to reduce stage base on the key of each pair. The whole process of MapReduce could be summarized as following 3 steps:

- **Map:** Each worker node reads data from cluster with lowest transmit cost and applies the "map()" function to the local data, and writes the output to a temporary storage.
- **Shuffle:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- **Reduce:**Worker nodes now process each group of output data, per key, in parallel.

One good and widely used MapReduce implementation is the Hadoop [1] MapReduce [16] which consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks to achieve fault tolerance. More specifically, fault tolerance in MapReduce is provided from two aspects: task failure that can be monitored by passing heartbeat message periodically between JobTracker and TaskTracker, JobTracker failure that can be guaranteed by the approach of checking point. Checking point is an approach for applications to recover from failure by taking the snapshot of current intermediate results, resource allocation and related information to file system. When a JobTracker starts up, it looks for such data, so that it can restart work from where it left off.

[1] `http://hadoop.apache.org/`

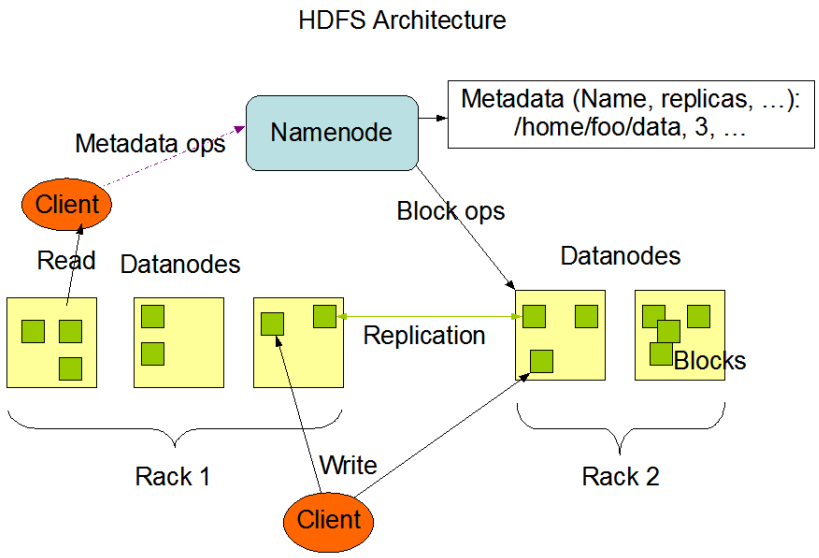

Figure 2.1: HDFS architecture [14]

In Hadoop MapReduce framework, locality is relied on Hadoop Distributed File System (HDFS) that can fairly divide input file into several splits across each worker in balance. In another word, MapReduce is built on the distributed file system and executes read/write operations through distributed file system. The next section will introduce more concrete principle and features of distributed file system by one of the most representative distributed file system, HDFS.

2.1.5 Hadoop Distribution File Systems

Hadoop Distributed File System [14] is open source project of Google File System (GFS) [18] that is deployed on computing cluster. In following text of this thesis, we will use HDFS as its abbreviation. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. [14] The assumptions and goals of HDFS includes: hardware failure, high throughput, large dataset, streaming access, data load balance and simple coherency model, "Moving Computation is Cheaper than Moving Data", and portability across heterogeneous hardware and software platforms.

An HDFS instance may consist of hundreds or thousands of nodes, each

handling part of the file system's data. Therefore, HDFS could store very large dataset and provide high throughput. Normally, the failure probability of a machine is non-trivial. But in a cluster with thousands machines, some nodes would be always non-functional. From Figure 2.1 it is clear to know that HDFS architecture is master-slaves architecture with a namenode and multi datanodes. Based on this architecture, any crashed datanode can be checked out by sending heartbeat messages periodically to namenode. Once certain datanode fail to send heartbeat message to namenode within regulated time period, namenode would mark that datanode inactive and ask other datanodes to replicate data on the failed datanode. When free space on certain servers are below the threshold specified on the configuration file, other datanodes are also asked to replicate data on overhead datanode to achieve data load balance. HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access.[14] A computation is much more efficient if it is executed in the same node or near the node where the request data located. Because transmission of data takes much more time than changing a server to do a computation. Base on this assumption, HDFS provides interfaces for applications to move themselves closer to where the data is located. In addition, one more thing in the respect of fault tolerance is referred. The principle of fault tolerance in HDFS refers to checkpoint and then HDFS recovers data from checkpoint at certain time point when certain slave server is crashed down.

In general, HDFS and MapReduce always works together. In a distribute cluster, each datanode in HDFS runs a TaskTracker as a slave node in MapReduce. MapReduce retrieves data from HDFS and executes computation and finally writes results back to HDFS.

2.1.6 Kafka

ApacheKafka is a distributed, partitioned, replicated commit log service developed by the Apache Software Foundation. It aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Before we go into architecture of Kafak, there are some basic messaging terminologies: [15]

- **Topic:** Kafka maintains feeds of messages in categories called topics.
- **Producer:** Processes that publish messages to a Kafka topic are called producers.

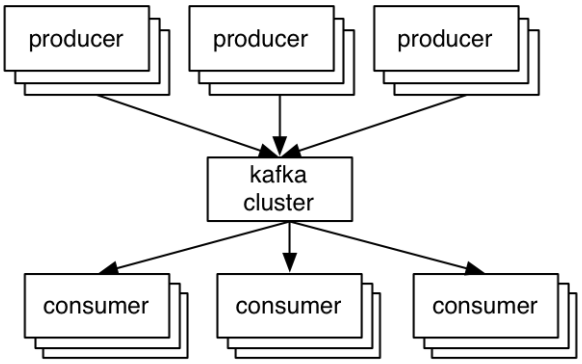

Figure 2.2: Kafka producer and consumer [15]

- **Consumer:** Processes that subscribe to topics and process the feed of published messages are consumers.
- **Broker:** Kafka is run as a cluster comprised of one or more servers each of which is called a broker.

As Figure 2.2 shown, producers send messages over the network to the Kafka cluster which holds on to these records and hands them out to consumers. More specifically, producers publish their messages to a topic, and consumers subscribe to one or more topics. Each topic could have multiple partitions that are distributed over the servers in Kafka cluster, allowing a topic to hold more data than storage capacity of any server. Each partition is replicated across a configurable number of servers for fault tolerance. Each partition is an ordered, immutable sequence of messages that is continually appended to?a commit log. The messages in the partitions are each assigned a sequential id number called the offset that uniquely identifies each message within the partition. Figure 2.3 shows a producer process appending to the logs for the two partitions, and a consumer reading from partitions sequentially.

At a high-level Kafka gives the following guarantees: [15]

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent. That is, if a message M1 is sent by the same producer as a message M2, and M1 is sent first, then M1 will have a lower offset than M2 and appear earlier in the log.
- A consumer instance sees messages in the order they are stored in the log.

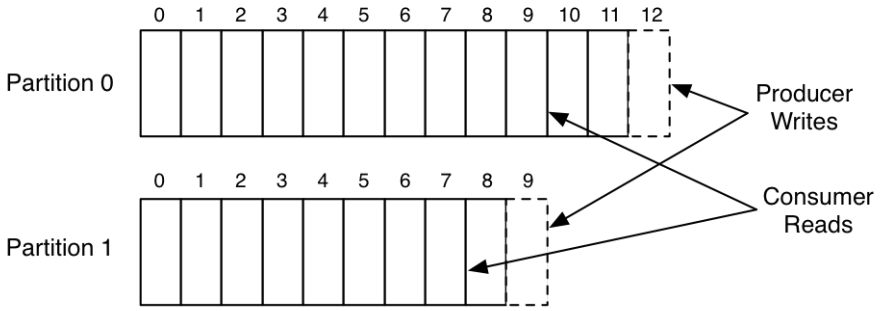

Figure 2.3: Kafka topic partitions

- For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any messages committed to the log.

Another very important feature of Kafka is messages with the same partition key will be sent to the same partition that is very important to our benchmark.

2.2 Benchmark

Batch processing technologies(Such as MapReduce, Hive, Pig) have matured and been widely used in the industry. These systems solved the issue processing big volumes of data successfully. However, first big data need to be collected and stored in a database or file system. Then it takes time to finish batch processing analysis job before get any results. While there are many cases that need analysed results from streaming data immediately. The demand for processing real time stream data is increasing a lot these days. A big data architecture contains several parts. Often, masses of structured and semi-structured historical data are stored in Hadoop (Volume + Variety). On the other side, stream processing is used for fast data requirements (Velocity + Variety)[36]. Several streaming processing systems are implemented and widely adopted, such as Apache Storm, JStorm, Apache Spark, IBM InfoSphere Streams and Apache Flink. They all support real-time stream processing, high scalability, and awesome monitoring. How to evaluate a real time stream processing system before choosing it to use in production development is a open question. Before these real time stream processing systems are implemented, Michael demonstrated the 8 requirements [34] of real-time stream processing, which gives us a standard to evaluate whether a real time

stream processing system satisfies these requirements. A very common and traditional approach to verify whether the performance of a system meets the requirements is benchmarking. Published benchmarking results from industry standard benchmark systems could help users compare products and understand features of a system easily.

2.2.1 Traditional Database Benchmarks

Traditional database management systems were evaluated with industry standard benchmarks like TPC-C, [8] TPC-H. [9] These have focused on simulating complete business computing environment where plenty of users execute business oriented ad-hoc queries that involve transactions, big table scan, join and aggregation. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. [9] The integrity of the data is verified during the process of the execution of the benchmark to check whether the DBMS corrupt the data. If the data is corrupted, the benchmark measurement is rejected entirely. [11] Benchmark systems for DBMS mature, with data and workloads simulating real common business use cases, they could evaluate performance of DBMS very well. Some other works were done related to specific business model. Linkbench [3] benchmarks database systems which store "social network" data specifically. The workload of database operations are based on Facebook's production workload and the data is also generated in such a way that key properties of the data match the production social graph data in Facebook.

2.2.2 Cloud Service Benchmarks

As the data size keep increasing, traditional database management systems could not handle some use cases with very big size data very well. To solve this issue, there are plenty of NoSql database systems developed for cloud data serving. With the widespread use of such cloud services, several benchmarks are introduced to evaluate these cloud systems.

One widely used and accepted extensible cloud serving benchmark named *Yahoo! Cloud Servicing Benchmark*(YCSB) developed by Yahoo. [6] It proposes two benchmark tiers for evaluating the performance and scalability of cloud data serving systems such as Cassandra, HBase and CouchDB. A core set of workloads are developed to evaluate different tradeoffs of cloud serving systems. Such as write/read heavy workloads to determine whether system is

write optimised or read optimised. To evaluate transaction features in later NoSql database, YCSB+T [11] extends YCSB with a specific workload for transaction called Closed Economy Workload(CEW). A validation phase is added to the workload executor to check consistency of these cloud databases. YCSB++ [32] is another set of extensions of YCSB to benchmark other five advance features of cloud databases such as bulk insertions, server-side filtering. YCSB++ could run multiple clients on different machines that coordinated with Apache ZooKeeper, which increases test ability of benchmark framework. Pokluda and Sun [33] explore the design and implementation of two representative systems and provide benchmark results using YCSB. In addition to the performance aspect of NoSql systems, they also benchmark availability and providing an analysis of the failover characteristics of each. Kuhlenkamp et al. [25] made some contributions to benchmarking scalability and elasticity of two popular cloud database systems HBase and Cassandra. The benchmark strategy is changing workloads and/or system capacity between workload runs, load and/or system capacity are changed.

These efforts are island solutions and not policed by any industry consortia. BigBench aims to be implemented as an industry standard big data benchmark [17]. It is an end to end benchmark identify business levers of big data analytics. Inherit from TPC-DS benchmark, BigBench implements the complete use-case of a realistic retail business. The data model of which covers three major characteristics described by Laney [26] of big data system: volume(larger data sizes), velocity(higher data arrive rates) and variety(increased data type disparity). The main part of the workload is the set of queries to be executed against the data model. These queries are designed along one business dimension and three technical dimensions. [17]

2.2.3 Distributed Graph Benchmarks

A specific type of big data which keeps increasing in day-to-day business is graph data. The growing scale and importance of graph data has driven the development of numerous specialized graph processing systems including Google's proprietary Pregel system [27], and Apache Giraph [1], PowerGraph[19]. In the paper, Guo et al. [21] demonstrate the diversity of graph-processing platforms and challenges to benchmark graph-processing platforms. Among these challenges, some are common for general benchmark systems, such as evaluation process, dataset selection and result reporting. To address these issues of evaluating graph-processing platforms, Guo et al. implemented a benchmark suit using an empirical performance-evaluation method which includes four stages: identifying the performance aspects and metrics of interest; defining and selecting representative datasets and algorithms; imple-

menting, configuring, and executing the tests; and analyizing the results. In order to create an industry-accepted benchmark, this method still raises some issues. In latest released papers[4, 23], the team implemented a benchmark called Graphalytics for graph-processing platforms.

2.2.4 Existing stream processing benchmarks

To help users have a better understanding of stream processing system and choose one intelligently in practical work, there are already several tests or benchmarks of stream processing systems published on the Internet. Early work by Córdova [7] focuses on analysing performance of two notable real time stream systems Spark streaming and Storm Trident. While it could not be a standard benchmark because it is not extensible and experiment cluster is too small with only 3 nodes. More ever, the data model and workloads are quite simple which could not reflect the real use cases in business. IBM compares the performance of IBM InfoSphere Streams against Apache Storm with a real-world stream processing application detect online spam. [29] The processing pipeline for the benchmark email classification system is divided into 7 stages and implemented by InfoSphere Streams and Apache Storm separately. The main drawback of this approach is there is only one scenario. First, the workload includes different steps(operations) makes it hard to detect the possible performance bottleneck. There are some other features of Streams and Storm that are not included in the application such as sort. xin [2] in her blog compared Storm and Spark Streaming side-by-side, including processing model, latency, fault tolerance and data guarantees. LinkedIn benchmarked its own real-time streaming process system Samza running four simple jobs and got excellent performance: 1.2 million messages per second on a single node [13]. Other similar works [24, 31] are all focusing on compare two or three specific real-time streaming process systems. Recently, Yahoo Storm Team demonstrated a stream processing benchmark. Design and more features of *The Yahoo Streaming Benchmark* will be introduced in detail in the next section.

2.2.5 The Yahoo Streaming Benchmark

The Yahoo Streaming Benchmark(YSB) is introduced to analysis what Storm is good at and where it needs to be improved compared to other stream processing systems by Yahoo Strom Team. [35] The benchmark is a single advertisement application to identify relevant advertisement events. There are a number of advertising campaigns, and a number of advertisements for each campaign. The application need read various JSON events from a Kafka

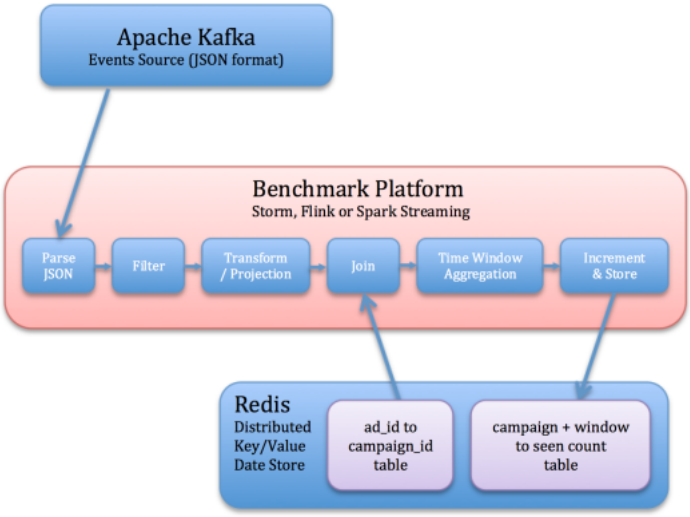

Figure 2.4: Operations flow of YSB [35]

topic, identify the relevant events, and store a windowed count of relevant events per campaign into Redis. The flow of operations could be shown as Figure 2.4.

Each event(message) in Kafka topic contains a timestamp marking the time producer created this event. Truncating this timestamp to a particular digit gives the begin-time of the time window that the event belongs in. When each window is updated in Redis, the last updated time is recored.

After each run, a utility reads windows from Redis and compares the windows? times to their last update times in Redis, yielding a latency data point. Because the last event for a window cannot have been emitted after the window closed but will be very shortly before, the difference between a window?s time and its last update time minus its duration represents the time it took for the final tuple in a window to go from Kafka to Redis through the application.

$$finalEventLatency = (lastUpdatedTime - timestamp) - duration$$

More details about how the benchmark setup and the configuration of experiment environment could be found here [2]. From experiments demonstrated in this page, Storm 0.10.0 was not able to handle throughputs above

[2] `http://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at`

135,000 events per second. The larget rate at which Kafka emitted data events into the Flink benchmark is varied 170,000 events/sec which doesn't reach throughput of Flink. From the design and conclusion of experiments, it is very obvious that the benchmark focus more on latency other than throughput of stream processing systems. Another shortage of this benchmark is one single workload could not reflect features of stream processing systems comprehensively, even the steps of benchmark flow attempt to probe common operations performed on data streams.

Chapter 3

Stream Processing Platforms

Introduce three widely used stream processing platforms, point out core concepts and key features

3.1 Apache Storm

3.1.1 Storm Architecture

3.1.2 Computing Model

3.2 Apache Flink

3.2.1 Flink Architecture

3.2.2 Memory Management

3.2.3 Flink Streaming

3.3 Apache Spark

3.3.1 Resilient Distributed Datasets(RDDs)

3.3.2 Spark Streaming

3.4 Other Stream Processing Systems

3.4.1 Amazon S4

3.4.2 Apache Samaza

3.4.3 Apache Apex

3.4.4 Aurora

Chapter 4

Benchmark Design

4.1 Architecture

4.2 Experiment Environment Setup

4.3 Data Source

4.3.1 Test Data Generation

4.3.2 Kafka

4.4 Experiment Log and Statistic

4.5 Extensibility

Chapter 5

Experiment

5.1 Experiment Environment

5.2 Classic Workload

5.2.1 WordCount

5.2.1.1 Offline WordCount

5.2.1.2 Short WordCount

5.2.1.3 Windowed WordCount

5.2.1.4 Online WordCount

5.2.2 Data Source

5.2.3 Results and Discussion

5.3 Multi-Streams Join Workload

5.3.1 Advertisements Click

5.3.2 Data Source

5.3.3 Results and Discussion

5.4 Iterate Workload

5.4.1 WordCount

5.4.2 Data Source

5.4.3 Results and Discussion

Chapter 6

Conclusions

Summary of experiment results

6.1 Selection in Practice

Summarize several factors which affect selection of stream processing systems in practic

6.1.1 Performance Summary

6.1.2 Issues

6.2 Future Work

Future works

6.2.1 Scale-out and Elasticity Evaluation

6.2.2 Evaluation of Other Platforms

Bibliography

[1] Giraph. URL http://giraph.apache.org/.

[2] Xinh's tech blog. URL http://xinhstechblog.blogspot.fi/2014/06/storm-vs-spark-streaming-side-by-side.html.

[3] Timothy G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. Linkbench: A database benchmark based on the facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1185–1196, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2037-5. doi: 10.1145/2463676.2465296. URL http://doi.acm.org/10.1145/2463676.2465296.

[4] Mihai Capota, Tim Hegeman, Alexandru Iosup, Arnau Prat-Pérez, Orri Erling, and Peter Boncz. Graphalytics: A big data benchmark for graph-processing platforms. 2015.

[5] F Chang, J Dean, S Ghemawat, WC Hsieh, DA Wallach, M Burrows, T Chandra, A Fikes, and R Gruber. Bigtable: A distributed structured data storage system. In *7th OSDI*, pages 305–314, 2006.

[6] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0036-0. doi: 10.1145/1807128.1807152. URL http://doi.acm.org/10.1145/1807128.1807152.

[7] Patricio Córdova. Analysis of real time stream processing systems considering latency.

[8] Transaction Processing Performance Council. Tpc-c benchmark specification, . URL http://www.tpc.org/tpcc/.

[9] Transaction Processing Performance Council. Tpc-h benchmark specification, . URL `http://www.tpc.org/tpch/`.

[10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] Anamika Dey, Alan Fekete, Raghunath Nambiar, and Uwe Rohm. Ycsb+t: Benchmarking web-scale transactional databases. In *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*, pages 223–230. IEEE, 2014.

[12] L Doug. Data management: Controlling data volume, velocity, and variety, 2001.

[13] Tao Feng. Benchmarking apache samza: 1.2 million messages per second on a single node. URL `https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node`.

[14] Apache Software Foundation. Hdfs, . URL `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`.

[15] Apache Software Foundation. Kafka, . URL `http://kafka.apache.org/documentation.html`.

[16] Apache Software Foundation. Mapreduce, . URL `https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html`.

[17] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1197–1208, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2037-5. doi: 10.1145/2463676.2463712. URL `http://doi.acm.org/10.1145/2463676.2463712`.

[18] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[19] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2, 2012.

[20] Yong Guo, Marcin Biczak, Ana Lucia Varbanescu, Alexandru Iosup, Claudio Martella, and Theodore L Willke. How well do graph-processing platforms perform? an empirical performance evaluation and analysis. pages 395–404, 2014.

[21] Yong Guo, Ana Lucia Varbanescu, Alexandru Iosup, Claudio Martella, and Theodore L Willke. Benchmarking graph-processing platforms: a vision. pages 289–292, 2014.

[22] Mohammad Haghighat, Saman Zonouz, and Mohamed Abdel-Mottaleb. Cloudid: Trustworthy cloud-based and cross-enterprise biometric identification. *Expert Systems with Applications*, 42(21):7905–7916, 2015.

[23] Alexandru Iosup, AL Varbanescu, M Capota, T Hegeman, Y Guo, WL Ngai, and Merijn Verstraaten. Towards benchmarking iaas and paas clouds for graph analytics. 2014.

[24] Robert Metzger Kostas Tzoumas, Stephan Ewen. High-throughput, low-latency, and exactly-once stream processing with apache flink. URL `http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/`.

[25] Jörn Kuhlenkamp, Markus Klems, and Oliver Röss. Benchmarking scalability and elasticity of distributed database systems. *Proc. VLDB Endow.*, 7(12):1219–1230, August 2014. ISSN 2150-8097. doi: 10.14778/2732977.2732995. URL `http://dx.doi.org/10.14778/2732977.2732995`.

[26] Doug Laney. 3d data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6:70, 2001.

[27] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing - "abstract". pages 6–6, 2009. doi: 10.1145/1582716.1582723. URL `http://doi.acm.org/10.1145/1582716.1582723`.

[28] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[29] Zubair Nabi, Eric Bouillet, Andrew Bainbridge, and Chris Thomas. Of streams and storms. *IBM White Paper*, 2014.

[30] Paulo Neto. Demystifying cloud computing. In *Proceeding of Doctoral Symposium on Informatics Engineering*, 2011.

[31] Manoj P. Apache-storm-vs spark-streaming. URL `http://www.ericsson.com/research-blog/data-knowledge/apache-storm-vs-spark-streaming/`.

[32] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs, and Billie Rinaldi. Ycsb++: Benchmarking and performance debugging advanced features in scalable table stores. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 9:1–9:14, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0976-9. doi: 10.1145/2038916.2038925. URL `http://doi.acm.org/10.1145/2038916.2038925`.

[33] Alexander Pokluda and Wei Sun. Benchmarking failover characteristics of large-scale data storage applications: Cassandra and voldemort.

[34] Michael Stonebraker, U?ur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34 (4):42–47, 2005.

[35] Yahoo Storm Team. Yahoo streaming benchmark. URL `http://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at`.

[36] Kai Wähner. Real-time stream processing as game changer in a big data world with hadoop and data warehouse, 2014. URL `http://www.infoq.com/articles/stream-processing-hadoop`.

Appendix A

Source Code

.1 WordCount

.2 Advertisements Click