Benchmarking Modern Distributed Streaming Platforms

Shilei Qian[1], Gang Wu[1], Jie Huang[2], Tathagata Das[3],
choice127@sjtu.edu.cn, wugang@cs.sjtu.edu.cn, jie.huang@intel.com, tdas@databricks.com
Shanghai Jiao Tong University[1], Intel APAC R&D Center[2], Databricks Inc[3]

***Abstract*—The prevalence of big data technology has generated increasing demands in large-scale streaming data processing. However, for certain tasks it is still challenging to appropriately select a platform due to the diversity of choices and the complexity of configurations. This paper focuses on benchmarking some principal streaming platforms. We achieve our goals on StreamBench, a streaming benchmark tool based on which we introduce proper modifications and extensions. We then accomplish performance comparisons among different big data platforms, including Apache Spark, Apache Storm and Apache Samza. In terms of performance criteria, we consider both computational capability and fault-tolerance ability. Finally, we give a summary on some key knobs for performance tuning as well as on hardware utilization.**

***Keywords*-distributed streaming computing; benchmark; big data; spark streaming; storm**

I. Introduction

Data stream computation has a wide range of scenarios that have been witnessed for a long time. Examples of such applications include financial applications, network monitoring, sensor networks and web log mining [1], [2], [3]. Nowadays we are experiencing a more rapidly expanding data scale because of the growing coverage of sensors, the prevalence of the mobile Internet and the soaring popularity of social media such as Twitter [4] and Facebook [5].

Although distributed stream processing systems have been under research by academia for over a decade (*e.g.* Medusa projects [6]), these efforts are merely distributed extensions to the single-node system Aurora [7]. The introduction of MapReduce [8] has revolutionized modern distributed computing systems. Owing to its scalability and fault-tolerance, MapReduce has become the basis for various data stream processing frameworks. Examples are S4 from Yahoo [9], Apache Storm [10], Apache Samza [11], Apache Spark Streaming [12] and TimeStream from Microsoft. As these systems keep sprouting and maturing, the necessity to evaluate and compare them rises. However, except StreamBench [13], which just carries out some evaluations to verify the correctness of its benchmark definition, no more evident efforts have been devoted to benchmark and compare these modern distributed stream processing frameworks.

Hence, this paper aims to take a step towards benchmarking several primary streaming computing platforms. In the first place we analyze the motivation for our benchmark work, followed by the description of some preliminaries. It is so hard to choose a proper platform and make best use of it because of the diversity of choices and the complexity of configurations for each platform. A comprehensive reference on how to choose a streaming platform is still unavailable. In this work, we accomplish this goal with the benchmark tool - StreamBench [13]. Considering the popularity and novelty of streaming platforms, we choose these three as our target: Apache Spark Streaming(both direct and receiver approach), Apache Storm(including Trident) and Apache Samza.

In the second step, we specify the benchmark analysis techniques. We introduce some modifications and extensions to accommodate StreamBench to our benchmark goals, and we also associate it with fair test configurations and reasonable parameters. For each platform, we compare the capability as well as the fault-tolerance ability, identify and tune some key knobs, and summarize the characteristics of hardware utilization.

Finally, we carry out experiments on each platform and provide further analysis. We notice that Spark direct approach and Storm Trident can saturate the network resource and guarantee larger throughput. Spark receiver approach also has higher throughput than Storm, while Storm has much shorter latency. Spark is quite fault tolerant and stable with the increase of data scale and node failure. We also find that some key knobs have a great impact on the performance of these platforms. In detail, batch size, write-ahead-logs mechanism and kryo serialization are keys to the efficiency of Spark, while Storm depends more on its ACK mechanism and the the size of pending list.

We hope this paper can provide a useful reference for users on how to properly choose a streaming platform and allocate resources for the deployment of streaming programs.

In summary, our key contributions are as follows:

1) A general description of several primary streaming platforms on capability and fault-tolerance ability.
2) Completely benchmark and analyze Apache Spark(both direct and receiver approach), Apache Storm(including Trident), and tune some key knobs which have a great influence on performance.
3) Conclude the characteristics of hardware utilization for these platforms.

The rest of this paper are organized as follows: Section II

This work is supported by National Natural Science Foundation of China(NSFC 61472241).

978-1-4673-8075-1/16/$31.00 ©2016 IEEE

describes the motivation for our work, Section III displays some preliminaries. Analysis techniques are discussed in Section IV, followed by the experiment and analysis in Section V. And we conclude this paper in Section VI.

II. Motivation

A user may always face some problems when choosing a streaming platform. We summarize them in this section, which form the motivation of our work.

The main problem is just how to select a proper streaming platform. There are many distributed streaming computing frameworks nowadays. However, reference on how to choose a streaming platform is still unavailable. StreamBench carries out some evaluations to verify the correctness of its benchmark definition, while omitting completely benchmarking and tuning work. Except that, no more evident efforts have been devoted to benchmark on these modern distributed stream processing platforms.

It is also challenging for users to know whether the platform is reliable and fault-tolerant. Apart from the capability issue, reliability and fault-tolerance ability are major criteria on selecting platforms to process real streaming data.

The lack of knowledge on factors impacting the performance of streaming applications can be a serious problem. The complexity of the configurations for each platform makes it time-consuming to reach the best performance. For a specific streaming platform, among these large numbers of parameters, there are some key parameters influencing the performance more significantly.

The last problem is the decision on the amount of hardware resources for streaming applications, including CPU, memory, network and disk. The allocations of these resources generally vary according to platforms and application requirements. How to characterize the hardware utilization for the streaming platforms is another important motivation of our work.

III. Preliminary

Before the introduction of our benchmark workload suites and the experiments, we need to present some preliminaries in this section.

A. Target Platform

Considering the popularity and novelty of streaming platforms, we choose these three as our target: Apache Spark Streaming, Apache Storm and Apache Samza. In our benchmark work, we take Spark Streaming and Storm as our primary target, and tune some key parameters for them. Particularly, for Samza, the benchmark on it is limited to the capacity issue due to the immaturity of this platform.

1) Apache Spark Streaming: Apache Spark Streaming is an extension of the core Spark API, supports the processing of live data streams. Internally, it receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches [14].

2) Apache Storm: Apache Storm is a distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, do for realtime processing. Trident is a high-level abstraction for doing realtime computing on top of Storm. It allows you to seamlessly intermix high throughput, stateful stream processing with low latency distributed querying [15].

3) Apache Samza: Apache Samza is a distributed stream processing framework. It uses Kafka for messaging and Apache Hadoop Yarn to provide fault tolerance, processor isolation, security, and resource management. The architecture of Samza makes it simple and pluggable [11].

B. Benchmark Tool

Regarding the convenience of our benchmark work, we modify and extend the definition of StreamBench to meet our needs. Some details about benchmark definition are discussed here. We try to make our test configurations and parameters fair and reasonable, even though the benchmark suite itself is not mature enough.

1) Benchmark Definition: StreamBench suggests using a message system as a mediator for data generation and consumption. In our experiments, we use Apache Kafka [16] as the message system.

Stream data are sent to target systems on the fly. To approach real-world cases, we choose to generate streams from seed data sets gathered from two main stream computing scenarios, respectively real time web log processing [17] and network traffic monitoring [18]. The data sets cover both text and numeric data.

2) Workloads: 7 benchmark programs are used in our experiments, summarized in Table I. The first four benchmarks are relatively atomic. Identity benchmark simply reads input and takes no operations on it. It intends to serve as the baseline of other benchmarks. Sample benchmark samples the input stream according to specified probability. It's a basic operation widely utilized [1], [19]. Projection benchmark extracts a certain field of the input. Grep checks if the input contains certain strings. It is a simple yet common operation also adopted in the evaluation of Spark Stream [12]. The last three benchmarks are more complex. Wordcount benchmark first splits the words, then aggregates the total count of each word and updates an in-memory map with word as the key and current total count as value. It is widely accepted as a standard micro-benchmark for big data [20]. DistinctCount benchmark first extracts target field from the record, puts it in a set containing all the words seen and outputs the size of the set which is the current distinct count.

Table I: Workload Categorization Mapping

Benchmark	Complexity	Input data seed
Identity Sample Projection Grep	Low - Staless	AOL [17]
Wordcount DistinctCount	Medium - Stateful	
Statistics		CAIDA [18]

Statistics benchmark calculates the maximum, minimum and sum of a field from input. Except Statistics using the numeric data with the record size of 200B, the other six workloads deal with the text source with the size of 60B for each record.

3) Modifications on StreamBench: StreamBench is defined for general platforms and supports several kinds of workloads. We need to choose a proper configuration, and introduce some modifications and extensions to accommodate it to our benchmark goals.

We choose two clusters, one for Kafka and one for streaming system. There are two kinds of deployments for Kafka integration, one is the Kafka and streaming system both deployed on same nodes, the other is what we apply, just forming two clusters. If we choose the previous one, the streaming system cannot fully occupy the resources of the node. The metrics of its behavior will also be confusing due to the influence of Kafka.

For the data preparation method, we treat offline data preparation as a better way. We need to ensure that there is no overhead from disks of Kafka. In our experiment, the scale of data does not exceed the capacity of disk cache, and full network utilization can be reached. Offline data preparation means that the generation of streaming data is done at first, and then fetch and process steps are taken. This can keep the impact of message system at a low level, and facilitate our benchmark process and validation.

The validation on the results of workloads for different platforms is necessary. Just as previous statement, the offline data preparation makes the validation work feasible. If we obtain the same result for workloads running on different platforms, we can guarantee the semantics of the workload implementations on different platforms are identical, and our work is reasonable.

Since the nodes we use in our experiments are much more powerful than those in StreamBench, we increase the data scale rather than the original definition. The partition number for the input streams is customized as well. Flow control for some stream systems is added to make them accommodate to the speed of the stream processing.

IV. Analysis Technique

In our benchmark work, we need to set some work goals and define some workload suites, as well as some metrics for them. The details of analysis techniques are displayed in this section.

A. Workload Suite

1) Capability Workload Suite: The capability workload suite is designed to test throughput, latency and capacity. It incorporates all of the seven benchmark programs, demands all input data be pushed to the message system where stream computing systems obtain data as fast as they can.

2) Fault-tolerance Workload Suite: This suite uses Identity workload and considers only one node failure. It requires intentionally failing one node in about the middle of the executions to capture penalty brought by the loss of the computing resource. For feasibility, we take a straightforward approach by simply collecting the performance metrics through the whole process and comparing it with failure free executions. We simply consider half of the run time of the failure free experiment done in the previous suite as the expected middle of this suite.

B. Key Knobs for Certain Platforms

The complexity of the configurations for each platform always takes users a lot of time getting the best performance. For a specific streaming platform, among these large numbers of parameters, there are some key parameters influencing the performance much more significantly but users are not aware of. We need to find these key knobs and do some tuning experiments.

C. Characteristic of Hardware

The utilization of hardware resources (*e.g.* CPU, memory, network, disk) varies significantly with different workloads and platforms. We aim to conclude a general description on the characteristics of hardware utilization for each streaming platform, based on the monitoring data of hardware during the experiments.

D. Metrics

Metric definition is correlated with workload suite. For performance related work suite, two main metrics, throughput and latency, are proposed to describe the capability of the platforms. Throughput metric is the data size in terms of bytes processed per second. The second metric is latency, which is the average time span from the arrival of a record till the end of processing this record.

New metrics are raised for the fault tolerance workload suite. Suppose originally N nodes are serving, as described in the workload suite definition, one node is deliberately failed approximately in the middle of execution, remaining N-1 nodes working. Denote the throughput for N nodes without failure to be T_N, throughput for N-1 nodes without failure to be T_{N-1} and the throughput of the workload with

Table II: Hardware Configuration Summary

Hardware	Configuration
CPU	Xeon E5-2699@2.30GHz 72 cores
Memory	128 G
Disk	12 SATA HDDs 1T
NIC	10 Gb

Table III: Software Version Summary

Software	Version
OS/Kernel	Ubuntu 14.04.2 LTS 3.16.0-30-generic x86_64
JDK	Oracle jdk1.8.0_25
Hadoop	Hadoop-2.3.0-cdh5.1.3
Storm	0.9.3
Spark	1.4.0
Samza	0.8.0
Kafka	Kafka_2.10-0.8.1

failure is $T_{failure}$. We define the throughput penalty factor TPF to be:

$$TPF = \frac{T_{failure}}{\frac{1}{2}(T_N + T_{N-1})}.$$

It's easy to tell that TPF is usually less than 1 and the closer to 1 the better. Similarly latency penalty factor LPF is:

$$LPF = \frac{L_{failure}}{\frac{1}{2}(L_N + L_{N-1})}.$$

where L stands for latency. LPF is likely to be greater than 1 and the closer to 1 the better.

To achieve the goal of characteristics, we need to monitor the usage of hardware resource during the computation.

V. Experiment and Analysis

This section briefly dips into the configurations of the cluster followed by the result and analysis of these platforms in terms of goals in previous sections. For Apache Spark Streaming, both receiver-based approach and direct approach introduced since Spark 1.3 are evaluated. And for Apache Storm, result of Trident is also carried out.

A. Experiment Hardware

6 identical nodes are used forming two clusters: 3-node Kafka cluster, running as input source and 3-node streaming system, doing stream computation. The hardware configurations of these nodes are given in Table II. The maximum ethernet link speed of these nodes can reach 10000Mb/s. Hyper threading is enabled for all tests, while hugepage is always disabled. One extra node is needed to deploy master and Apache Zookeeper providing services for Apache Kafka.

B. Cluster Configuration

All software versions can be found in Table III. For Spark, we set executor memory size to 100G, with 20 seconds batch duration. For Storm, 4 workers are set up in each node, each with 32G max memory allocation. And for Samza, it needs the Apache Hadoop YARN environment, where we use Hadoop-2.3.0-cdh5.1.3. We don't make any in-depth tuning and just make adequate configurational adjustments according to system hardware resources, trying to make full utilization of them.

C. Workload Suite Experiment

This section presents the result for the workload suites defined previously on these platforms.

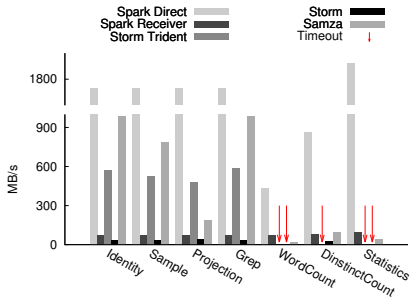
1) Capability Workload Suite: The capability workload suite is designed to test the throughput, latency and capacity.

Figure 1(a) shows the throughput for platforms. The batch interval here we choose for Spark is 20 seconds, and 500MB for the Storm Trident batch size. Input data scale is defined, namely 1 billion records for record size of 60B and 0.5 billion for 200B.

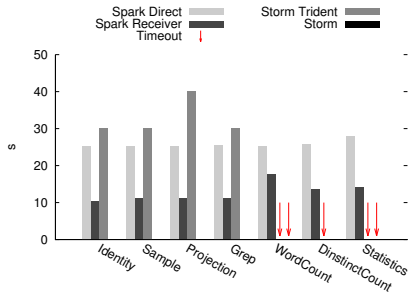
Here come our observations. Observation 1 to 4 are based on workload Identity. **[Observation 1]** To the Spark platform, the throughput for direct mode is the best. It can reach the network upper bound. **[Observation 2]** Storm Trident can also saturate the network and gain a good throughput performance due to the batch work. **[Observation 3]** The throughput of Storm is the worst in these platforms with ACK mechanism on. **[Observation 4]** For Samza, its throughput is much better than Spark receiver approach.

Following observations are about the throughput performance on different workloads for a certain platform. **[Observation 5]** The throughput of Spark direct mode varies with benchmark programs, and the maximum (Identity) throughput is close to 4 times that of the minimum (WordCount). However, the throughput for Spark receiver mode is quite similar among different workloads. The reason is that WordCount in direct mode needs more CPU resource, and cannot saturate the network, while the network utilization for receiver mode is quite similar for different workloads. **[Observation 6]** The throughput for Storm Trident also varies a lot, because different implementations of workloads ask for different allocation of resources. **[Observation 7]** For Storm, its throughput metric is almost constant to various benchmark programs.

In Figure 1(b), the latency for platforms is displayed. The latency for Storm is very short, around 100 milliseconds, so that it can hardly seen in the figure. And the latency for Samza haven't been collected. Two observations come out regarding latency. **[Observation 1]** The latency for Spark direct approach is about twice that for receiver approach, except WordCount. **[Observation 2]** The weakness of Storm Trident is its long latency. However, the latency for Storm is the shortest among these platforms. If we decrease the batch size of Spark and Storm Trident, we can gain better latency, but still no better than Storm.



(a) Throughput for platforms



(b) Latency for platforms

Figure 1: Performance overview for platforms

Table IV: Storm result of WordCount in small scale

Scale (million)	Storm Throughput(MB/s)	Storm Latency(s)	Storm Trident Throughput(MB/s)	Storm Trident Latency(s)
5	10.3	0.31	103.0	3
50	9.7	0.29	154.5	20
500	Timeout	Timeout	Timeout	Timeout

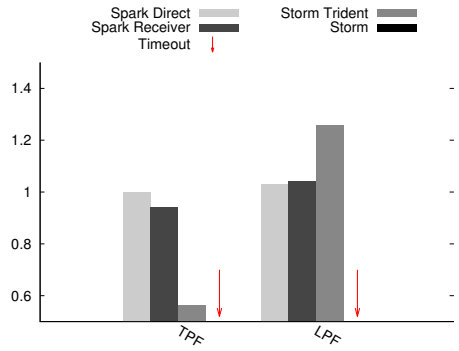


Figure 2: Fault tolerance evaluation for platforms

For the capacity, Spark and Samza can handle larger amount of data than Storm. Take the WordCount workload into acount, it is so complex that Storm as well as Storm Trident cannot handle this workload at 1 billion data scale. This is mainly because high CPU occupation increases the possibility of message failure and resending, resulting in the workload timeout. We execute it at some smaller scales. The result can been found in Table IV.

In conclusion for this capability workload suite, we find that Spark direct approach and Storm Trident can saturate the network resource and have a larger throughput. Spark receiver approach also has higher throughput than Storm. But Storm has much shorter latency. The capacity for Spark is better than Storm.

2) Fault-tolerance Workload Suite: This workload suite focuses on the fault tolerance evaluation for platforms. As you can see in Figure 2, we evaluate Spark Streaming direct mode, receiver mode, Storm and Storm Trident, both TPF (Throughput Penalty Factor) and LPF (Latency Penalty Factor) defined in the metrics section are obtained and calculated.

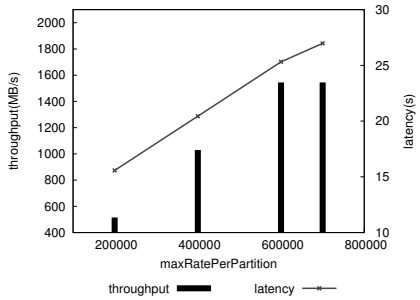
TPF is usually less than 1 and the closer to 1 the better. From Figure 2, the TPF for Spark, both for direct approach and receiver approach, is quite close to 1, which means the failure doesn't impose any significant penalty to throughput in Spark. In contrast, TPF for Storm Trident is below 0.6, suggesting over a drop of one third for throughput is witnessed. LPF, which is an index related to latency penalty, is usually larger than 1 and also the closer to 1 the better. It is evident that Spark's LPF is really close to 1 in both two approaches, which indicates that just TPF shows, failure doesn't impact Spark platform as much. However, the LPF for Storm Trident is quite high, which means a great impact on latency has been caused by the failure.

We choose the same record size and data scale for all fault tolerance tests, but 2 nodes seem unable to handle that amount of scale for Storm. That's why we mark it timeout for the Storm's TPF and LTP. We can infer that, for Storm and Storm Trident, due to errors occurring in some segments of topology, the related data has to be reprocessed; moreover, resources for services providing are declined, putting more loads on the existing worker nodes, so we see a big performance loss.

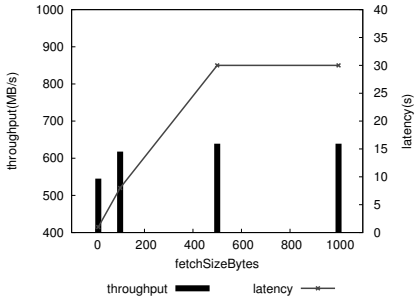
D. Key Knobs for Certain Platforms

There are some key parameters having a significant impact on the performance of each platform. The tuning result for them is present here.

The first parameter we consider is the batch size for Spark direct approach and Storm Trident. The throughput and latency for Spark direct approach at different batch size are displayed in Figure 3(a). The parameter *maxRatePartition* defines the records number per second per partition. When it changes from 200000 to 600000, the throughput becomes



(a) Spark direct approach



(b) Storm Trident

Figure 3: Batch size evaluation for Spark direct approach and Storm Trident

larger while the latency becomes longer. After 600000, the stream process comes into an unhealthy status, which means the data in a batch can not be handled in one batch interval time. Figure 3(b) shows the result for Trident. The parameter *fetchSizeBytes* defines the data scale per partition for a batch, and both throughput and latency become convergency with the increase of this parameter.

Spark Streaming has a lot of features, we tune some key parameters in our experiments, that is the batch interval, the write-ahead-logs feature and the kryo serializer. Here come our observations. **[Observation 1]** We mainly choose 20s to be the batch interval of Spark, but also experiment for 1s. The latency is obviously lower in 1 second interval. As for the throughput, in direct approach, it is quite similar in one step workload and Statistics, but worse in WordCount and DistinctCount, especially for WordCount, about ten times difference. And the result for receiver approach is approximately similar to direct approach. Since short batch interval cannot benefit much from batch work. **[Observation 2]** Write-ahead-logs mechanism is created to achieve strong fault-tolerance guarantees since Spark 1.2. When WAL option is enabled, the default 2 copies will be reduced to 1, thus greatly reduce network and disk overheads, but HDFS are replicated to 3 copies. According to our result, it doesn't effect throughput much, but increases the latency by about 12%. **[Observation 3]** Kryo serialization has always been recommended. Compared to the default java serialization, 33% throughput performance improvement has been observed, as well as 26% latency decrease, in receiver approach. While in direct approach, both throughput and latency score are steady due to the full utilization of network.

The ACK mechanism and the size of pending list for Storm are quite important to its performance and fault tolerance ability. **[Observation 1]** The ACK option is enabled to acknowledge message if it has been successfully processed. But it can be turned off to gain better performance, about 8 times increase in throughput in our experiment. **[Observation 2]** The size of pending list we mainly choose in Storm is 1000, however, when we increase it to 10000, 50% throughput increase has been observed, but the latency is about 5.6 times longer.

Table V: System resource maximal utilization for platforms

Platform	CPU	Network	Memory
Spark Streaming Direct	10%	100%	40%
Spark Streaming Receiver	10%	13%	32%
Storm Trident	60-80%	100%	60%
Storm	40%	7%	40%
Samza	60-80%	40%	20%

The number of partitions for Kafka for all previous experiments is 48, 4 partitions per broker and 4 brokers per node in Kafka configuration. We try to increase this to 162 (9 partitions per broker and 6 broker per node) to see if there is any performance improvement. We test the WordCount workload on Spark direct approach. The result shows 69% throughput improvement with 33% latency increase. This is because Spark Streaming is not fully loaded with the default concurrency number on this workload. However, the experiments on Storm and Storm Trident are still out of time, due to the large data scale beyond their handling abilities.

E. Characteristic of Hardware

System resource maximal utilization is given in Table V. Both Spark direct approach and Storm Trident can saturate the network resource. This is the main cause of the high throughput for these two platforms. However, Storm Trident is poorly pipelined in receiving and computing. It has a notable period of network bandwidth full-load before each mini-batch starts, which is caused by data sending and receiving. Following that period, Trident starts processing every batch data and consequently has a relatively high CPU load. In contrast, Spark Streaming realizes good pipeline operations in network utilization and computing during this process, which leads to better utilization of resources. Therefore, the throughput performance of Trident is worse

than Spark direct approach.

However, in Spark receiver mode, the mechanism of serializer and deserializer makes it hard to fetch the data as fast as Spark direct mode. The CPU load is just as low as in direct mode.

Storm has higher CPU utilization even in single step workloads. Besides, with the increase of data size, Storm and Storm Trident have a high rate of data resending errors especially on computation intensive applications, due to high CPU occupation. Whereas, the network utilization for storm is just at a low level.

VI. Conclusion

In this paper, we benchmark several primary streaming platforms. We first present the motivation of our work: to deal with the diversity of streaming computing platforms and the complexity of the configurations, as well as the lack of reference. We then present some preliminaries. We target on Apache Spark Streaming, Apache Storm(including Trident) and Apache Samza, and use StreamBench as our benchmark tool, with some modifications and extensions. Afterwards, some analysis techniques are defined, followed by the real experiments and analyses. For benchmark criteria, we not only consider capability, but also fault-tolerance ability. A general evaluation for different streaming platforms, some key knobs influencing performance tuning, as well as the characteristics of hardware utilization for them have been carried out.

In summary, we find that Spark direct approach and Storm Trident can saturate the network resource and have larger throughput, especially for Spark direct approach due to the well pipelined operations. Spark receiver approach also has higher throughput than Storm. But Storm has much shorter latency. Spark is quite fault tolerant and stable with the increase of data scale and node failure. Besides, some key knobs really have a great impact on the performance of these platforms.

We hope this paper can be a reference for a user who need to choose a streaming platform and deploy some streaming computation programs. For future work, complete benchmark on Apache Samza need to be carried out. And more test cases can be included, such as more kinds of hardware resources, more tuning parameters, etc.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2002, pp. 1–16.

[2] L. Golab and M. T. Özsu, "Issues in data stream management," *ACM Sigmod Record*, vol. 32, no. 2, pp. 5–14, 2003.

[3] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.

[4] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time url spam filtering service," in *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 447–462.

[5] Z. Shao, "Real-time analytics at facebook," in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2011, pp. http://www–conf.slac.stanford.edu.

[6] U. Cetintemel, "The aurora and medusa projects," *Data Engineering*, vol. 51, no. 3, 2003.

[7] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," vol. 12, no. 2, pp. 120–139, 2003.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[9] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010, pp. 170–177.

[10] "Storm: Distributed and fault-tolerant realtime computation." http://http://storm.apache.org/.

[11] "Samza." http://samza.apache.org/.

[12] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 2012, pp. 10–10.

[13] R. Lu, G. Wu, B. Xie, and J. Hu, "Streambench: Towards benchmarking modern distributed stream computing frameworks," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*. IEEE, 2014, pp. 69–78.

[14] "Spark Streaming Programming Guide." http://spark.apache.org/docs/1.4.0/streaming-programming-guide.html.

[15] "Trident Tutorial," http://storm.apache.org/documentation/Trident-tutorial.html.

[16] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011.

[17] "AOL Search Data." http://www.researchpipeline.com/mediawiki/index.php?title=AOL_Search_Query_Logs.

[18] CAIDA, "The CAIDA UCSD Statistical information for the CAIDA Anonymized Internet Traces," http://www.caida.org/data/passive/passive_trace_statistics.xml.

[19] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2012.

[20] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 41–51.