Benchmark Suite to Evaluate Performance of Distributed Stream Processing Platforms

Milinda Pathirage
School of Informatics and Computing
Indiana University
mpathira@indiana.edu

Beth Plale
School of Informatics and
Indiana University
plale@cs.indiana.edu

ABSTRACT

Interest on continuous queries on streams of data has increased over last couple of years due to the need of derving actionable information as soon as possible to be competitive in the fast moving world. As a result of the limitations in batch processing technologies from previous generation, distributed stream processing systems like Yahoo's *S4*, Twitter's *Storm*, *Spark Streaming* were introduced into the fast growing Big Data eco-system. Even though there are various different stream processing platforms and frameworks on top of them with different capabilities and characteristics, in-depth comparative studies of performance, scalability and reliability has never been done. Users of these system often face difficulties when choosing a system as a solution to a task at hand. In this paper we use some of the popular distributed stream processing use cases we identified by going through mailing list discussions, presentations and publication to evaluate three (*Storm, Spark Streaming, Samza*) of the most popular distributed processing systems used widely today.

1. INTRODUCTION

Internet has become a central part of our daily lives and recent developments in *Smart Phones*, *Internet of Things* and *Smart Homes* related technologies are going to make us connected through Internet more than ever. As more and more people and devices are connected together via web applications which Internet is built, these application will produce massive amount of data at rates which we haven't experienced yet. In parallel to the increase of data generation, business entities and users require this data and information extracted or created out of this data available to them as fast as possible and trending towards decreasing latency for ever.

As we have experienced in recent past, traditional relational database technologies couldn't cope up with these ever increasing data rates, volumes and ever decreasing latency. So technology giants like Google, Yahoo and Amazon invented novel storage technologies like GFS [10], HDFS, Big Table [6] and Amazon Dynamo [8] and distributed data processing frameworks such as Map-Reduce [7]. As the trend continues, these systems which follow traditional pull based approach to answering queries couldn't handle the latency requirements and throughput requirements, *Distributed Stream Processing* platforms were built on top of modern messaging infrastructures to fulfill these requirements. While most of these new distributed stream processing frameworks are inspired by Map-Reduce and related big data technologies, most of the underlying concepts were from previous work done by both academia [1] [2] [15] [12] [3] [14] and industry [9] [11].

With this trend of moving to push based streaming processing platforms for reducing the latency and to support increasing throughput requirements, several distributed and single-node stream processing platforms were implemented and they are continue to evolve to tackle ever changing requirements. But due to different architectural decisions underlying these platforms and variations in problem classes and non-functional requirements these frameworks are trying to address, our experiments shows that these frameworks behave differently under different types of data and algorithms. In addition to that these systems support different types of programming abstractions.

These differences make the decision of selecting a specific platform hard. There is a stream data management benchmark [4] defined almost a decade ago and there are existing results on how these platforms perform individually [16]. But its hard to find comparative studies and existing benchmark application [4] doesn't cover all aspects of modern distributed stream processing systems. In this work, we present a comprehensive evaluation of several popular distributed stream processing platforms and defines evaluation process which addresses multiple performance aspects such as *throughput*, *latency*, *scalability*, *resource utilization*, *fault tolerance* and *behavior under increasing load* across *type of streaming data* and *stream processing algorithm* dimensions. We believe that the *process*, *metrics*, *data sets* and *applications* described here make a comprehensive stream processing benchmark suite which can use to compare different distributed stream processing engines.

We implement this benchmarking suite on three popular distributed stream processing platforms - Apache Storm, Apache Samza and Apache Spark Streaming. This demonstrates the applicability of our benchmarking approach to compare and contrast distributed stream processing platforms.Our main contributions are:

- We propose number of micro-benchmarks which cover fundamental building blocks of stream processing applications such as filtering, enrichment, aggregation, join, counting, pattern detection and windowing.
- We present first comprehensive evaluation of several distributed stream processing platforms, along several dimensions including types of streaming data, stream processing applications, scalability and fault tolerance.

Type of Streaming Data - Event stream processing has uses in various different fields like Internet of Things, real-time analytics, fraud detection and trading. Each of these applications generate various types of data streams from time series data to streaming graphs. Then dimensionality of a tuple in these streams can varies from one to more than hundred (**Need references**).

Stream Processing Tasks - According to [5], we can categorize stream processing tasks on two dimensions, complexity of task and latency requirements of task. For example counting, averages and count distinct can be very simple where as fraud detection, outlier detection or predictions based on streams of data can be complex. On the other hand reporting and monitoring applications may not have strong latency requirements. Even 1 minute delay would be fine in most cases. But applications which controls some machine or device based on streaming data may require sub millisecond or sub second latency requirements.

2. BENCHMARKING FRAMEWORK AND METHODOLOGY

2.1 Benchmarks

**TODO* - Find streaming algorithms and references to their applications. The question is can we only use queries defined in [13]

- Keen.io like ingest use case
- Twitter Top-K hash tags
- Its important to measure how each of these systems responds to the variations in data rate.
- Its important to measure the performance metrics when multiple continuous computations are running.

2.1.1 Oracle Complex Event Processing Design Patterns

- Event Filtering
- New Event Detection
- Event Partitioning
- Event Enrichment
- Event Aggregation
- Event Correlation
- Application-time Events
- Missing Event Detection

3. EXPERIMENT SETUP

In this section we describe the implementation of benchmarking framework we defined in Section 2. Rest of the section describes how stream processing platforms are selected, hardware and software configurations, stream processing algorithm related configurations and details on tuning specific stream processing platform.

Across all the stream processing platforms, we use Kafka as the data stream source and all the incoming messages are thrift encoded except for cases we have specifically mentioned. *If we need to convince the use of Kafka, this is a good resource.

4. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al. The design of the borealis stream processing engine. In *CIDR*, volume 5, pages 277–289, 2005.

[2] 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. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, 12(2):120–139, 2003.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. Stream: the stanford stream data manager (demonstration description). In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 665–665. ACM, 2003.

[4] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: a stream data management benchmark. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 480–491. VLDB Endowment, 2004.

[5] M. L. Braun. Streamdrill: Analyzing big data streams in realtime, May 2014.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[7] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[9] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. Spade: the system s declarative stream processing engine. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1123–1134. ACM, 2008.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[11] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)*, 46(4):46, 2014.

[12] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik. High-availability algorithms for distributed stream processing. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 779–790. IEEE, 2005.

[13] M. R. Mendes, P. Bizarro, and P. Marques. A performance study of event processing systems. In *Performance Evaluation and Benchmarking*, pages 221–236. Springer, 2009.

[14] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. CIDR, 2003.

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

[16] 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*, pages 10–10. USENIX Association, 2012.