STREAM-PROCESSING-AS-A-SERVICE (SPaaS) FOR THE CLOUD

Part – II

SWE 561 Cloud Computing

Submitted by,

Sumitha P K

92065

Table of Contents

1 INTRODUCTION 2

2 CURRENT LEADING APPROACHES 3

2.1 SPARK 3
2.2 APACHE STORM 3
2.3 SAMZA 4
2.4 AMAZON'S KINESIS 4
2.5 COMMERCIAL ENTERPRISE STREAMING 4
2.6 TRADITIONAL HADOOP APPROACH 4

3 CURRENT RESEARCH APPROACHES 4

3.1 TECHNIQUES TO LOWER THE TAIL LATENCY IN STREAM PROCESSING SYSTEMS 5
3.1.1 Adaptive Timeout Strategy 5
3.1.2 Improved Concurrency For Worker Process 6
3.1.3 Latency-based Load Balancing 6
3.2 OPERATOR-AWARE APPROACH FOR BOOSTING PERFORMANCE 6
3.3 PROACTIVE SCALING USING WORKLOAD MODELLING 7
3.3.1 Predicting incoming workload 7
3.3.2 Performance modeling of DSPS work flows 7
3.3.3 Scheduling and scaling decisions 8
3.4 ENABLING ELASTIC STREAM PROCESSING IN SHARED CLUSTERS 8

4 RESEARCH ANALYSIS AND PROPOSALS 9

5 SUMMARY 11

6 REFERENCES 12

1 Introduction

A stream is defined as a possibly infinite sequence of elements. The data type of the elements depends on the service which generates the stream. Examples of services whose output can be turned into a stream are databases, where a result set consists of several tuples, or a program whose standard output can be split into several lines or records.

Stream processing as a Service uses streaming data, like data coming from sensors in commercial buildings or manufacturing. Applications for stream processing include marketing campaigns where you are looking for real-time analytics, online surveys or product reviews, and cyber security data analytics. Stock exchanges, sensor networks and other publish/subscribe systems need to deal with high-volume streams of real-time data. Especially financial data has to be processed with low latency in order to cater for high-frequency trading algorithms.

Current distributed technologies, such as Hadoop, have been used to address these use-cases, however they are often not appropriate for use-cases involving non-deterministically available data, or dynamic real-time data. There are currently numerous real-time data processing platforms that are in development and in production use, both in industry and academia.

2 Current leading approaches

Stream processing is a programming model that makes parallel processing hardware and software easier to use. The "stream" refers to a data sequence. In stream processing, a series of operations called kernel functions apply to each element in the stream. Uniform streaming generally uses one kernel function that applies to all the elements in the stream. In the 1980s, developers created Streams and Iteration in Single Assignment Language (SISAL) which is an example of stream processing used in dataflow programming. Stream processing is a data-driven model that proves very useful in processing image, video and digital signals. It proved less useful in processing random data access such as we find in databases.[1]

Current leading approaches in Stream Processing are as follows.

2.1 Spark

Spark is Apache's open-source data processing framework that uses in-memory clusters and is extremely fast. Spark runs in-memory and graph processing, as well as stream processing. Businesses like Groupon and Yahoo use it. It can function stand-alone or on top of Hadoop.

Spark is an example of a stream processing system that is faster than Hadoop because of the way that Spark processes data. Spark completes data analysis in memory and in pretty close to real-time. When you are talking billions of bits of data, speed can make the crucial difference. Spark beats Hadoop on batch processing by ten times and is 100 times faster in data analytics.

Hadoop distributes massive collections of data through multiple intersections within groups of servers. The servers are cheaper, often consider disposable servers, so you do not have to buy expensive equipment. Hadoop keeps track of that big data which enables faster analytics. Though Spark and Hadoop are viewed as competitors in the big-data space, but the growing consensus is that they're better together.

2.2 Apache Storm

Apache Storm is a real-time computation system useful for real-time analytics, machine learning, and continuous computation. Apache says it will do for stream processing what Hadoop did for batch processing. Apache developed Storm for use with any programming language. Storm runs on top of Hadoop YARN and businesses use it with Flume for data storage. Examples of businesses using Storm: WebMD, Spotify, and Yelp.

2.3 Samza

Samza is another Apache distributed stream processing framework based on Kafka and YARN. It works similarly to MapReduce and includes snapshot management.

2.4 Amazon's Kinesis

Amazon's Kinesis provides real-time stream processing in the cloud. It connects to Amazon's other services and results in a big data architecture. Businesses use Kinesis for dashboards, alerts, and pricing.

2.5 Commercial Enterprise Streaming

Big firms like IBM, Microsoft, and Informatica all have their own versions of stream processing technology.

2.6 Traditional Hadoop approach

Hadoop relies heavily on its distributed file system to provide the data to mappers and reducers. Streaming applications could use the ZooKeeper tools to control the distributed file system and monitor changes. Use of the distributed file system would incur a significant overhead in processing and prevent us from being able to give any sort of latency guarantee. The distributed file system must be eliminated from the MapReduce cycle in order to accomodate streaming queries. As a batch processing system, Hadoop has a few bottlenecks that might cause problems when we execute our streaming query.[3]

3 Current Research Approaches

"Traditional" methods of processing big data, involving MapReduce jobs to batch process data, are not completely suitable for real-time use-cases involving processing data with non-deterministic availability. Real-time data processing, enabled by DSPS (Distributed Stream Processing Systems) technologies, allows data to be processed as soon as it is made available, without the need of a storage system, such as HDFS. A 2013 industry survey on European company use of big data technology shows that over 70% of responders show a need for real-time processing.[4]

One very notable DSPS technology developed independently of Hadoop, and that is gaining immense popularity and growth in its user base, is the *Storm* project. *Spark* is another popular big data distributed processing framework, offering of both real-time data processing and more traditional batch mode processing, running on top of

Hadoop YARN. Spark is notable for its novel approach to in-memory computation, through Spark's main data abstraction, the resilient distributed dataset (RDD). *Samza* is a relatively new real-time big data processing framework originally developed in-house at LinkedIn, which has since been open-sourced at the Apache Software Foundation. Samza offers much similar functionality to that of Storm. While Samza is lacking in maturity and adoption rates, as compared to projects such as Storm, it is built on mature components, such as YARN and Kafka, which many core features are offloaded to.
The following sections highlight some of the latest researches in the field of Stream Processing as a Service.

3.1 Techniques to lower the tail latency in stream processing systems

Over the past decade, the demand for real time processing of huge amount of streaming data has emerged and grown rapidly. Apache Storm, Apache Flink, Samza and many other stream processing frameworks have been proposed and implemented to meet this need. Although lots of effort has been made to reduce the average latency of stream processing systems, how to shorten their tail latency has received little attention.
The thesis "New techniques to lower the tail latency in stream processing systems", by guangxiang du presents a series of novel techniques for reducing the tail latency in stream processing systems like Apache Storm. These techniques have been implemented in Apache Storm, and present experimental results using sets of micro-benchmarks as well as two topologies from Yahoo! Inc. The results show improvement in tail latency in the range of 2%-72.9%. The three techniques described in the thesis are as follows.

3.1.1 Adaptive Timeout Strategy

Adaptive Timeout Strategy uses adaptive timeout coupled with selective replay to catch straggler tuples.
The thesis proposes to use an adaptive timeout strategy to selectively replay tuples that have not been fully processed within the timeout. This technique continuously collects the statistics of tuple latency, and periodically adjusts the timeout value based on latency distribution of recent issued tuples. Intuitively, how aggressively (or not) to set the timeout value is decided based on how long the tail has been in the last adjustment period.

3.1.2 Improved Concurrency For Worker Process

Shared queues among different tasks of the same operator are used to reduce overall queueing delay.
By default in today's systems (Storm, Flink) each task has an independent queue to buffer incoming tuples. The second technique to improve tail latency applies when a worker process contains more than one task from the same operator (and in the same topology). Then the latency can be improved by merging the input queues for these tasks. A task, whenever free, then opportunistically grabs the next available tuple from the shared input queue.

3.1.3 Latency-based Load Balancing

Many stream processing systems run in heterogeneous conditions, e.g., the machines (or VMs) may be heterogeneous, the task assignment may be heterogeneous (machines have different number of tasks), etc. In these scenarios, some tasks may be faster than other tasks within the same operator. Partitioning the incoming stream of tuples uniformly across tasks thus exacerbates the tail latency.
The key idea is to collect statistics only periodically, and to pair up slow and fast tasks allowing the fast part of the pair to steal work. The key features of this technique are periodic statistics collection and smooth load adjustment among tasks to suppress load oscillation.

3.2 Operator-aware approach for boosting performance

To enable efficiency in stream processing, the evaluation of a query is usually performed over bounded parts of (potentially) unbounded streams, i.e., processing windows "slide" over the streams. To avoid inefficient re-evaluations of already evaluated parts of a stream in respect to a query, incremental evaluation strategies are applied, i.e., the query results are obtained incrementally from the result set of the preceding processing state without having to re-evaluate all input buffers. This method is highly efficient but it comes at the cost of having to maintain processing state, which is not trivial, and may defeat performance advantages of the incremental evaluation strategy. In the context of RDF streams, the problem is further aggravated by the hard-to-predict evolution of the structure of RDF graphs over time and the application of sub-optimal implementation approaches, e.g., using

relational technologies for storing data and processing states which incur significant performance drawbacks for graph-based query patterns. To address these performance problems, a set of novel operator-aware data structures coupled with incremental evaluation algorithms which outperform the counterparts of relational stream processing systems is proposed[6].

3.3 Proactive scaling using Workload Modelling

In recent years there has been significant development in the area of distributed stream processing systems (DSPS) such as Apache Storm, Spark, and Flink. These systems allow complex queries on streaming data to be distributed across multiple worker nodes in a cluster. The aim of this doctoral work[7] is to investigate the viability of a proactive scaling system for DSPS. This system will use workload modelling and prediction to optimally allocate resources whilst ensuring compliance with a latencyor throughput-based SLA. Initially, the system will be targeted at Apache Storm but eventually it is anticipated that it will be applicable to other popular DSPS.

3.3.1 Predicting incoming workload

In order to enable proactive scaling, a key feature of the system is the prediction of incoming workload to the DSPS. This involves the application of well-established time series analysis techniques, as well as previous researches into work load modelling for multi-tier web architectures. Once trained on the incoming data, the workload models are used to predict sufficiently far ahead in time, that the response of the system can be estimated and, if needed, reconfigure the system before the higher workload arrives. This should allow the SLA(s) to be maintained through periods of high workload. Similarly, if the workload models are predicting a significant reduction in incoming workload, then the effect of releasing resources (whole worker nodes or worker processes for a particular work flow) can be assessed via the system response model. If the response model shows that the SLA is still met with the proposed operator arrangement, on the reduced cluster resources, then the system can be reconfigured. This allows costs to be saved during periods of low workload.

3.3.2 Performance modeling of DSPS work flows

The next key feature of this work is the modelling of the end-to-end latency / throughput of a DSPS work flow with respect to incoming workload and resource usage. This research is built on previous work applying a queueing theory approach to complex event processing and multi-tier architectures. A key advantage of using a queuing theory approach, compared to treating each work flow as a black box model, is the ability to identify bottlenecks at the individual operator instance level and also to assess the impact of new operator arrangements on the latency/throughput of the DSPS work flows. This will be vital in assessing whether a new operator arrangement, produced by the scheduling algorithms, will meet a given SLA.

3.3.3 Scheduling and scaling decisions

There have been several papers published on optimized scheduling algorithms for Apache Storm and other stream processing systems. Therefore, investigating a new scheduling algorithm is not a primary focus of this doctoral work. However, the system response model allows any proposed operator deployment, produced by a scheduling algorithm, to be assessed against the SLA. If the modeling shows that the new operator deployment is unlikely to meet the SLA, then additional resources are added and the scheduling algorithm is run again. Issues around when and if to scale are investigated once the accuracy of the incoming workload and system response predictions has been established.

3.4 Enabling Elastic Stream Processing in Shared Clusters

Distributed data stream processing has become an increasingly popular computational framework due to many emerging applications which require real-time processing of data such as dynamic content delivery and security event analysis. These distributed data stream processing applications are often run on shared, multi-tenant clusters as companies try to consolidate from dedicated clusters for each application (batch and streaming) to a single cluster using a global cluster manager such as Hadoop YARN. In shared cluster environments, guaranteeing the quality of service constraints for throughput and response time for both stream processing applications and batch applications is a significant challenge. Stream processing applications often face an elastic demand where the input rate can vary drastically. The typical solution to solve workload elasticity is to

guarantee enough resources to the application, but this solution is not possible when resources are being shared among multiple applications.
In the paper, "Enabling Elastic Stream Processing in Shared Clusters", by Jack Li, Calton Pu, Yuan Chen, Daniel Gmach, Dejan Milojicic, IEEE 2016, an approach is presented for supporting elastic scaling of distributed data stream processing applications and efficiently scheduling and coordinating stream processing with batch processing in shared clusters. The solution consists of a congestion detection monitor which detects bottlenecks in the streaming system and a global state manager that performs non-disruptive, stateful scaling of streaming applications. The solution is implemented using Storm, a popular stream processing framework, and tested on a Hadoop YARN cluster using a real-time security event processing workload. The experimental results show that their solution improves stream processing application throughput by 49% over default Storm while decreasing average request response times by 58%.[8]

4 Research analysis and Proposals

Spark Streaming is regarded as the most accessible of the candidate Stream Processing technologies in terms of programmability. Spark Streaming, as of version 2.1.0, has official support for Java, Scala, and Python. Python use with PySpark allows for Spark Streaming projects to be written and run directly from the Python source files deployed to a Spark installation. With Storm and Samza, there are a number of interfaces that are required to be implemented to make even the most simple of projects. Spark Streaming has been noted as the most simple of the candidate DSPS technologies in terms of programmability and overall learning curve. Storm presents the next most simple technology with a lot of the configuration and software dependencies being abstracted away from the programmer. Storm shows great overall performance in processing data in real-time, and also allows for extensible projects to be created, suitable for large projects, intended to be in production for long periods of time, and subject to change.

In any stream processing system, there are three steps in processing the data.

1. *Receiving the data*: The data is received from sources using Receivers or otherwise. Different input sources provide different guarantees.
2. *Transforming the data*: The received data is transformed using DStream and RDD(Resilient Distributed Datasets) transformations(Spark 2.1.0).

3. *Pushing out the data*: The final transformed data is pushed out to external systems like file systems, databases, dashboards, etc.

Most real-world data is produced continuously and arrives continuously. Traditionally, this continuous data flow had to be interrupted, and data had to be either gathered in a central location or cut in batches in order for applications to interact with it. Thus it would be a great idea to have a streaming paradigm in which applications can be developed directly on continuously-flowing data, enabling benefits such as localized and correct state, better isolation of applications and teams, and better handling of time series data.

Let us consider a stream processing pipeline that is network-bottlenecked. At the hardware level, there is no reason for such a tradeoff to exist. The network's capacity is ultimately what dictates both the maximum attainable throughput and the lowest attainable latency. A well-engineered software system will achieve the physical limits allowed by the network and not introduce bottlenecks itself. Latency can also be improved by using adaptive time out strategy, shared queues among different tasks and latency based load balancing.

Performance of Stream Processing systems can be improved by avoiding inefficient revaluation of already evaluated parts of the stream. Incremental evaluation technique helps in boosting the overall performance of the stream processing system. Operator aware data structures along with incremental evaluation algorithms outperform the counterparts of relational stream processing systems.

In order to maintain SLAs for Stream Processing systems proactive scaling is essential. This proactive system would need to be able to anticipate increases in incoming workload, assess via a model of the Stream Processing systems if the predicted incoming workload will result in a breach of the SLA, and if so find the optimum operator and cluster deployment to maintain the SLA. The operator scheduling systems described above provide one part of this requirement, the optimum deployment, but the other elements also need to be investigated for Stream Processing systems.

Elastic scaling allows a data stream processing system to react to a dynamically changing query or event workload by automatically scaling in or out. Thereby, both unpredictable load peaks as well as underload situations can be handled. However, each scaling decision comes with a latency penalty due to the required operator movements. Therefore, in practice an elastic system might be able to improve the

system utilization, however it is not able to provide latency guarantees defined by a service level agreement (SLA).

5 Summary

This project has been very helpful in gaining fruitful insight into current technologies in Stream Processing as a Service. It is a vast field of research and a lot of advanced studies and research is being carried out to make Stream Processing highly efficient and accurate.

This paper focuses on the existing technologies and the latest researches in the field of Stream Processing with regard to reducing latency, boosting performance, proactive scaling and improved elasticity. Various novel technniques have been proposed by the researchers and they are dynamically evolving the Stream Processing paradigm.
Stream processing has a great future and will become very important for most comp anies.

6 References

1. http://info.telligent-data.com/blog/the-differences-between-hadoop-and-stream-processing
2. https://www.academia.edu/29587226/Real_Time_Big_Data_Analytics_and_Stream_Processing_for_Enterprise_Ready_Hadoop
3. "Stream Processing in the Cloud" by Wilhelm Kleiminger Submitted in part fulfilment of the requirements for the MEng Honours degree in Computing of Imperial College.
4. "An Evaluation of Data Stream Processing Systems for Data Driven Applications", by Jonathan Samosir, Maria Indrawan-Santiago and Pari Delir Haghighi in the proceedings of ICCS 2016, The International Conference on Computational Science.
5. "New techniques to lower the tail latency in stream processing systems", by guangxiang du
6. "Operator-aware approach for boosting performance in RDF stream processing", by Danh Le-Phuoc, Web Semantics: Science, Services and Agents on the World Wide Web, January 2017
7. "Proactive Scaling of Distributed Stream Processing Work Flows using Workload Modelling", by Thomas Cooper, 2016 ACM.
8. "Enabling Elastic Stream Processing in Shared Clusters", by Jack Li, Calton Pu, Yuan Chen, Daniel Gmach, Dejan Milojicic, IEEE 2016.