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

Spring 2017

SWE 561 Cloud Computing

Submitted by,

Sumitha P K

92065

Table of Contents

1 Introduction

An SPaaS system provides data stream processing as a service. It executes a set of continuous queries on a potentially unbounded data stream. A data stream is said to be unbounded, if the data set is produced incrementally over time, rather than being completely available before the processing starts. Internet traffic analysis, financial trading, monitoring of manufacturing equipment and transaction log mining are some of the examples with high volume of unbounded data. These scenarios require a high throughput and a low end-to-end latency from the system, despite possible fluctuations in the workload. The emergence of data-intensive streaming applications has begun to change the traditional way of viewing database as a static store of data.

In stream processing, data has to be processed fast, so that a firm can react to changing business conditions in real time. A "too late architecture" cannot realize these use cases. A stream processing service aims to process, analyze and make decisions on the fly based on huge quantities of data streams being dynamically generated at high rates. Stream services often exhibit spike work- loads.

Research in data stream processing has gained additional momentum in recent years.

Data stream processing can be divided into 3 generations [1] :

- First generation stream processing systems - They have been built as stand-alone prototypes or as extensions of existing database engines. They were developed with a specific use case in mind and are very limited regarding the supported operator types as well as available functionalities. Examples : Niagara, Telegraph.
- Second generation systems – They extended the ideas of data stream processing with advanced features such as fault tolerance, adaptive query processing, as well as an enhanced operator expressiveness. Examples : Borealis, System S.
- Third generation systems - Their design is strongly driven by the trend towards cloud computing, which requires the data stream processing engines to be highly scalable and robust towards faults. Well-known systems include Apache S4, D-Streams, Storm and StreamCloud.

2 Problem definition

Stream Processing as a service should be implemented as an elastic scalable and fault tolerant system. Scalable algorithms and infrastructure need to be devised that scale to realistic scenarios with high stream frequencies, large numbers of concurrent queries and large dynamic and static data sizes. It should also provide with the possibility to deploy them in a hosted Cloud environment to achieve elasticity in the load profiles and enable "pay-as-you-go" scenarios. Robust fault tolerant mechanisms need to be designed with less overhead. [2]

As users of unbounded data stream applications expect fresh results, we see a new kind of stream processing systems (SPS) that are designed to scale to large numbers of cloud-hosted machines.[5] Some of the challenges faced by such systems are :

- On demand parallelism - In order to benefit from the "pay-as-you-go" model of cloud computing, they must scale out on demand, parallelising operators and acquiring additional virtual machines (VMs) and when the workload increases

- Resource efficient failure recovery - Failures are common with deployments on hundreds of VMs—systems must be fault-tolerant with fast recovery times, yet low per-machine overheads.

Thus, the main problem is how to achieve these two goals when stream queries include stateful operators, which must be scaled out and recovered without affecting query results.

Another issue is load balancing so that data processing operators are allocated at the nodes of clusters in such a way as to balance workload dynamically. Since the data volume and rate can be unpredictable, static mapping between operators and cluster resources often results in unbalanced operator load distribution.

3 Related research and solutions

3.1 Elastic scalability mechanisms

An important aspect of any data stream processing system is its ability to scale with increasing load, where the load is characterized by the number and complexity of issued queries and the rate of incoming events to be analyzed.
Cloud based data streaming engines, in particular, is designed to dynamically scale to hundreds of computing nodes and automatically cope with varying workloads.
The design of such engines poses two major technical challenges:

1. Data Partitioning : The major challenge is to allow parallel evaluation of multiple data elements ensuring semantical correctness. This involves a reasonable data partitioning and merging scheme as well as mechanisms to detect points for parallelization.[1]

2. Query Partitioning : If a single host is not able to process all incoming data or all queries running in the system a distributed setup is applied. This involves the question, how to distribute the load across available hosts and how to achieve a load balance between these machines.[1]

3.1.1 Stela (STream processing ELAsticity)

A stream processing system that supports scale-out and scale-in operations in an on-demand manner, i.e., when the user requests such a scaling operation. Stela meets two goals:
1. It optimizes post-scaling throughput,
2. It minimizes interruption to the ongoing computation while the scaling operation is being carried out.

Customers want to use a framework that can process large dynamic streams of data on the fly and serve results with high throughput. To meet this demand, several new stream processing engines have been developed recently, and are widely in use in industry, e.g., Storm , System S , Spark Streaming , and others. Apache Storm is the most popular among these.[3]

A Storm application uses a directed graph (dataflow) of operators (called “bolts”) that runs user-defined code to process the streaming data.[3]

Unfortunately, these new stream processing systems used in industry largely lack an ability to seamlessly and efficiently scale the number of servers in an on-demand manner.

To select the best operators to give more resources when scaling-out, Stela uses a new metric called ETP (Effective Throughput Percentage). The key intuition behind ETP is to capture those operators (e.g., bolts and spouts in Storm) that are both congested, i.e., are being overburdened with incoming tuples, and affect throughput the most because they reach a large number of sink operators. For scale-in, ETP-based approach is used to decide which machine to remove and where to migrate operator[3].

3.1.2 Elastic auto-parallelization scheme

In the research paper "Elastic Scaling for Data Stream Processing", by Bugˇra Gedik, Scott Schneider, Martin Hirzel, and Kun-Lung Wu [4], the authors have proposed auto parallelization scheme.
Streaming applications are structured as directed graphs where vertices are operators and edges are data streams. To scale such applications, the stream processing system is free to decide how the application graph will be mapped to the set of available hosts. Auto-parallelization is an effective technique that can be used to scale stream processing applications in a transparent manner. It involves detecting parallel regions in the application graph that can be replicated on multiple hosts, such that each instance of the replicated region (which we refer to as a channel) handles a subset of the data flow in order to increase the throughput. This form of parallelism is known as data parallelism. Transparent data parallelization involves detecting parallel regions without direct involvement of the application developer and applying runtime mechanisms to ensure safety: the parallelized application produces the same results as the sequential one.[4]

3.1.3 Using operator state management.

In the research paper "Integrating Scale Out and Fault Tolerance in Stream Processing using Operator State Management" by Raul Castro Fernandez, Matteo Migliavacca, Evangelia, the key idea is to expose internal operator state explicitly to the SPS(Stream Processing Systems) through a set of state management primitives.

Based on which they have described an integrated approach for dynamic scale out.[5]

Stream processing operators can be stateless (e.g. filter or map) or stateful (e.g. join or aggregate).Externalized operator state is check pointed periodically by the SPS and backed up to upstream VMs. The SPS identifies individual operator bottlenecks using heuristics and automatically scales them out by allocating new VMs and partitioning the check pointed state. At any point, restoring check pointed state on a new VM and replaying unprocessed tuples recover failed operators. This approach has been evaluated with the Linear Road Benchmark on the Amazon EC2 cloud platform and show that it can scale automatically to a load factor of L=350 with 50 VMs, while recovering quickly from failures. This scale out mechanism partitions operator state and streams without violating query semantics.[5]

3.2 Improving Fault Tolerance

New fault tolerance mechanisms strive to be more robust and at the same time introduce less overhead. Data streaming systems aimed at cloud platforms must thus be designed to efficiently cope with faults. Faults may appear in a data streaming system at several different places. However, it can be generally assumed that a fault will either affect a single message (e.g. an overloaded network link discarding packets) or a computational element (e.g. an operator becomes unavailable after the crash of the CPU it was running in). As a consequence, one can differentiate between State management techniques employed to make operator state survive faults and Event tracking techniques employed to track the correct handling of events injected in the system.[1]

3.2.1 Active and Passive replication

In the tutorial “Cloud-based Data Stream Processing” by Thomas Heinze, Leonardo Querzoni, Zbigniew Jerzak [1], the authors have described how the data stream processing system allows a user to scale out to hundreds of processing nodes, and ensures a fault tolerant execution even in an error-prone environment.

Active replication requires the system to evaluate an event in parallel on k identical operators (where k is the replication degree) such that the operator state is correctly maintained as long as less than k operators fails at the same time. In order for this technique to work correctly operator replicas must be guaranteed to evaluate the same set of events in the same order and they must implement deterministic computations . While active replication is considered very effective in providing uninterrupted service in the presence of faults, it is generally regarded as an inefficient technique for data stream processing systems.

Passive replication is a lazy technique where operators are required to periodically backup their state to make it survive possible future faults. Upon a fault, a new copy of the failed operator is instantiated and its state restored from the backup. The advantage of this approach is that few resources are periodically consumed at runtime to backup operator state, while most of the load is incurred only when an operator fails. In this case, in fact, beside restoring the latest operator internal state backup, all events managed by the failed operator before its failure that were not included in the backup must be evaluated again. Passive replication is today the most commonly used approach for providing operator fault tolerance in data stream processing systems, because it can easily be adapted and is tailored to work efficiently with different system architectures.[1]

3.2.2 StreamCloud fault tolerance protocol

In the research paper "StreamCloud: An Elastic Parallel-Distributed Stream Processing Engine", by Vincenzo Gulisano, StreamCloud fault tolerance protocol has been studied and implemented.

StreamCloud is an elastic parallel distributed stream processing engine that enables for processing of large data stream volumes. StreamCloud minimizes the distribution and parallelization overhead introducing novel techniques that split queries into parallel subqueries and allocate them to independent sets of nodes. Moreover, StreamCloud defines a novel fault tolerance protocol that introduces minimal overhead while providing fast recovery.[6]

Three main base techniques can be adopted in order to provide fault tolerance to the operators of a query: active standby, passive standby and upstream backup. The three basic approaches differ primarily in how to maintain the information that might be lost in case of failure .

The **Active standby** (or active replicas) protocol provides fault tolerance for an operator by means of a secondary copy of the operator that is fed with the same tuples forwarded to its primary peer.

With the **Passive standby** fault tolerance technique, the state of the operators belonging to the node we want to protect are periodically copied to a secondary node. Copies can be continuously installed at replica nodes or they can be stored in a dedicated server and installed in a replacement instance in case of failure.

Finally, **Upstream Backup** defines a different approach where no replicas nodes are used. The idea is to maintain the tuples forwarded to the primary operator in order to replay them to the replacement operator in case of failure. Tuples need to be maintained by the operator upstream peer (tuples will be lost if maintained by the same instance we want to protect from faults).

3.2.3 Discretized Streams

Record-at-a-time systems provide recovery through either replication, where there are two copies of each processing node, or upstream backup, where nodes buffer sent messages and re play them to a second copy of a failed downstream node. Neither approach is attractive in large clusters: replication needs 2x the hardware and may not work if two nodes fail, while upstream backup takes a long time to recover, as the entire system must wait for the standby node to recover the failed node's state.

In the paper"Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters", by Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, Ion Stoica. , new programming model is presented, discretized streams (D-Streams), that overcomes these challenges. The key idea behind D-Streams is to treat a streaming computation as a series of deterministic batch computations on small time intervals. For example, we might place the data received each second into a new interval, and run a MapReduce operation on each interval to compute a count. Similarly, we can perform a running count over several intervals by adding the new counts from each interval to the old result. Two immediate advantages of the D-Stream model are that consistency is well-defined (each record is processed atomically with the interval in which it arrives), and that the processing model is easy to unify with batch systems. In addition, we can use similar recovery mechanisms to batch systems, albeit at a much smaller timescale, to mitigate failures more efficiently than existing streaming systems, i.e., recover data faster at a lower cost.[9]

3.3 Load balancing in Stream Processing

Stream processing systems must handle stream data coming from real-time, high-throughput applications, for example in financial trading. Timely processing of streams is important and requires sufficient available resources to achieve high throughput and deliver accurate results. However, static allocation of stream processing resources in terms of machines is inefficient when input streams have significant rate variations, machines remain underutilized for long periods of average load.[7]

3.3.1 Adaptive load balancing

In “Balancing Load in Stream Processing with the Cloud”, by Wilhelm Kleiminger, Evangelia Kalyvianaki , Peter Pietzuch, adaptive load balancing is proposed.
The paper evaluates the load balancer as it propagates windows to the cloud stream processors and uses more cloud resources with increasing input rates. As the input rates increase, the local processor handles a constant portion of the input data. At the same time, the adaptive load balancer forwards a larger number of windows to the cloud processor—the throughout of the cloud processor increases. At an input rate of 12,000 KB/s, our 4 nodes in the test cloud are overloaded. Since the maximum throughput (see Table I) is reached, windows are discarded. To scale beyond 4 nodes, they have moved the cloud stream processor to 10 nodes in the Amazon EC2 cloud. With the load balancer running in our university data centre, they have achieved an average processing latency of 2 seconds for a window size of 10,000 tuples.

3.3.2 Optimization method to minimize load variance

In “Optimization of Load Adaptive Distributed Stream Processing Services”, by Xing Wu , Yan Liu, they have proposed proposes an optimization method that combines correlation of resource utilization of nodes and capacity of clusters. Their implementation is evaluated by a top-N topic list application on Twitter streams. The results demonstrate improved stream processing throughputs and cluster resource utilization. [8]

In the paper, an optimization method for dynamic operator distribution of stream processing services has been proposed. The optimization method is based on the architecture of DoDo, a software layer constructed on top of networked cluster nodes and provides a flexible framework to support the mobility of operators or processing elements. The optimization algorithm takes both the correlations between the load series of clusters and the capacities of clusters into consideration.
The goal of optimization is to minimize the average load variance and maximize the throughputs of stream processing services. They have designed an optimization method applied in the mapping process to assign PEs to a cluster node At the bottom layer, the physical nodes are connected by a high bandwidth network in a cloud environment. At the top layers, as an example, both opera- tors PEA and PEB are allocated on Cluster1. Hence when Cluster1 is overloaded, moving PEB or PEA to Cluster2 could potentially solve the problem. [8]

4 Potential benefits and contributions

Employing elastic scalability mechanisms for stream processing is very beneficial considering the fact that traffic patterns vary and resource availability fluctuates as these long-running applications perform their data analysis tasks while coexisting with other applications that share the same computational environment. Bottleneck detection is very crucial because they limit scalability. When detecting operator bottlenecks, it is important to focus on compute bottlenecks because they are the most common type observed in practice. Also, adopting a scaling policy based on measured CPU utilization of operators is very effective.

With regard to fault tolerance, the main aim of Stream Processing is to develop a fault tolerant scheme that does not require reserving half the memory in a cluster for fault tolerance. Discretized streams (D-Streams), a stream programming model for large clusters that provides consistency, efficient fault recovery, and powerful integration with batch systems. The key idea is to treat streaming as a series of short batch jobs, and bring down the latency of these jobs as much as possible. This brings many of the benefits of batch processing models to stream processing, including clear consistency semantics and a new parallel recovery technique that we believe is the first truly cost-efficient recovery technique for stream processing in large clusters.

A huge number of applications such as network monitoring, traffic engineering systems, intelligent routing of cars, sensor networks, mobile telecommunications, logistics applications and air traffic control require continuous and timely processing of high volume of data originated from many distributed sources as well as mobile communication and monitoring. Another contribution of this paper is to look into the novel approaches in load balancing. The benefits of the optimization method on better utilizing available resources, which is a useful feature for cloud computing services, as it helps to prevent unnecessary autoscaling to clusters that are overloaded while other clusters still have capacities to handle streams.

5 Summary

This project has been very helpful in learning and understanding basic concepts of data stream processing and also the problems faced while implementing stream processing as a service in cloud.
As the data is immense and unbounded, it is necessary to produce accurate results on the fly every time without any exception. This paper focuses on the researches done in the area of elastic scalability, fault tolerance and load balancing in the context of stream processing. Researchers have proposed various novel methods so far and it is always interesting to learn how cloud computing technology is dynamically evolving.
This is a very promising field for further studies and research.

6 References

1. “Cloud-based Data Stream Processing” by Thomas Heinze, Leonardo Querzoni, Zbigniew Jerzak

2. “Elastic and scalable processing of Linked Stream Data in the Cloud”, by Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Chan Le Van, and Manfred Hauswirth, Digital Enterprise Research Institute, National University of Ireland, Galway

3. “Stela: Enabling Stream Processing Systems to Scale-in and Scale-out On-demand” by Le Xu , Boyang Peng , Indranil Gupta.

4. “Elastic Scaling for Data Stream Processing”, by Bug˘ra Gedik, Scott Schneider, Martin Hirzel, and Kun-Lung Wu in the proceedings of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 25, NO. 6, JUNE 2014.

5. “Integrating Scale Out and Fault Tolerance in Stream Processing using Operator State Management” by Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, Peter Pietzuch in the proceedings of SIGMOD’13, June 22–27, 2013, New York, New York, USA.

6. “streamcloud: An Elastic Parallel-Distributed Stream Processing Engine”, by Vincenzo Gulisano

7. “Balancing Load in Stream Processing with the Cloud”, by Wilhelm Kleiminger, Evangelia Kalyvianaki , Peter Pietzuch

8. “Optimization of Load Adaptive Distributed Stream Processing Services”, by Xing Wu , Yan Liu in the proceedings of 2014 IEEE International Conference on Services Computing.

9. “Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters”, by Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, Ion Stoica.