IMPROVED REAL-TIME DATA ELASTICITY ON STREAM CLOUD

Chetan anil Joshi

Abstract

Most of the applications in cloud domains such as online data processing, Froud detection, large scale sensor network etc. where large amount of data should processed in real time. Earlier, for data stream processing, the centralized system environment was using with store and then process paradigms. After that some advancement has been introduced with distributed environment for data stream processing. Data Stream processing using novel computing paradigm which take query as input and splits that query into multiple sub queries and process the data on multiple sub clusters in such a way that reduces the distribution overheads. This kind of application generates very high input data which needs to process with the available clusters So High availability and elasticity are two key characteristics on the cloud computing services. High availability ensures that the cloud applications are sensible to failure. Elasticity is a key feature of cloud computing where availability of resources are related with the runtime demand. So in this paper we present a comprehensive framework for obtaining elasticity and scheduling technique for highly availability.

Keywords: *Scalability, Elasticity, High availability, Load balancing, Reliability.*

Introduction

Number of real time applications in which large amounts of data should process continuously. But there are some limitation comes with the traditional store the process paradigm [1]. So for overcoming this issue, some advancement has been presented in the stream process engines. Stream process engines are computing systems which are designed to process continuous stream of input data with the minimum time delay. Instead of store then process, in this system data streams are process on the fly using continuous queries. This is due to the amount of input data which discourage persistent storage and the prompt result requirement. Here the query is continually standing in streaming tuple and produces continues output.

Here in this system there is substantial development in the stream processing engine. Earlier it was running on the centralized stream processing engine [2]. Centralized engine using store then process paradigm which causes unnecessary value storage and other limitations. But now it's also running on the distributed environment. With distributed environment stream process engine distributes different queries among a cluster of nodes which we is called it as interquery parallelism or distributing different operators of the query across different nodes which is called as interoperator parallelism [3]. Most of the applications for scalable stream processing engine which need to aggregate the computing power of hundreds which need to process the millions of tuples per second. Here for obtaining higher scalability and avoiding the single node bottleneck problem stream process engine need to lies in distributed stream process engine with intra operator parallelism [4].

While doing the query parallelization, this requires to addressing additional number of challenges. Query parallelization should be semantically and syntactically transparent. Semantic transparent means query should produce exact the same output like non parallel queries. Syntactically transparency means the query should get automatically parallelized. It should be oblivious to the user. While doing parallelization, usage of resources should also be the cost effective. The parallel stream process engine should be elastic and it should manage the amount of its resources to the workload. The elasticity also combined with the dynamic load balancing technique. It should able to manage load across available nodes or the clusters.

In this paper we are presenting as inproved real time data elasticity on stream cloud [6] and elastic stream process engine which provides a transparent query parallelization. That is stream processing engine will accept the input query which is automatically paralyzed. This query will splits in multiple sub query and process individual sub query on clusters of nodes [6]. Stream process engine handles the stream of tuples. A stream is potentially infinite sequence of tuples which is sharing a given schema. All tuples having a time stamp attribute which sets at the data source [7]. The data source has clocks which are synchronized with the other system nodes. When clock synchronization is not feasible tuple can be time stamped at the entry point of the data streaming system. In SPE, query is defined as a cyclic graph where as node is an operator and edges defines the data flow. Here focus has given on stateless and steteful operators [8]. Stateless operator does not keep any state across tuples and perform computing operation only based on input tuple. (eg. Map,union and filter. Stateful operators perform computation operations on sliding windows of tuple defined over a fixed time period (eg. Aggregate, cartision product and join).

So in the proposed system stream cloud gives high scalability, reliability for stream processing engines. The input queries which are executing are automatically and all tuples provide transparent parallelization. System gives high scalability by giving interoperator parallelism.

System also contains load balancing, task assignment and scheduling for the execution nodes which executes its operation based on available execution information. Additionally system calculates the execution power capacity of each node. So execution task assignment can be performed efficiently System also uses heart bit technology which gives node alive status with transferring the signals in between each others for informing the live status.

Literature Survey:

A literature survey includes related work in the data stream processing in the stream line cloud for real time data processing which shows the system reliability and scalability. Some of legacy applications and most of the real time applications data processing should be continuous. So for such applications we need to use stream process engines. There has been advancement from centralized to distributed environment. There is a substantial change from store then process to tuple-on-the fly. It is also called as continues queries in which queries are continuously standing with a streaming data for real time processing. while doing the parallel stream processing, attention must be given at stateful operators (Aggregate, joins and Cartesian products) and stateless operators (map unions and filters) .Also here the basically two factors has for number of hops performed by each tuple and communication fan out 0f each node has considered. Here there are different strategies for parallelization such as,

Operator cloud strategy

In the operator cloud strategy, the query parallelization unit is a single operator. So each of input data deployed on different subset of node. We can also called it as a subclustor. If we will consider that there are 15 nodes presented and 5 operators are presented. Communication happens from every subclustor to all its peers in the next presented subclustors. So total number of hopes is 5 and fan out for every node is 15.

Operator set cloud strategy

The above operator cloud strategy has been exhibits the trade-off in-between the distribution cost and number of hopes. The operator set cloud strategy introduced for minimizing both things at the same time. Here for guarantee semantic transparency the communication is required to be done with stateful operators. Here each input query is splits in between the multiple sub queries as stateful operators plus an additional one. Sub query consists of stateful operators followed by stateless operators which are connected to its output [6]. Here these both strategies minimize the number of hopes and fan out.

Applications of the Data Streaming Systems

I n this section we show the application area in some last decade in the field of data streaming. These presented applications include both existing and emerging applications that share the same need for high processing throughput with low latency and legacy constraints. Among others, examples of these scenarios include fraud detection, network security, financial markets, large scale sensor networks and military applications.

Fraud detection in cellular telephony

Fraud detection methods are continuously being developed to checkmate criminals who adopt new techniques regularly. The development of new fraud detection method is made difficult due to the severe limitation imposed by restricted information flow about the outcome of fraud detection efforts.

Fraud detection applications in mobile telephony require the ability of processing data whose size is in the order of tens or hundreds of thousand messages per second. In cellular telephone fraud detection applications, there is a need for low latency processing arises from the fact that the faster the detection, the lower the amount of money the fraudster costs to the company. That is, if a cloned number is detected after one week, the money spent during that week is lost by the company.

On-line trading

The act of buying and selling international currencies, stocks, bonds and other financial instruments through the internet scenarios involving credit card transactions are another example of applications demanding for high processing capacity and very low processing latency. With respect to applications that involve credit cards transactions it is even more imperative to provide low latency guarantees as such applications must comply with strict time limitations that are often smaller than one second.

Financial Markets applications

Another example of existing application demanding for processing of large volume data with low delay is related to financial Markets. The growing rate has been so fast that the required capacity exceeded the 1 million messages per second in 2008. For on-line trading applications, financial market applications demand for very low processing latencies that are often below the one second threshold.

Network traffic monitoring

As considering applications monitoring traffic network (e.g., Intrusion Detection applications), the need for high processing capacity arises from the huge volume of data moving through the Internet. As per the Cooperative Association for Internet Data Analysis, an Internet Service Provider (ISP) usually sustains traffic volumes that is at least in the range of 10 GBps. With respect to applications that analyze the traffic to detect possible threats, the need for high processing capacity arises not only from the big traffic volume that must be processed in real-time in order to block possibly harmful events but also from the type of computations run over the data, that usually define not trivial aggregation and comparison of on-line and historic data.

Sensor networks, RFID networks

Several emerging applications demanding for high processing capacity and low processing latency arise from sensor networks. These applications are becoming popular due to the decreasing costs of sensors that, nowadays, allow for the creation of big networks of such devices with small deployment costs. Examples of such scenarios include military applications where soldiers (or vehicles) can be equipped with GPS devices to monitor their location. Another sample scenario is RFID tagging for animal tracking or RFID tagging of products being produced or sold in big industry.

Another important aspect of the data processed by the presented application scenarios is the possible presence of incomplete and out-of-order data. Consider, as an example, a data streaming application used to process data collected from a sensor network. In such scenario, part of the data produced by sensor might disappear (e.g., when a sensor runs out of battery). In this case, the application should still consume the data produced by the other sensor and produce the desired result. Similarly, a sensor might experience some delay in the forwarding of the produced data. In this case, the data streaming application should address the possibility of late processing of information in order to correct imprecise results computed before.

SPEs offer the best capabilities since they are designed and optimized from scratch to address the requirements of stream processing. Both DBMSs and rule engines were originally architected for a different class of applications with different underlying assumptions and requirements.

Rules for Stream Processing

There are some rules and regulations has been defined which needs to be done for data stream processing [1]. These rules are as follows,
Rule 1: Keep the Data continuously moving
The first requirement for a real-time online data stream processing system is to process messages "in-stream", without any requirement to store them to perform any operation or sequence of operations. Ideally the system should also use an active (i.e., non-polling) processing model.
Rule 2: Query using SQL on Streams (StreamSQL)
The second requirement is to support a high-level "StreamSQL" language with built-in extensible stream oriented primitives and operators. In streaming applications, some querying mechanism must be used to find output events of interest or compute real-time analytics. Historically, for streaming applications, general purpose languages such as C++ or Java have been used as the workhorse development and programming tools. Unfortunately, relying on low-level programming schemes results in long development cycles and high maintenance costs.
Rule 3: Handle Stream Imperfections (Delayed, Missing and Out-of-Order Data
The third requirement is to have built-in mechanisms to provide resiliency against stream "imperfections", including missing and out-of-order data, which are commonly present in real-world data streams.
Rule 4: Generate Predictable Outcomes
The fourth requirement is that a stream processing engine must guarantee predictable and repeatable outcomes.
Rule 5: Integrate Stored and Streaming Data
The fifth requirement is to have the capability to efficiently store, access, and modify state information, and combine it with live streaming data. For seamless integration, the system should use a uniform language when dealing with either type of data.
Rule 6: Guarantee Data Safety and Availability
The sixth requirement is to ensure that the applications are up and available, and the integrity of the data maintained at all times, despite failures.
Rule 7: Partition and Scale Applications Automatically
The seventh requirement is to have the capability to distribute processing across multiple processors and machines to achieve incremental scalability. Ideally, the distribution should be automatic and transparent.
Rule 8: Process and Respond Instantaneously
The eighth requirement is that a stream processing system must have a highly-optimized, minimal-overhead execution engine to deliver real-time response for high-volume applications.

So for doing data streaming, these above criteria should get fulfilled. So stream process engine can accept the input and query will get processed.

Different Stream Process Engines
This section covers some of the stream processing engines and methodologies for data stream processing.

Pioneer SPE

This section will give a short overview of some of the pioneer SPEs prototypes. We present how they marked an evolution with respect to previous existing solutions overcoming their limitations. We also provide an overview of the evolution of these pioneer SPEs and their limitations and introduce the challenges that motivated our work.
This idea starting approximately from the year 2000, several SPE prototypes have emerged. These prototypes have been designed either as improvements of DB based solutions (i.e., solutions that still rely on DBs) either as “fresh" solutions that do not rely on DBs. With respect to the prototype applications that were designed to improve DB based solutions to fit with the requirements of data streaming, we introduce Cougar, Telegraph and NiagaraCQ. The Cougar research project focused on sensor databases, where long running queries are issued to combine live data from a sensor network with some stored data. NiagaraCQ is one of the first research projects that extends a query language to adapt it to continuous queries. The project focused on how to efficiently run continuous queries over XML data files. Where as the TelegraphCQ research project is one the first projects that marks a significant step in the evolution of SPEs as it relies on DBs to persist information but introduces separate data processing units that manipulate the data in parallel with Cougar, NiagaraCQ and TelegraphCQ, two research projects have focused on SPEs that do no rely on DBs to store or manipulate data.
With stream cloud, research has focused on how to ease the migration from DBMS to SPEs. One of the most important aspect of STREAM is the introduction of the CQL query language, an evolution of the SQL language, enriched with data streaming related operations. One of the requirements of data streaming applications is to define operations over portions of the incoming information (referred to as windows). The idea is to represent streams by means of relations so that data manipulation can be defined using SQL-like commands that operate on them. That is, the relevant portion of incoming information that is queried is maintained like a relation and its records are queried by means of traditional DBMSs queries. Finally, the resulting relation is converted to a stream of tuples outputted to the end user. But there are also some limitations for the stream processing engines. The main limitation of both centralized and distributed SPEs is that they both concentrate data streams to single nodes. A query running at a centralized SPE receives and processes all the data at the same node. In the case of a distributed SPE, even if different operators of a single query are deployed at different nodes, each data stream is processed by a single machine (i.e., all the input data stream is routed to the machine running the first operator of the query, the stream generated by the first operator is concentrated to the node running the second operator, and so on). This single-node bottleneck implies that centralized and distributed SPEs do not scale, their processing capacity is bound to the capacity of a single node. In order to overcome this limitation, data streams should not be processed by a single node.

Aurora: A Centralized Stream Processor
In Aurora, data is assumed to come from a variety of sources such as computer programs that generate values at regular or irregular intervals or hardware sensors[7]. We will use the term data source for either case. A data stream is a potentially unbounded collection of tuples generated by a data source. Unlike the tuples of the relational database model, stream tuples are generated in real-time and are typically not available in their entirety at any given point in time. Aurora processes tuples from incoming streams according to a specification made by an application administrator. Aurora is fundamentally a data-flow system and uses the popular boxes and arrows paradigm found in most process flow and workflow systems [8].

Actual implementation of the proposed system:

In the proposed system, we have focused on the Load balancing, HA and elasticity with the help of operator cloud strategies and operator set cloud strategies with some modified algorithms and concepts. These steps are as follows,

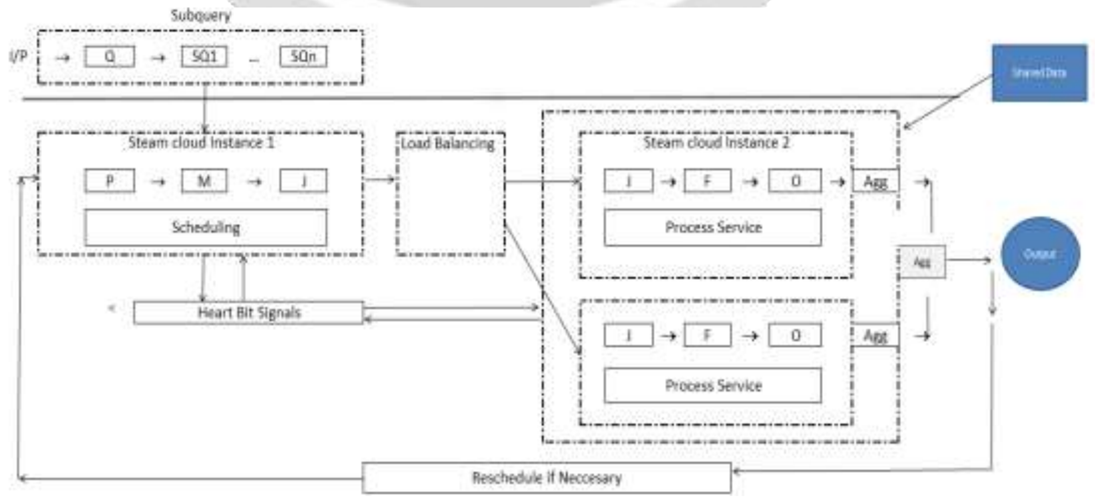
Algorithm : Input: Query data **Output**: Result files.

1. Data € Query data.
2. Split data using split criteria
3. BuildTasks() -> task
4. GetClientStatus ()

```
5.  Check busy Bit of the individual client.
6.  Check client Attribute -> RAM, Processor, Memory
7.  While(checkBusyBit())
8.  Start
9.  If (allTaskCompleted)
10. Break;
        11. Client feasibility checking for associated task.
        12. Schedule task to client
        13. TaskTssign() -> Client
        14. End
        15. Contribute Results ()
        16. Display Results ()
        17. End.
```

System Architecture

Architecture diagram shows the improved real-time data system on stream cloud system. Here figure represents complete 3T system. In this system, First Q input query comes and it is divided into multiple sub queries SQ1, SQ2. The spitted sub queries have been assigned to stream cloud instance for execution. While doing this, Stream cloud served uses some operations like parsing, Mapping and Job creation and job assignment. While doing this, it has to take care of the different status of the client nodes like computing power, CPU, RAM etc. So as per this criteria job has been assigning to execution. System also contains the heart bit signal passing mechanism which is using for checking the live status of the available nodes. System passes signals continuously for knowing node active status after fixed time of interval. If in case reply from node will not get within the time then system assumes that node as dead. This technique is useful for improving efficiency of the system. It also continuously checks whether new node is presented in the system which helps for HA. Once the operation done successfully, then it is directly giving the output.



Stream cloud compliments elastic resource management with dynamic load balancing for guaranty that the new instances are only be provisioned when a subclustors of node is not able to cope the incoming load. Here system also checks processing power of the computer system like CPU and RAM. So as per the collected information task can be assigned for the execution which helps for balancing the load across subclustors. System uses different strategies for obtaining elasticity such as

Elastic reconfiguration protocols

Subclustor reconfiguration required the transferring the owner ship from one instance of subclustor to another ie. from the old instance to new one in the same subclustor. This triggers reconfiguration by one or more reconfiguration actions.

Reconfiguration starts

The process is initiated by the elastic manager that decides to perform a reconfiguration either for provisioning, decommissioning, or load balancing purposes.

Windows reconfiguration protocol

The Window Recreation protocol aims at avoiding communication between the instances being reconfigured.

Conclusion

In this paper, we have presented improved Real-time data elasticity on stream cloud is presented. System also presents transparent query parallelization that keeps the syntax and semantics of the centralized system. HA, Elasticity and scalability are attained by means of novel parallelization strategy which minimizes the distribution overheads and also improves the performance of the system. Stream cloud elasticity and dynamic load balancing gives efficiency with minimizing number of resources. This evolution demonstrates the scalability, elasticity and high availability of stream cloud.

Refrences

[1] M. Stonebraker, U. C¸ etintemel, and S.B. Zdonik, “The 8 Requirements of Real-Time Stream Processing,” SIGMOD Record, vol. 34, no. 4, pp. 42-47, 2005.
[2] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M.A. Shah, “Telegraphcq: Continuous Dataflow Processing for an Uncertain World,” Proc. First Biennial Conf.Innovative Data Systems Research (CIDR), 2003.
[3] D.J. Abadi, Y. Ahmad, M. Balazinska, U. C¸ etintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S.B. Zdonik, “The Design of the Borealis Stream Processing Engine,” Proc. Second Biennial Conf. Innovative Data Systems Research (CIDR), pp. 277-289, 2005.
[4] M.T. O ¨ zsu and P. Valduriez, Principles of Distributed Database Systems, third ed. Springer, 2011
[5] V. Gulisano, R. Jime´nez-Peris, M. Patin˜ o-Martı´nez, and P. Valduriez, “Streamcloud: A Large Scale Data Streaming System,” Proc. Int’l Conf. Distributed Computing Systems (ICDCS ’10), pp. 126-137, 2010.
[6] StreamCloud: An Elastic and Scalable Data Streaming System ,Vincenzo Gulisano,Ricardo Jime´nez-Peris,Marta Patin˜ o-Martı´nez,Claudio Soriente,Patrick Valduriez VOL. 23, NO. 12, DECEMBER 2012.
[7) N. Tatbul, U. C¸ etintemel, and S.B. Zdonik, “Staying Fit: Efficient Load Shedding Techniques for Distributed Stream Processing,” Proc. Int’l Conf. Very Large Data Bases (VLDB), pp. 159-170, 2007.
[8) D.J. Abadi, D. Carney, U. C¸ etintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S.B. Zdonik, “Aurora: A New Model and Architecture for Data Stream Management,” VLDB J., vol. 12, no. 2, pp. 120-139, 2003.
[9) Y. Xing, S.B. Zdonik, and J.-H. Hwang, “Dynamic Load Distribution in the Borealis Stream Processor,” Proc. Int’l Conf. Data Eng. (ICDE), pp. 791-802, 2005.