www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issues 6 June 2016, Page No. 16842-16845

Bandwidth Optimization In Data Retrieval From Cloud Using Continuous Hive Language

S.Surendran[1] *Teaching Fellow,* ***K.Prema[2]*** *PG Student*

[1]Department of Computer Science and Engineering, BIT Campus, Anna University,
Tiruchirappalli 620024
Email:surendran.infotech@@gmail.com

[2]M.E Computer Science and Engineering, BIT Campus, Anna University,
Tiruchirappalli 620024
Email:prema012@gmail.com

ABSTRACT

In distributed applications data centers process high volume of data in order to process user request. Using SQL analyzer to process user queries is centralized and it is difficult to manage large data sets. Retrieving data from the storage is also difficult. Finally we can't execute the system in a parallel fashion by distributing data across a large number of machines. Systems that compute SQL analytics over geographically distributed data operate by pulling all data to a central location. This is problematic at large data scales due to expensive transoceanic links. So implement Continuous Hive (CHIVE) that facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to structure the data and query the data using a SQL-like language called HiveQL and it optimizes query plans to minimize their overall bandwidth consumption. The proposed system optimizes query execution plans and data replication to minimize bandwidth cost.

Key words - **Cloud computing, Event Stream Processing, Bandwidth optimization, Query processing.**

1 .INTRODUCTION

In big data the solution for big data analytics operates on large clusters of nodes, located in the same data center. Transporting high velocity data or huge size event consumes higher available bandwidth which makes the cost of data transfer is high.

CHIVE (Continuous HIVE) facilitates the execution of SQL queries to process large dataset. CHIVE executes continuous queries on data collected in online fashion. The fundamental concept of CHIVE is that it minimizes overall bandwidth consumption by optimizing query plan when it is used in distributed cloud. CHIVE rewrites query plans in such a way that event can be processed as close to their source. It limits the amount of data that needs to be sent through the network. It provides higher bandwidth reduction in distributed cloud. CHIVE executes queries in parallel manner in order to minimize bandwidth consumption and to increase execution speed.

Event Stream Processing (ESP) is developed for the construction of event-driven information systems. It includes event visualization, event-driven middleware, and event processing languages. ESP processes event stream data for identifying meaningful patterns in the streams. It provides techniques for detection of relationship between multiple events, event hierarchies, event correlation and event timing. Event Stream processing stores event processing workflows instead of storing data. The arrival of new data events triggers the execution of the workflows. ESP systems facilitate the creation and deployment of distributed event stream processing. ESP enables applications such as RFID event processing applications, fraud detection, process monitoring and location based services.

Event-driven architecture (EDA) is a framework for creation, detection, consumption of event and the responses to the event. Event-driven architecture consists of event creator, event consumer and event manager. The creator is the source of the event which only knows about that the event has occurred. Event consumers are the entities that process the event or they may be affected by the event. Event manager receives event notification from the event creator and forwards the events to registered consumers.

2. RELATED RESEARCH WORK

In batch processing user specifies the map function which process key value pairs. The reduce function merges all values with the same intermediate key. In this method programs are parallelized and executed on large cluster of machines. Query optimization technique is used in HIVE to

relieve the data analyst from optimizing queries before processing them. Chive is at the same level as Hive in the data processing, but it uses Event Stream Processing instead of batch processing. Chive avoids back-haul all event data to a single DC before processing. Chive stores raw events which are close to event source which requires transferring only minimal information in the network.

Complex Event Processing (CEP) technique analyzes continuous event streams. Complex Event Processing tools such as Esper can operate as a standalone application or can be embedded as a library in java applications. Esper has no out-of –the-box support for executing continuous queries distributed over multiple JVMs. Complex Event Processing supports scalability by running multiple instances in parallel.

3. DESIGN AND IMPLEMENTATION

Chive uses high level query language for event stream processing. Chive includes various client APIs such as Command Line Interface (CLI), web interface and java API. THE client submits Chive query expressions, network topology information, event type and the source of events. Query plan compiler creates distributed Chive query to be used in the network topology. It needs valid Chive query expression as an input to generate Chive query plan. Query execution library provides the functionality to execute Chive query plan. It uses windowing support and Query primitive library for running query plans. Query Deployment engine is responsible for deploying generated Optimized Query Plan (OQP) onto the processing nodes in the network topology.

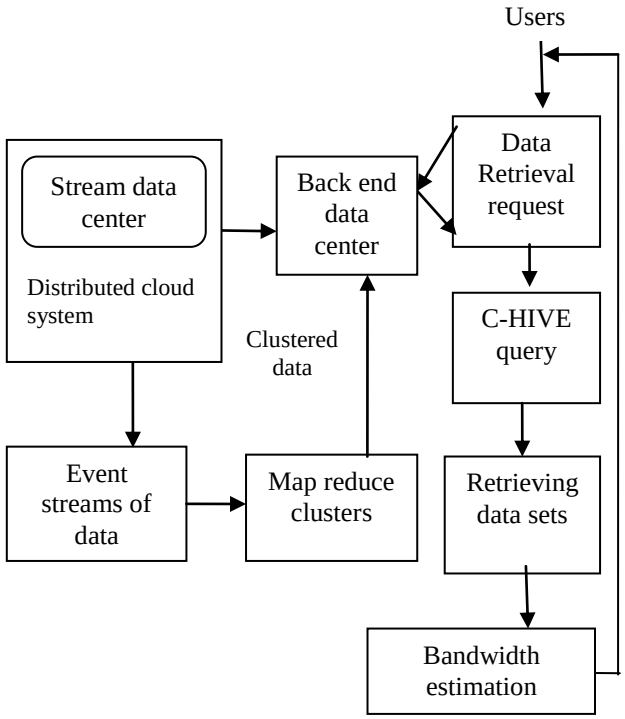

Figure 1: System Architecture

3.1 Distributed cloud framework:

The distributed systems are based on object-oriented programming (OOP) paradigm. While OOP is an intuitive way to model complex systems, it has been marginalized by the popular service-oriented architecture (SOA). However, at the system level, developers have to think in terms of loosely-coupled partitioned services, which often do not match the application's conceptual objects. This has contributed to the difficulty of building distributed systems by mainstream developers. The actor model brings OOP back to the system level with actors appearing to developers very much like the familiar model of interacting objects. Distributed cloud framework contains cloud owners, cloud provider and cloud users. Cloud owner is responsible for upload their data in cloud storage. Cloud provider is responsible for maintain the data and cloud users' access data from cloud storage. Then this framework also called as multi cloud system.

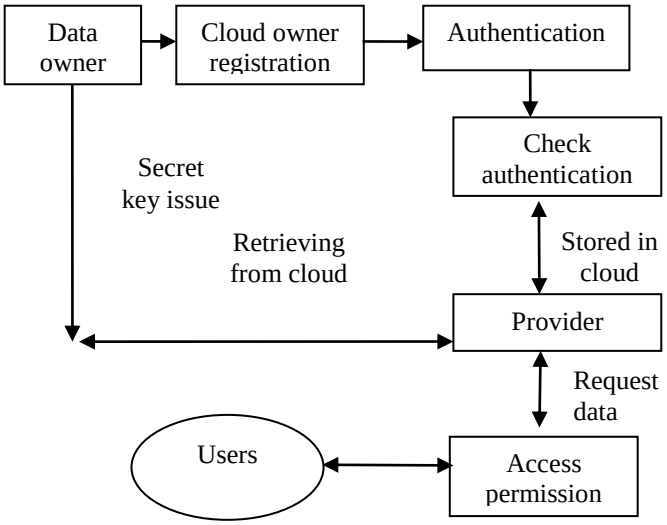

Figure 2: Distributed Cloud Framework

3.2 Event Stream Processing

Stream processing is mostly used application, even among big data users. Complex Event Processing, sometimes called Event Stream Processing, deals with discrete events where true Streaming engines deal with an ongoing flow of diverse sets of information (or at least they should). Event processing is a method of tracking and analyzing (processing) streams of information (data) about things that happen (events), and deriving a conclusion from them. Complex event processing, or CEP, is event processing that combines data from multiple sources to events or patterns that suggest more complicated circumstances. The goal of complex event processing is to identify meaningful events and respond to them as quickly as possible. The same logic that resulted in the creation of cloud-based data centers can be applied to cellular backhaul networks, according to the startup Parallel Wireless. In this we can get the cloud data as event streams and stored as CSV format.

DOI: 10.18535/ijecs/v5i6.20

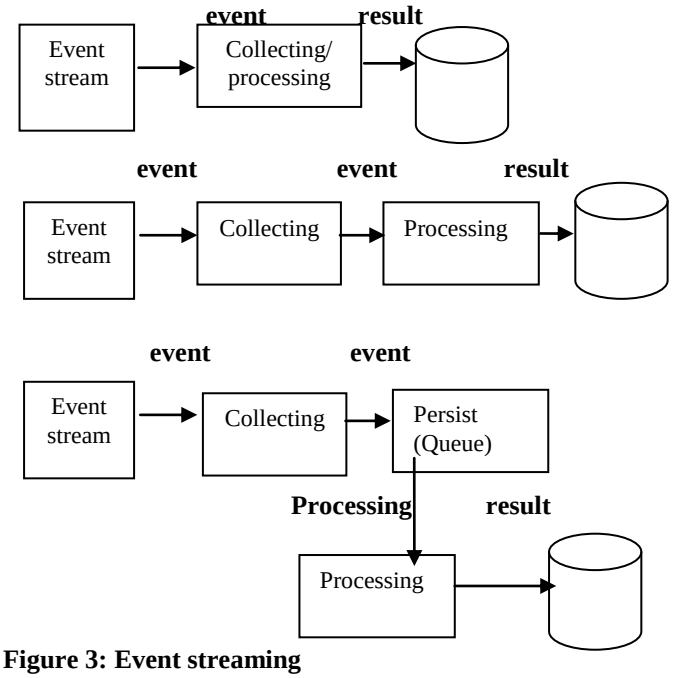

Figure 3: Event streaming

3.3 Map Reduce Clusters

Map-Reduce have been facilitated by big data as a programming framework to analyze massive amounts of data. It uses for distributed data processing on large datasets across a cluster of machines. Since the input data is too large, the computation needs to be distributed across thousands of machines within a cluster in order to finish each part of computation in a reasonable amount of time. Map-Reduce programming model using two components: a Job Tracker (master node) and many Task Trackers (slave nodes). The Job Tracker is responsible for accepting job requests, for splitting the data input, for defining the tasks required for the job, for assigning those tasks to be executed in parallel across the slaves, for monitoring the progress and finally for handling occurring failures.

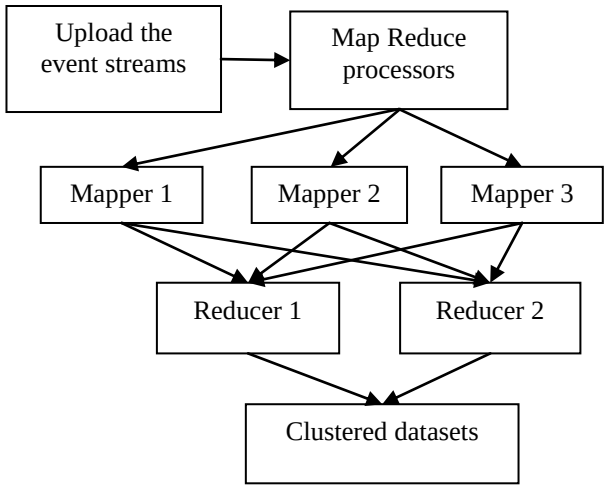

Figure 4: Map Reduced clusters

3.4 Data retrieval:

Traditionally adding new data into Hive requires gathering a large amount of data onto HDFS and then periodically adding a new partition. But in this module, we can implement continuous hive query. This is essentially a "stream insertion". Insertion of new data into an existing partition is not permitted in hive query. So using C-Hive Streaming API allows data to be pumped continuously into Hive. The incoming data can be continuously committed in small batches of records into an existing Hive partition or table. Once data is committed it becomes immediately visible to all Hive queries initiated subsequently. This API is intended for streaming clients such as Storm, which continuously generate data. Streaming support is built on top of cloud based insert/update support in Hive. So using C-HIVE query approach in Hadoop Framework. It can be constructed as NO SQL query. Query Plan Compiler and Query Execution Library are implemented. The data are retrieved efficiently.

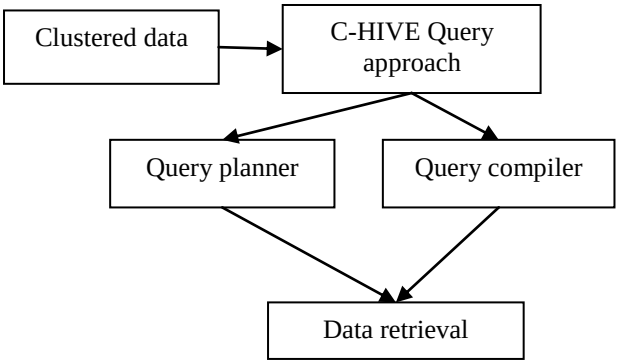

Figure 5: Data Retrieval Flow Chart

3.5 Evaluation criteria:

Many cloud computing service providers consider quality of service in terms of guaranteed bandwidth, dedicated hardware, system availability, and/or fault tolerance. QoS is generally considered in terms of guaranteed resource allocation (e.g., bandwidth, CPU utilization, memory, storage, etc.). In this module, minimize the communication cost of the query evaluation since the amounts of available bandwidth between different data centers varies over time and the communication cost is expensive due to large quantities of data transfers during the query evaluation. The query evaluation for big data analytics usually is both compute and bandwidth intensive, the computing resource in data centers and the communication bandwidth resource on links between inter-data centers must meet the query resource demands.

4. EXPERIMENTAL RESULTS

In distributed environment the maximum bandwidth reduction depends on the query and the properties of event streams. Chive produces maximum bandwidth reduction when compared to other query languages. Performance of the query

execution library is based on the amount of historical data stored during continuous query processing. Chive provides automatic parallelization for increasing the performance of query execution engine based on the hints provided in the query. By executing queries in parallel manner it reduces bandwidth consumption and increase the throughput in distributed environment.

5. CONCLUSION

Distributed cloud computing, also known as on-demand computing, is a kind of Internet-based computing, where shared resources, data and information are provided to computers and other devices on-demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources. Event stream processing is very difficult to analyze continuous streams of data. Continuous streams need NOSQL database. And ordered evaluation of continuous queries over data streams is crucial in stream processing systems. In this project, we studied the problem of providing continuous execution of window joins over asynchronous data streams. We showed that the C-HIVE approach that enforces ordered processing of input tuples to guarantee ordered output can result in increased response time and reduce the bandwidth. We can use the CHIVE query planner and compiler to minimize the bandwidth.

REFERENCES

[1] D. Peng and F. Dabek, "*Large-scale incremental processing using distributed transactions and notifications*," in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. USENIX Association, 2010, pp. 1–15.

[2] F. Frattini, K. S. Trivedi, F. Longo, S. Russo, and R. Ghosh, "*Scalable analytics for iaas cloud availability*," IEEE Transactions on Cloud Computing, vol. 2, no. 1, pp. 57–70, 2014.

[3] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Z. 0002, S. Anthony, H. Liu, and R. Murthy, "*Hive - a petabyte scale data warehouse using hadoop*," in Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA. IEEE, 2010, pp. 996–1005.

[4] S. Babu and J. Widom, "*Continuous queries over data streams*," SIGMOD Rec., vol. 30, no. 3, pp. 109–120, Sep. 2001.

[5] L. Mai, L. Rupprecht, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "*Supporting application-specific in-network processing in data centres*," in Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 519–520.

[6] I. Satoh, "*Mapreduce processing on iot clouds*," 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, vol. 1, pp. 323–330, 2013.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[8] M. Hayes and S. Shah, "*Hourglass: A library for incremental processing on hadoop*," in Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA, 2013, pp. 742–752.

[9] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P.Wyckoff, and R. Murthy, "*Hive: A warehousing solution over a map-reduce framework*," Proc. VLDB Endow., vol. 2, no. 2, pp. 1626–1629, Aug. 2009.

[10] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi, "*Query optimization for massively parallel data processing*," in Proceedings of the 2Nd ACM Symposium on Cloud Computing. ACM, 2011, pp. 12:1–12:13.

[11] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul,Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in In CIDR, 2005, pp.277–289.

[12] N. Marz. Trident tutorial. [Online]. Availablehttps://github.com/nathanmarz/storm/wiki/Trident-tutorial

[13] P. Nathan, Enterprise Data Workflows with Cascading, 1st ed.O'Reilly Media, Inc., 2013.

[14] Cascading. [Online]. Available:http://www.cascading.org

[15] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in Proceedings of the 7thUSENIX Conference on Networked Systems Design and Implementation,ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010,pp. 21–21.