International Journal of Research
Available at https://edupediapublications.org/journals

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue 10
September 2017

SQL-Style Processing Geo-Distributed Framework

Swetha Koduri & T. Satya kiranmai
[1]Assistant Professor, Information Technology, Malla Reddy College of Engineering and Technology, Maisammaguda, Dhulapalli,Telangana
[2]Assistant Professor, Computer science and engineering CMR College of Engineering and Technology, Kandlakoya, Medchal, Telangana.
koduriswetha@gmail.com & tadepallikiranmai84@gmail.com

Abstract—*Hadoop is widely used distributed processing frameworks for large-scale data processing in an efficient and fault-tolerant manner on private or public clouds. These big-data processing systems are extensively used by many industries, e.g., Google, Facebook, and Amazon, for solving a large class of problems, e.g., search, clustering, log analysis, different types of join operations. However, all these popular systems have a major drawback in terms of locally distributed computations, which prevent them in implementing geographically distributed data processing. The increasing amount of geographically distributed massive data is pushing industries and academia to rethink the current big-data processing systems. The novel frameworks, which will be beyond state-of-the-art architectures and technologies involved in the current system, are expected to process geographically distributed data at their locations without moving entire raw datasets to a single location. In this paper, we investigate and discuss challenges and requirements in designing geographically distributed data processing frameworks and protocols. We classify and study Map Reduce-based system and SQL-style processing geo-distributed frameworks, models, and algorithms with their overhead issues.*

INTRODUCTION

In contrast, in the present time, data is generated geodistributively at a much higher speed as compared to the existing data transfer speed; for example, data from modern satellites. There are two common reasons for having geo-distributed data, as follows: (i) many organizations operate in different countries and hold datacenters (DCs) across the globe. Moreover, the data can be distributed across different systems and locations even in the same country, for instance, branches of a bank in the same country. (ii) Organizations may prefer to use multiple public and/or private clouds to increase reliability, security, and processing. In addition, there are several applications and computations that process and analyze a huge amount of massively geo-distributed data to provide the final output. For example, a bioinformatic application that analyzes existing genomes in different labs and countries to track the sources.

Geo-distributed databases and systems have been in existence for a long time. However, these systems are not highly fault tolerant, scalable, flexible, good enough for massively parallel processing, and simple to program, able to process a large-scale data, and fast in answering a query.

On a **positive side**, several big-data processing programming models and frameworks such as Map-Reduce, Hadoop, and Spark have been designed to overcome the disadvantages (e.g., fault-tolerance, unstructured/massive data processing, or slow processing time) of parallel computing, distributed databases, and cluster computing. Thus, this survey paper focuses on the Map-Reduce, Hadoop, and Spark based systems.

On a **negative side**, these frameworks do not regard geo-distributed data locations, and hence, they follow a trivial solution for geo-distributed data processing: copy all raw data to one location before executing a locally distributed computation.

In this work, we will review several models, frameworks, and resource allocation algorithms for geo distributed big-data processing that try to solve the abovementioned problems. In a nutshell, geo-distributed big-data processing frameworks have the following properties:

Ubiquitous computing: The new system should regard different data locations, and it should process data at different locations, transparent to users. In other words, new geo-distributed systems will execute a geo-computation like a locally distributed computation on geo-locations and support any type of big-data processing frameworks, languages, and storage media at different locations.

Data transfer among multiple DCs: The new system should allow moving only the desired data, which eventually participate in the final output in a secure and privacy preserving manner among DCs, thereby reducing the need for high bandwidth.

High level of fault-tolerance: Storing and processing data in a single DC may not be fault-tolerant when the DC crashes. The new system should also allow data replication from one DC to different trusted DCs, resulting in a higher level of fault-tolerance.

Advantages of geo-distributed data processing:- The main advantages of geo-distributed big-data processing are given and listed below:

• A geo-distributed Hadoop based system can perform data processing across nodes of multiple clusters while the standard Hadoop/Spark and their variants cannot process data at multiple clusters.
• More flexible services, e.g., resource sharing, load balancing, fault-tolerance, performance isolation, data isolation, and version isolation, can be achieved when a cluster is a part of a geo-distributed cluster.
• A cluster can be scaled dynamically during the execution of a geo-distributed computation.
• The computation cost can be optimized by selecting different types of virtual nodes in clouds according to the user requirement and transferring a job to multiple clouds.

Iterative and SQL queries

The standard Map-Reduce was not developed for supporting iterative and a wide range of SQL queries. Hive, Pig and Spark SQL were developed for supporting SQL-style queries. However, all these languages are designed for processing in-home/local data. Since relational algebra is a basis of several different operations, it is required to develop a geo-distributed query language regarding data locality and the network bandwidth. The new type of query language must also deal with some additional challenges such as geo-distributed query optimization, geo-distributed query execution plan, geo-distributed indexing, and geo-distributed caching. The problem of joining of multiple tables that are located at different locations is also not trivial.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue 10
September 2017

In this case, moving an entire table from one location to the location of the other table is naive yet cumbersome, because of network bandwidth, time, and cost. Hence, we see the joining operation in geo-distributed settings is a major challenge. The joining operation gets more complicated in the case of streaming of tables where a window-based join does not work because the joining values of multiple tables may not synchronously arrive at an identical time window, thereby leading to missing outputs. Processing iterative queries on the classical Hadoop was a cumbersome task due to disk-based storage after each iteration. However, Spark can efficiently process iterative queries due to in-memory processing. Processing iterative queries in a geo-computation requires us to find solutions to store intermediate results in the context of an iterative query.

SQL-style processing framework for pre-located geo-distributed Data

A central command layer, pseudo-distributed measurement of data transfer, and a workload optimizer. The main component of Geode is the central command layer that receives SQL analytical queries from the user, partitions queries to create a distributed query execution plan, executes this plan over involving DCs, coordinates data transfers between DCs, and collates the final output. At each DC, the command layer interacts with a thin proxy layer that facilitates data transfers between DCs and manages a local cache of intermediate query results used for data transfer optimization. The workload optimizer estimates the current query plan or the data replication strategy against periodically obtained measurements from the command layer. These measurements are collected using the pseudo distributed execution technique. Geode is built on top of Hive and uses less bandwidth than centralized analytics in a Microsoft production workload, TPC-CH, and Big Data Benchmark.

Pros. Geode performs analytical queries locally at the data site. Also, Geode provides a caching mechanism for storing intermediate results and computing differences for avoiding redundant transfers. The caching mechanism reduces the data transfer for the given queries by 3.5 times.

Cons. Geode does not focus on the job completion time and iterative machine learning workflows.

SQL-style processing framework for user-located geo-distributed data

A globally distributed data management system. Database aspects, e.g., distributed query execution in the presence of sharding / re-sharding, query restarts upon transient failures, and range/index extraction, of Spanner are discussed. In Spanner, table interleaving is used to keep tables in the database, i.e., rows of two tables that will join based on a joining attribute are kept co-located, and then, tables are partitioned based on the key. Each partition is called a shard that is replicated to multiple locations. A new type of operation is introduced, called Distributed Union that fetches results from the entire shard according to a query. However, performing the distributed union before executing any other operations, e.g., scan, filter, group by, join and top-k, will cause to read multiple shards, which may not participate in the final output. Hence, all such operators are pushed to the table before the distributed union, which takes place at the end to provide the final answer. Three different mechanisms of index or range retrieval are given, as follows: distribution range extraction, seek range extraction, and lock range extraction. A recent paper carries the same flavor of the hybrid cloud computation, suggests a general framework for executing SQL queries, specifically, select, project, join, aggregation, maximum, and minimum, while not revealing any sensitive data to the public cloud during the computation.

In the context of geo-data processing, data locality refers to data processing at the same site or nearby sites where the data is located. However, the current Hadoop/Spark/SQL-style based geo distributed data processing systems are designed on the principle of data pulling from all the locations to a single location, and hence, they do not regard data locality . In addition, due to a huge amount of raw data generated at different sites, it is challenging to send the whole dataset to a single location; hence, the design and development of systems that take the data locality into account are crucial for optimizing the system performance. In contrast, sometimes a framework regarding the data locality does not work well in terms of performance and cost, due to the limited number of resources or slow inter-DC connections. Hence, it may be required to access/process data in nearby DCs, which may be faster than the local access. Thus, we find a challenge in designing a system for accessing local or remote (nearby) data, leading to optimized job performance.

Scheduling In Geo-Distributed Systems

We present some methods/architectures that preprocess a job before deploying it over distributed locations to find the best way for data distribution and/or the best node for the computation. The main idea of the following methods is in reducing the total amount of data transfer among DCs. Note that the following methods work offline and do not provide a way for executing a geo distributed job on top of Hadoop/Spark, unlike systems in §4.2 that execute a job and may handle such offline tasks too.

Scheduling for Geo-distributed Map-Reduce-based Systems:-

WANalytics preprocesses a Map-Reduce job before its real implementation and consists of two main components, as follows: Runtime analyzer: executes user’s job directed acyclic graph (DAG), which is about job execution flow, in a distributed way across DCs. The runtime analyzer finds a physical plan that specifies where does each stage of the job to be executed and how will data be transferred across DCs. The runtime layer consists of a centralized coordinator, only with one DC that interacts with all the other DCs. Users submit a DAG of jobs to the coordinator that asks the workload analyzer to provide a physical distributed execution plan for the DAG.

Workload analyzer: continuously monitors and optimizes the user’s DAG and finds a distributed physical plan according to the DAG. The plan is determined in a manner that minimizes the total bandwidth usage by considering DC locations and data replication factor.

Cons. Unlike Iridium, WANalytics does not consider the network bandwidth and job latency, and only focuses on the amount of data transfer among DCs. In addition, WANalytics is not designed to handle iterative machine learning workflows.

Shuffle-aware data pushing at the map phase. It finds all those mappers that affect the job completion in a DC, and hence, rejects those mappers for a new job. In other words the algorithm selects only mappers that can execute a job and shuffle the intermediate data under a time constraint. Mappers are selected based on monitoring the most recent jobs. The algorithm is presented for a single DC and can be extended to geo-distributed settings. It is assumed that the same mappers have appeared in previous jobs; otherwise, it is hard to have a prior knowledge of mappers.

CONCLUSION

The classical parallel computing systems cannot efficiently process a huge amount of massive data, because of fewer resiliencies to faults and limited scalability of systems. Map-Reduce, developed by Google, provide efficient, fault-tolerant, and scalable large-scale data processing at a single site. Hadoop and Spark were not designed for on-site geographically distributed data processing; hence, all the sites send their raw data to a single site before a computation proceeds. In this survey, we discussed requirements and challenges in designing geo-distributed data processing using Map-Reduce and Spark. Hadoop and HMR require a global reducer at a pre-defined location. However, the selection of a global reducer has been considered separately while it directly affects the job completion time. Hence, a global reducer may be selected dynamically while respecting several real-time parameters. Though not each site sends its complete datasets, there still exists open questions to deal with, e.g., should all the DCs send their outputs to a single DC or to multiple DCs that eventually converge, should a DC send its complete output to a single DC or partition its outputs and send them to multiple DCs, and what are the parameters to select a DC to send outputs.

The existing work proposes frameworks that allow a limited set of operations. However, it is necessary to find answers to the following question: how to perform many operations like the standard Map-Reduce on geographically distributed Map-Reduce based framework. Also, we did not find a system that can process secure SQL-queries on geo-distributed data, but they focus on the hybrid cloud and store a significant amount of non-sensitive data in the private cloud too. Most reviewed frameworks do not deal with the job completion time. In a geo-distributed computation, the job completion time is affected by distance and the network bandwidth among DCs, the outputs at each DC, and the type of applications. However, there is no other framework that jointly optimizes job completion time and inter-DC transfer while regarding variable network bandwidth. Thus, there is a need to design a framework that optimizes several real-time parameters and focuses on the job completion time. In addition, the system must dynamically learn and decide whether the phase-to-phase or the end-to-end job completion time is crucial? Answering this question may also require us to find straggling mappers or reducers in the partial or entire computation. We also discussed critical limitations of using Hadoop and Spark in geo-distributed data processing. We can conclude that geo-distributed big-data processing is highly dependent on the following five factors: task assignment, data locality, data movement, network bandwidth, and security and privacy.

REFERENCES

[1] A. Jonathan and et al., “Aswan: Locality-aware resource manager for geo-distributed

data-intensive applications," in IC2E, 2016, pp. 32–41.
[2] A. Vulimiri and et al., "WANalytics: analytics for a geo-distributed data-intensive world," in CIDR, 2015.
[3] D. A. Reed and J. Dungaree, "Exascale computing and big data," Common. ACM, vol. 58, no. 7, pp. 56–68, 2015.
[4] K. A. Harwich and et al., "Distributed frameworks and parallel algorithms for processing large-scale geographic data," Parallel Computing, vol. 29, no. 10, pp. 1297–1333, 2003.
[5] K. Clouds and et al., "PIXIDA: optimizing data parallel jobs in wide-area data analytics," PVLDB, vol. 9, no. 2, pp. 72–83, 2015.
[6] http://www.computerworld.com/article/2834193/cloud-computing/5-tips-for-building-a-successful-hybrid-cloud.html.
[7] http://www.computerworld.com/article/2834193/cloud-computing/5-tips-for-building-a-successful-hybrid-cloud.html.
[8] R. Tudor an and et al., "Bridging data in the clouds: An environment-aware system for geographically distributed data transfers," in Sigrid, 2014, pp. 92–101.
[9] R. Tudor an, G. Antonio, and L. Bough, "SAGE: geo-distributed ´ streaming data analysis in clouds," in IPDPS Workshops, 2013, pp. 2278–2281.
[10] Q. Up and et al., "Low latency geo-distributed data analytics," in SIGCOMM, 2015, pp. 421–434.
[11] Q. Zhang and et al., "Improving Hadoop service provisioning in a geographically distributed cloud," in IEEE Cloud, 2014, pp. 432–439.
[12] A. Rabin, M. Are, S. Sen., V. S. Pay, and M. J. Freedman, "Making every bit count in wide-area analytics," in Hoots, 2013.
[13] M. Cardoso and et al., "Exploring Map Reduce efficiency with highly-distributed data," in Proceedings of the Second International Workshop on Map Reduce and Its Applications, 2011, pp. 27–34.
[14] B. Heinz, A. Chandra, R. K. Sitar man, and J. B. Weismann, "End to-end optimization for geo-distributed Map Reduce," IEEE Trans. Cloud Computing, vol. 4, no. 3, pp. 293–306, 2016.
[15] B. Tang, H. He, and G. Freda, "Hybrid: a new approach for hybrid Map Reduce combining desktop grid and cloud infrastructures," Concurrency and Computation: Practice and Experience, vol. 27,
no. 16, pp. 4140–4155, 2015.
[16] L. Wang and et al., "G-Hadoop: Map Reduce across distributed data centers for data-intensive computing," FGCS, vol. 29, no. 3, pp. 739–750, 2013.
[17] A. P. Sheath and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," ACM Compute. Surd., vol. 22, no. 3, pp. 183–236, 1990.
[18] J. Dean and S. Ghemawat, "Map Reduce: Simplified data processing on large clusters," in OSDI, 2004, pp. 137–150.
[19] Apache Hadoop. Available at: http://hadoop.apache.org/.
[20] M. Zaharias and et al., "Spark: Cluster computing with working
sets," in Hot Cloud, 2010.
[21] M. Izard and et al., "Dryad: distributed data-parallel programs from sequential building blocks," in Neurosis, 2007, pp. 59–72.

[22] G. Malefic and et al., “Pregl: a system for large-scale graph processing,” in SIGMOD, 2010, pp. 135–146.
[23] Apache Graph. Available at: http://giraph.apache.org/.
[24] C. Jay lath, J. J. Stephen, and P. Easter, “From the cloud to the atmosphere: Running Map Reduce across data centers,” IEEE Trans. Computers, vol. 63, no. 1, pp. 74–87, 2014.
[25] H. Garden, I. Rodeo, and M. Parishes, “Investigating Map Reduce framework extensions for efficient processing of geographically scattered datasets,” SIGMETRICS Performance Evaluation Review, vol. 39, no. 3, pp. 116–118, 2011.