ISSN 2395-695X (Print)
ISSN 2395-695X (Online)

International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)

Vol. 2, Special Issue 10, March 2016

SECURED BIG DATA MINING SCHEME FOR CLOUD ENVIRONMENT

Ms. K.S. Keerthiga, IInd ME., Ms. C.Sudha M.E., Asst. Professor/CSE
Surya Engineering College, Erode, Tamilnadu, India
keerthigaks@gmail.com

Abstract

Big data sharing operations are handled using the resources provided in the cloud computing environment. Cloud environment is divided into two categories based on the user privilege levels. They are public cloud and private cloud environment. Public cloud environment provides resources for the all the users. Private cloud resources are provided to a group of people only. Big data can be used in the health care, scientific and industrial applications.

Big data are provided in the private cloud environment. Services are provided under the public cloud environment. Cross cloud environment is constructed using the public and private cloud resources. Service selection methods are adapted to identify the suitable service provider from the public cloud environment for the private cloud big data values. Service provider selection is achieved using History record based Service optimization method (HireSome-II). Service selection operations are carried out with the support of the history records. HireSome-II scheme provides privacy ensured service selection support.

Big data mining operations are integrated with the History record based Service optimization method (HireSome-II). The mining operations are performed with security and privacy features. Scalability is supported with privacy ensured MapReduce scheme. Big data classification is performed under the big data mining process.

1. Introduction

Processing large datasets has become crucial in research and business environments. Practitioners demand tools to quickly process increasingly larger amounts of data and businesses demand new solutions for data warehousing and business intelligence. Big data processing engines have experienced a huge growth. One of the main challenges associated with processing large datasets is the vast infrastructure required to store and process the data. Coping with the forecast peak workloads would demand large up-front investments in infrastructure. Cloud computing presents the possibility of having a large-scale on demand infrastructure that accommodates varying workloads.

Traditionally, the main technique for data crunching was to move the data to the computational nodes, which were shared. The scale of today's datasets has reverted this trend and led to move the computation to the location where data are stored. This strategy is followed by popular MapReduce implementations. These systems assume that data is available at the machines that will process it, as data is stored in a distributed file system such as GFS, or HDFS. This situation is no longer true for big data deployments on the cloud. Newly provisioned VMs need to contain the data that will be processed.

2. Related Works

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)

International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)

Vol. 2, Special Issue 10, March 2016

A number of distributed frameworks have recently emerged for big data processing [5], [6], [7]. We discuss the frameworks that improve MapReduce. HaLoop, a modified version of Hadoop, improves the efficiency of iterative computation by making the task scheduler loop-aware and by employing caching mechanisms. Twister [9] employs a lightweight iterative MapReduce runtime system by logically constructing a Reduce-to-Map loop. iMapReduce [10] supports iterative processing by directly passing the Reduce outputs to Map and by distinguishing variant state data from the static data. i2MapReduce improves upon these previous proposals by supporting an efficient general-purpose iterative model. The above MapReduce-based systems, Spark [2] uses a new programming model that is optimized for memory- resident read-only objects. Spark will produce a large amount of intermediate data in memory during iterative computation. When input is small, Spark exhibits much better performance than Hadoop because of in memory processing. Its performance suffers when input and intermediate data cannot fit into memory. We experimentally compare Spark and i2MapReduce in [11] and see that i2MapReduce achieves better performance when input data is large.

Pregel follows the Bulk Synchronous Processing (BSP) model. The computation is broken down into a sequence of supersteps. In each superstep, a Compute function is invoked on each vertex. It communicates with other vertices by sending and receiving messages and performs computation for the current vertex. This model can efficiently support a large number of iterative graph algorithms. Open source implementations of Pregel include Giraph, Hama [12] and Pregelix [13]. Compared to i2MapReduce, the BSP model in Pregel is quite different from the MapReduce programming paradigm. It would be interesting future work to exploit similar ideas in this paper to support incremental processing in Pregel-like systems.

Besides Incoop [15], several recent studies aim at supporting incremental processing for one-step applications. Stateful Bulk Processing addresses the need for stateful dataflow programs. It provides a groupwise processing operator Translate that takes state as an explicit input to support incremental analysis. But it adopts a new programming model that is very different from MapReduce. In addition, several research studies [1] support incremental processing by task-level re-computation, but they require users to manipulate the states on their own. In contrast, i2MapReduce exploits a fine-grain kv-pair level re-computation that are more advantageous. Naiad [14] proposes a timely dataflow paradigm that allows stateful computation and arbitrary nested iterations. To support incremental iterative computation, programmers have to completely rewrite their MapReduce programs for Naiad. In comparison, we extend the widely used MapReduce model for incremental iterative computation. Existing Map- Reduce programs can be slightly changed to run on i2MapReduce for incremental processing.

3. Hadoop for MapReduce

Mapreduce has emerged as a popular and easy-to-use programming model for large-scale data analytics in data centers. It is an important application for numerous organizations to process explosive amounts of data, perform massive computation and extract critical knowledge out of big data for business intelligence. The efficiency of MapReduce performance and scalability can directly affect our society's ability to mine knowledge out of raw data [4]. In addition, energy consumption accounts for a large portion of the operating cost of data centers in analyzing such big data. While business and scientific applications are increasingly relying on the MapReduce model, the energy efficiency of MapReduceis also critical for data centers' energy conservation.

Hadoop is an open-source implementation of MapReduce, currently maintained by the Apache Foundation and supported by leading IT companies such as Facebook and Yahoo!. It implements the MapReduce model by distributing user inputs as data splits across a large number of compute nodes. Hadoop uses a master program to command many TaskTrackers and schedule map tasks and reduce tasks to the TaskTrackers, A Hadoop program processes data through two main functions. The analytic functions are performed in two phases: mapping and reducing. In the mapping phase, the input dataset of a program is divided into many data splits. Each split is organized as many records of key and value (< >) pairs. One MapTask is launched per data split to convert the original records into intermediate data in the form of many ordered < k',v' > pairs. These ordered records are stored as a MOF (Map Output File) split. A MOF is organized into many data partitions, one per ReduceTask. In the reducing phase, each ReduceTask applies the reduce function to its own share of data partitions (a.k.a segments).

Between the mapping and reducing phases, a ReduceTask needs to fetch a segment of the intermediate output from all finished MapTasks. Globally, this leads to a shuffling of intermediate data from MapTasks to Reduce- Tasks. For data-intensive MapReduce programs such as TeraSort, data shuffling can add a significant number of disk accesses, contending for the limited I/O bandwidth. In order to elaborate this problem, we conduct a data-intensive MapReduce test, where we run 200 GB TeraSort on 10 slave nodes. We have examined the wait time and the service time of I/O requests during the execution. The wait time can be more than 1,100 milliseconds. Worse yet, most I/O requests are spending close to 100% of this time waiting in the queue, suggesting that the disk is not able to keep up with the requests. The shuffling of intermediate data competes for disk bandwidth with MapTasks. This can significantly overload the disk subsystem. It results in a serious bottleneck along with the severe disk I/O contention in data-intensive MapReduce programs, which entails further research on efficient data shuffling techniques. Although a number of recent efforts have investigated data locality of MapReduce by either preventing stragglers [3] or applying high performance interconnects to transfer data in large-scale Hadoop cluster [8], few studies have addressed the need of efficient I/O during data shuffling in the Hadoop MapReduce framework. Condie et al. have proposed a MapReduce online architecture to open up direct network channels between MapTasks and ReduceTasks and speed up the delivery of data from MapTasks to ReduceTasks. While their work reduces job completion time and improves system utilization, it cannot accommodate a gigantic dataset that does not fit in memory and also complicates the fault tolerance handling of Hadoop tasks. It often falls back to spilling data to the disks for large data sets. Our prior work has offered a network-levitated merging scheme to keep the data shuffling phase above disks. But the aggressive use of memory buffers in network-levitated merging makes it unable to cope with MapReduce jobs with tens of thousands or even millions of data splits.

4. Big Data Sharing under Cloud Environment

Cloud Computing and big data receives enormous attention internationally due to various business-driven promises and expectations such as lower upfront IT costs, a faster time to market and opportunities for creating value-add business. As the latest computing paradigm, cloud is characterized by delivering hardware and software resources as virtualized services by which

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)

International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)

Vol. 2, Special Issue 10, March 2016

users are free from the burden of acquiring the low level system administration details. Cloud computing promises a scalable infrastructure for processing big data applications such as the analysis of huge amount of medical data. Cloud providers including Amazon Web Services (AWS), Sales force. com, or Google App Engine, give users the options to deploy their application over a network of a nearly infinite resource pool. By leveraging Cloud services to host Web, big data applications can benefit from cloud advantages such as elasticity, pay-per-use and abundance of resources with practically no capital investment and modest operating cost proportional.

To satisfy different security and privacy requirements, cloud environments usually consist of public clouds, private clouds and hybrid clouds, which lead a rich ecosystem in big data applications. Current implementations of public clouds mainly focus on providing easily scaled up and scaled-down computing power and storage. If data centers or domain specific services center tend to avoid or delay migrations of themselves to the public cloud due to multiple hurdles, from risks and costs to security issues and service level expectations, they often provide their services in the form of private cloud or local service host. For a complex web-based application, it probably covers some public clouds, private clouds or some local service host. For instance, the healthcare cloud service, a big data application involves many participants like governments, hospitals, pharmaceutical research centres and end users. As a result, a healthcare application often covers a series of services respectively derived from public cloud, private cloud and local host.

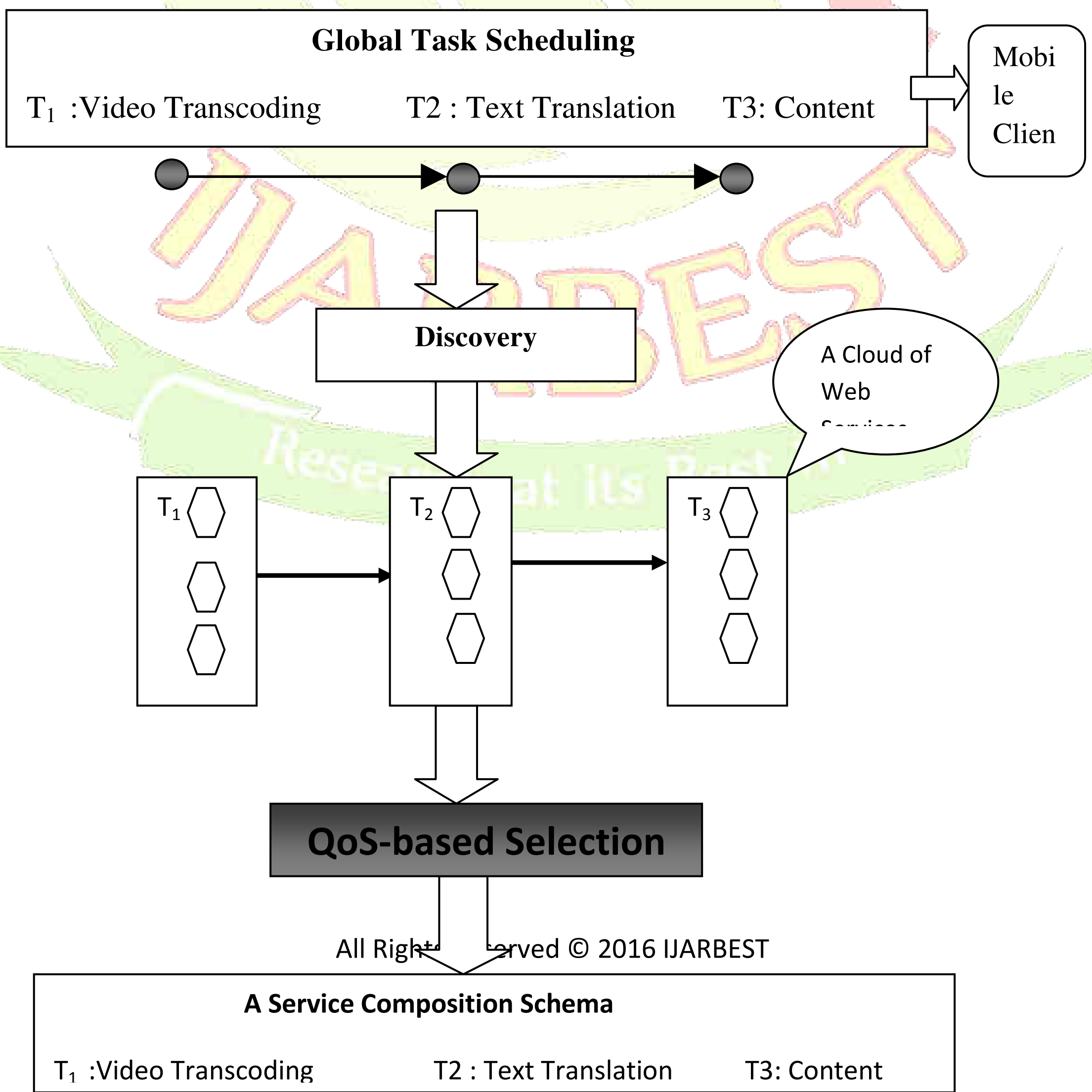

Figure 4.1.: Cross Cloud Service Composition Scenario

Some big data centers or software services cannot be migrated into a public cloud due to some security and privacy issues. If a web based application covers some public cloud services, private cloud services and local web services in a hybrid way, cross-cloud collaboration is an ambition for promoting complex web based applications in the form of dynamic alliance for value-add applications. It needs a unique distributed computing model in a network-aware business context. Cross-cloud service composition provides a concrete approach capable for large-scale big data processing. Existing analysis techniques for service composition, often mandate every participant service provider to unveil the details of services for network-aware service composition, especially the QoS information of the services. Unfortunately, such an analysis is infeasible when a private cloud or a local host refuses to disclose all its service in detail for privacy or business reasons. In such a scenario, it is a challenge to integrate services from a private cloud or local host with public cloud services such as Amazon EC2 and SQS for building scalable and secure systems in the form of mashups. As the diversity of Cloud services is highly available today, the complexity of potential cross-cloud compositions requires new composition and aggregation models.

As a cloud often hosts a lot of individual services, cross cloud and on-line service composition is heavily time-consuming for big data applications. It always challenges the efficiency of service composition development on Internet. Besides, for a web service which is not a cloud service and its bandwidth probably fails to match to the cloud, it is a challenge to trade off the bandwidth between the web service and the cloud in a scaled-up or scaled-down way for a cross-cloud composition application. The time cost is heavy for cross-platform service composition. With these observations, it is a challenge to tradeoff the privacy and the time cost in cross cloud service composition for processing big data applications. In view of this challenge, an enhanced History record-based Service optimization method named HireSome-II, is presented in this paper for privacy-aware cross-cloud service composition for big data applications. In our previous work, a similar method, named HireSome aims at enhancing the credibility of service composition. HireSome-I is incapable of dealing with the privacy issue in cross-cloud service composition. Compared to HireSome-I, HireSome-II greatly speeds up the process of selecting a optimal service composition plan and protects the privacy of a cloud service for cross-cloud service composition.

Cloud computing environment provides scalable infrastructure for big data applications. Cross clouds are formed with the private cloud data resources and public cloud service components. Cross cloud service composition provides a concrete approach capable for large scale big data processing. Private clouds refuse to disclose all details of their service transaction records. History record based Service optimization method (HireSome-II) is privacy aware cross cloud service composition method. QoS history records are used to estimate the cross cloud service composition plan. k-means algorithm is used as a data filtering tool to select representative history records. HireSome-II reduces the time complexity of cross cloud service composition plan for big data processing. The following drawbacks are identified from the existing system. Big data processing is not integrated with the system.

5. Secured Big Data Mining Scheme

History record based Service optimization method (HireSome-II) is enhanced to process big data values. Security and privacy is provided for cross cloud service composition based big data processing environment. Privacy preserved map reduce methods are adapted to support high scalability. The HireSome-II scheme is upgraded to support mining operations on big data.

Security and privacy preserved big data processing is performed under the cross cloud environment. Big data classification is carried out with the support of map reduce mechanism. Service composition methods are used to assign resources. The system is divided into six major modules. They are Cross Cloud Construction, Big Data Management, History Analysis, Map Reduce Process, Service Composition and Big Data Classification.

Public and private clouds integrated in the cross cloud construction process. Big data management module is designed to provide big data for the cloud users. Resource sharing logs are analyzed under the history analysis. Task partition operations are performed under the map reduce process. Service provider selection is carried out service composition module. Classification process is carried out under the cross cloud environment.

5.1. Cross Cloud Construction

Private and public cloud resources are used in the cross cloud construction process. Big data values are provided under the data centers in private cloud environment. Service components are provided from public cloud environment. Public cloud services utilize the private cloud data values.

5.2. Big Data Management

Larger and complex data collections are referred as big data. Medical data values are represented in big data form. Anonymization techniques are used to protect sensitive attributes. Big data values are distributed with reference to the user request.

5.3. History Analysis

Service provider manages the access details in the history files. User name, data name, quantity and requested time details are maintained under the data center. History data values are released with privacy protection. Data aggregation is applied on the history data values.

5.4. Map Reduce Process

Map reduce techniques are applied to break the tasks. Map reduce operations are partitioned with security and privacy features. Redundancy and fault tolerance are controlled in the system. The data values are also summarized in the map reduce process.

5.5. Service Composition

HireSome-II scheme is adapted for the service composition process. History records are analyzed with K-means clustering algorithm. Privacy preserved data communication is employed in the system. Public cloud service components are provided to the big data process.

5.6. Big Data Classification

Medical data analysis is carried out on the cross cloud environment. Privacy preserved data classification is applied on the medical data values. Public cloud resources are allocated for the classification process. Bayesian algorithm is tuned to perform data classification on parallel and distributed environment.

6. Conclusion

Service composition methods are used to provide resources for big data process. History record based Service optimization method (HireSome-II) is used as privacy ensured service composition method. HireSome-II scheme is enhanced with privacy preserved big data process mechanism. Map reduce techniques are also integrated with the HireSome-II scheme to support high scalability. Security and privacy are provided for the big data and history data values under the cloud environment. Map reduce techniques reduces the computational complexity in big data processing. Data classification is performed on sensitive big data values with cloud resources. Efficient resource sharing is performed under cross cloud environment

References

[1] T. J€org, R. Parvizi, H. Yong and S. Dessloch, "Incremental recomputations in mapreduce," in Proc. 3rd Int. Workshop Cloud Data Manage., 2011, pp. 7–14.
[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for, in-memory cluster computing," in Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation, 2012, p. 2.
[3] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha and E. Harris, "Reining in the outliers in Map-Reduce clusters using Mantri," in Proc. 9th USENIX Symp. Oper. Syst. Design Implementation (OSDI), R.H. Arpaci-Dusseau and B. Chen, eds., Oct. 4–6, 2010, pp. 265–278.
[4] P. Balaji, S. Aameek, L. Ling and J. Bhushan, "Purlieus: Localityaware resource allocation for MapReduce in a cloud," in Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC'11), Nov. 2011, pp. 58:1–58:11.
[5] S. R. Mihaylov, Z. G. Ives and S. Guha, "Rex: Recursive, deltabased data centric computation," in Proc. VLDB Endowment, 2012, vol. 5, no. 11, pp. 1280 1291.
[6] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," in Proc. VLDB Endowment, 2012, vol. 5, no. 8, pp. 716–727.
[7] S. Ewen, K. Tzoumas, M. Kaufmann and V. Markl, "Spinning fast iterative data flows," in Proc. VLDB Endowment, 2012, vol. 5, no. 11, pp. 1268–1279.
[8] Y. Wang, X. Que, W. Yu, D. Goldenberg and D. Sehgal, "Hadoop acceleration through network levitated merge," in Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC'11), Nov. 2011, pp. 57:1–58:10.
[9] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu and G. Fox, "Twister: A runtime for iterative mapreduce," in Proc. 19th ACM Symp. High Performance Distributed Comput., 2010, pp. 810–818.
[10] Y. Zhang, Q. Gao, L. Gao and C. Wang, "imapreduce: A distributed computing framework for iterative computation," J. Grid Comput., vol. 10, no. 1, pp. 47–68, 2012.

[11] Y. Zhang, S. Chen, Q. Wang and G. Yu, "i2mapreduce: Incremental mapreduce for mining evolving big data," CoRR, vol. abs/ 1501.04854, 2015.
[12] Apache hama. [Online]. Available: https://hama.apache.org/, 2015.
[13] Y. Bu, V. Borkar, J. Jia, M. J. Carey and T. Condie, "Pregelix: Big (ger) graph analytics on a dataflow engine," in Proc. VLDB Endowmen, 2015, vol. 8, no. 2, pp. 161–172.
[14] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham and M. Abadi, "Naiad: A timely dataflow system," in Proc. 24th ACM Symp. Oper. Syst. Principles, 2013, pp. 439–455.
[15] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar and R. Pasquin, "Incoop: Mapreduce for incremental computations," in Proc. 2nd ACM Symp. Cloud Comput., 2011, pp. 7:1–7:14.