7th National Conference on
"Role of Engineers in Nation Building" organized by VIVA Institute of Technology, Mumbai
(1st and 2nd March 2019)
International Journal of Engineering Research & Science (IJOER)
ISBN: [978-93-5321-411-1] ISSN: [2395-6992] [Vol-5, Issue-3, March- 2019]

Cloud MapReduce

Monali Pimpale

Computer Engineering Department, VIVA Institute of Technology, Mumbai

Abstract**—** *Now a days Cloud computing has a great attention due to its provision of configurable computing resources. MapReduce is a framework for data-intensive distributed computing of batch jobs. MapReduce suffers from some drawbacks. We describe Cloud MapReduce to overcome the limitations of MapReduce. A cloud OS is responsible for managing the low level cloud resources and presenting a high level interface to the application programmers in order to hide the infrastructure details like a traditional Operating System. However, unlike a traditional Operating system, a cloud Operating system has to manage these resources at higher scale. Cloud OS has already taken on the complexity to make its services scalable, we describes Cloud MapReduce (CMR), which implements the MapReduce programming model on top of the cloud OS. CMR is a demonstration that it is possible to overcome the cloud limitations and simplify system design by building on top of a cloud OS.*

Keywords— Cloud Computing, Cloud Map Reduce, MapReduce, Operating System.

I. INTRODUCTION

Like a server Operating System, a cloud Operating System is intended for managing resources. In a server (e.g., a Personal Computer), the OS is responsible for managing the various hardware resources, such as CPU, memory, disks, network interfaces – everything inside a server. It hides the hardware operation details and allows these scarce or insufficient resources to be shared efficiently. A cloud OS is used for the same reason.

A cloud OS is responsible for managing the cloud infrastructure, hiding the cloud infrastructure details from the application programmers and coordinating the sharing of the limited resources instead of managing a single machine's resources. A cloud OS is much more complex, because it has to manage a much bigger infrastructure including that it has to serve many more customers. Today, very few of companies, such as Microsoft, Google, Amazon and Yahoo, need and are capable of building a cloud. For example, it is reported that Google has well over 1 million servers, and it serves millions of customers. Managing such big an infrastructure and supporting so many users require the Operating System to be extremely robust and scalable.

1.1 Cloud OS

A cloud OS is similar to a traditional server OS in terms of the services it provides, even though the underlying resources it manages are different.

i. It provides computer services, such as Amazon EC2 and Windows Azure workers. They provide computing power in the form of Virtual Machines instead of threads.

ii. It provides storage services, such as Amazon S3 and Windows Azure blob storage.

iii. A cloud OS provides communication services, such as Amazon's Simple Queue Service and Windows Azure queue service, which are similar to a pipe on a UNIX OS, in case of pipeline system a user push in messages at one end and pop out messages at the other end.

iv. A cloud OS also provides persistent storage services, such as Amazon's SimpleDB and Windows Azure table services. They provide persistent storage similar to the registry service in a Windows OS.

A key difference between server OS and cloud OS is that it is scalable. There are two reasons for cloud os being scalable. First reason is, a cloud OS has to manage a much bigger infrastructure. Secondly, a cloud has to support hundreds of thousands of people instead of just a few users on a PC. To meet this scalability challenges, cloud providers are forced to build new solutions from the scratch.

International Journal of Engineering Research & Science (IJOER)
ISBN: [978-93-5321-411-1] ISSN: [2395-6992] [Vol-5, Issue-3, March- 2019]

For example, Google designed their own GFS to manage files and their own Big Table to store a large amount of semi-structured data. On other hand Amazon have designed Dynamoto manage storage and amazon had built their own management infrastructure to support their web services API[2]. As cloud seems to be necessity cloud providers have spent large efforts in implementing the highly scalable cloud operating system which can manage large infrastructure which can be shared by many people.

1.2 Challenges posed by a cloud OS

A cloud Operating system's scalability comes at a price. It has to be traded off with other desirable system properties. The theorem states that, of the three properties of shared-data systems

i) Data Consistency,

ii) System Availability,

iii) Tolerance To Network Partition

Only two properties can be achieved at any given time theorem]. Because a cloud is used by thousands of people, it has to be highly scalable and always available; thus, the only property it can give up is data consistency.To overcome this limitations cloud MapReduce is implemented on top of the cloud.

1.3 Advantages of Cloud MapReduce

Cloud MapReduce has several highly desirable properties:

1.3.1 Incremental scalability:

Cloud MapReduce can scale incrementally in the number of computing nodes. A user can launch a number of servers at the beginning as well as launch additional servers in the middle of a computation if the user thinks the progress is too slow.

1.3.2 Symmetry and Decentralization:

Every node in Cloud MapReduce has the same responsibilities as its peers. There are not concept of master or slave nodes. Symmetry makes simple system provisioning, configuration and failure recovery.

1.3.3 Heterogeneity:

The computing nodes may have varying computation capacity. In that case the faster nodes can work more than the slower nodes. In addition to that the computing nodes could be distributed geographically.

II. THE ARCHITECTURE

The Cloud MapReduce architecture is shown in Fig. 1. The architecture is with the several SQS queues:

i) Input queue,

ii) Master reduce queue

iii) Output queue

iv) Reduce queues.

Input queue, master queue and output queue each with one queue and reduce in many forms. Input queue holds the inputs to the MapReduce computation. At the start of the computation, the user provides contains a list of S input key-value pairs to an input queue. Each key and value pair corresponds to a split of the input data that will be processed by one map task. For the purpose of tracking, each key-value pair also has a unique map ID. In the word count application, the input queue contains the document collections where the key is the document name and the value is a pointer into S3 storage. SQS is designed for the purpose of message communication, hence it has message size limitation of 8KB.

International Journal of Engineering Research & Science (IJOER)
ISBN: [978-93-5321-411-1] ISSN: [2395-6992] [Vol-5, Issue-3, March- 2019]

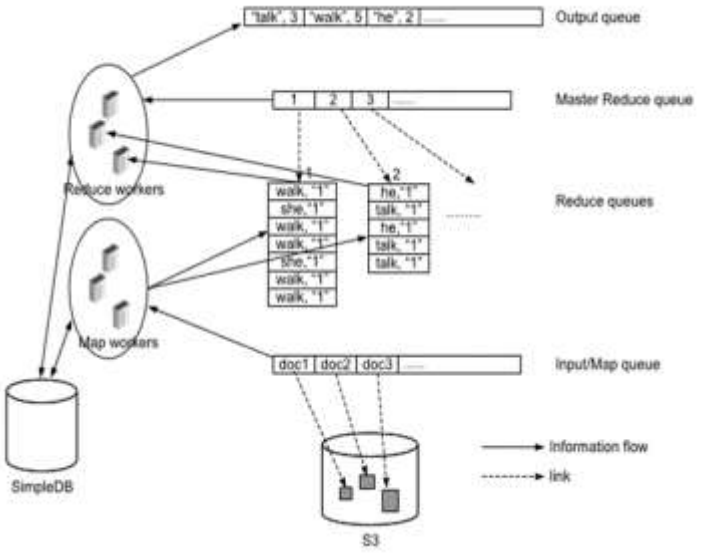

FIG 1: cloud MapReduce architecture

Input queue, master queue and output queue each with one queue and reduce in many forms. Input queue holds the inputs to the MapReduce computation. At the start of the computation, the user provides contains a list of S input key-value pairs to an input queue. Each key and value pair corresponds to a split of the input data that will be processed by one map task. For the purpose of tracking, each key-value pair also has a unique map ID. In the word count application, the input queue contains the document collections where the key is the document name and the value is a pointer into S3 storage. SQS is designed for the purpose of message communication, hence it has message size limitation of 8KB.

Instead of storing whole file data which will be too large to fit we store a pointer to large S3. Including to pointer to the file location in S3, it also contain a range specification which specifies the chunk of the file. By using range specification user could divide the file into small pats and can process the file part separately. The output queue is used to hold the results of the MapReduce computation. In the word count application, the output holds the resulting key-value pairs. In the architecture there is only one master reduce queue which holds the many pointers, one for each reduce queue. The master reduce queue is used to assign reduce tasks. There are large number of reduce queues. The number of them, denoted by Q, is a configurable parameter that is set by the user. The reduce queues and the master reduce queue, as well as the entries in the master reduce queue, are created distributed before the start of the MapReduce job.

In the word count e.g., the input value is a pointer to a document stored in S3. The map function first downloads the document from S3 on to the local machine. It then parses the document, and for each word (e.g., "talk") it sees, it emits a key-value pair, where the key is the word (e.g., "talk") and the value is simply "1" to indicate that it has seen this word once.

The MapReduce framework collects the intermediate key value pairs from the map function, then writes them to the reduce queues. A reduce key maps to one of the reduce queues through a hash function. A default hash function is provided, but the users could also supply their own.

CMR uses the network to transfer the intermediate key value pairs once they are available, thus it overlaps data shuffling with map processing. This technique is different from other implementations where the intermediate key-value pairs are only copied after a map task finishes. Overlapping and shuffling is used when pipelining MapReduce used. Compared to the implementation of pipelining where it has to implement pairwise socket connections and buffering/copying mechanism, our implementation using queues is much simpler. Since the map phase is typically long, overlap shuffling has the effect of spreading out traffic. This

International Journal of Engineering Research & Science (IJOER)
ISBN: [978-93-5321-411-1] ISSN: [2395-6992] [Vol-5, Issue-3, March- 2019]

can help alleviate the in cast problem (switch buffer overflow caused by simultaneous transfer of a large amount of data) if it occurs [3][4].

As soon as the map workers finish their jobs, the reduce workers start to poll work from the master reduce queue. Once a reduce worker dequeues a message, it is responsible for processing all data in the reduce queue indicated by the message. It dequeues messages from the reduce queue and feeds them into the user-defined reduce function as an iterator. After the reduce function finishes processing all data in the reduce queue, the worker goes back to the master reduce queue to fetch the next message to process. Just like in other MapReduce implementations, the user defined reduce function writes a set of key-value pairs as the outputs. The reduce workers collect the outputs and write them to the output queue. The name of the output queue has been specified before the start of the MapReduce job. It can be used either as the final output or as the input to the next MapReduce job. Besides reading from and writing to the various queues, the workers also read from and write to SimpleDB to update their status. By communicating status with a central scalable SimpleDB service, we not only avoid a single point bottleneck in our architecture, but we also make our implementation fully distributable. Workers work independently of all other workers and they do not care how many other workers are there. In addition, workers can be heterogeneous. They can be located anywhere in the world and they can have a vastly different computing capacity.

Even though we have shown two sets of workers i.e. map workers and reduce workers as shown in Fig. 1, both could run on the same set of computing nodes. Reduce workers start only after the map phase has finished. A new worker can join the computation at any time. When new worker joins, it can determine whether the map phase has finished or not ,by querying SimpleDB, and it can then poll the input queue or the master reduce queue accordingly for work.

In this architecture, it is easy for the job owner to get a rough sense of the job progress. The input queue length as a percentage of S. The original input queue length is a good approximation of the map progress. Similarly, the master reduces queue length as a percentage of Q. The original master reduce queue length is a good approximation of the reduce progress.

III. CONCLUSION

It is obvious why Cloud MapReduce is more scalable. Unlike other MapReduce, we have adopted a fully distributed architecture and we do not have a single master node as a bottleneck. The architecture uses queues to shuffle results from Map to Reduce. Mappers write results as soon as they are available and Reducers filter out results from failed nodes, as well as duplicate results.. CMR indicate that using queues to overlap the map and shuffling stage seems to be a promising approach to improve MapReduce performance.

REFERENCES

[1] Huan Liu, Dan Orban, "Cloud MapReduce: A MapReduce Implementation on Top of a Cloud Operating System," 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 23 - 26, 2011

[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo:amazon's highly available key-value store," SIGOPS Oper. Syst. Rev.,vol. 41, no. 6, pp. 205–220, 2007.

[3] A. Phanishayee, E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, G. Gibson, and S. Seshan, "Measurement and analysis of tcp throughput collapse in cluster-based storage systems," in Proc. File and Storage Technologies (FAST), Feb 2008.

[4] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, , D. Andersen, G. Ganger, G. Gibson, and B. Mueller, "Safe and effective fine grained tcp retransmissions for datacenter communication," in Proc. SIGCOMM, 2009.

[5] R. Griffith, Y. Chen, J. Liu, A. Joseph, and R. Katz, "Understanding tcp incast throughput collapse in datacenter networks," in Proc. SIGCOMM