2012 International Conference on Advanced Computer Science Applications and Technologies

An In-depth Study of Map Reduce in Cloud Environment

Novia Nurain
Computer Science and Engineering
United International University
novia@cse.uiu.ac.bd

Hasan Sarwar
Computer Science and Engineering
United International University
hsarwar@cse.uiu.ac.bd

Md.Pervez Sajjad
Computer Science and Engineering
United International University
pervez.cse@gmail.com

Moin Mostakim
Computer Science and Engineering
Bangladesh University of Engineering and Technology
shisshir11@gmail.com

***Abstract*—Cloud computing has introduced an utility computing model which offers an alternative to traditional servers and computing clusters. Due to fault tolerance and scalability feature, MapReduce distributed data processing architecture has now become the choice for data-intensive analysis in the clouds. Recently Mapruduce is used as a service or for setting up one's own Mapreduce cluster. In this paper we evaluate the architecture and performance of different MapReduce framework, such as AzureMapreduce built using the Microsoft Azure cloud infrastructure, Cloud MapReduce(CMR) built on top of Amazon cloud OS. We belief that the techniques we discussed in this paper are general enough that would encouraged others to use them in other applications. Our survey would definitely open an promising approach to improve MapReduce performance for Cloud Computing.**

***Keywords*-cloud computing; mapreduce; AzureMapReduce; CloudMapReduce**

I. Introduction

Today, scientific research is increasingly reliant on the processing of very large amounts of data. Scientists are more reliant on computing resources than ever and desire more power and greater ease of use. At the same time, we notice that the industryś introduction of the concepts of Cloud Computing [1] and MapReduce [3] gives scientists very viable alternatives for their computational needs. In fact, the utility computing model offered by cloud computing is well-suited. While clouds offer raw computing power combined with cloud infrastructure services offering storage and other services, there is a need for distributed computing frameworks to harness the power of clouds both easily and effectively. MapReduce framework model introduced by Google are well suited for the execution of large distributed job in a cloud infrastructure.

A. Cloud Computing

Cloud computing [4] is a new term for a long held dream of computing as a utility, which has recently emerged as commercial reality. Cloud Computing refers to both the application delivered as services over the Internet and the hardware and system software in the data centers that provide those services. This sudy focuses on SaaS models for public cloud. From hardware point of view three aspects are new for Cloud Computing.

- The illusion of infinite computing resources available on demand.
- The elimination of an up-front commitment by cloud users.
- The ability to pay for use of computing resources on a short term basis as needed.

B. MapReduce

The MapReduce [3] distributed data analysis framework model introduced by Google provides an easy-to-use programming model that features fault tolerance, automatic parallelization, scalability and data locality-based optimizations. Due to their excellent fault tolerance features, MapReduce frameworks are well-suited for the execution of large distributed jobs in brittle environments such as commodity clusters and cloud infrastructures. In brief, a map-reduce computation executes as follows: Some Map tasks are given one or more chunks from a distributed file system. Each of these Map tasks turns the chunk into a sequence of key-value pairs and these pairs are written to local disk as intermediate file, partitioned into R (number of Reduce tasks) regions by the partitioning function. The locations of these regions are passed back to the master, who is responsible for forwarding these locations to the Reduce tasks. Each of R Reduce tasks is responsible for one of these regions to apply reduce. So, all key-value pairs with the same key wind up at the same Reduce task. The Reduce tasks work on one key at a time, and combine all the values associated with that key in user defined way.

C. Overview

This paper is organized as follows. Section II and III describes the challenges for MapReduce in the Cloud and the performance factors that affect the performance of MapReduce. Section IV describes several MapReduce frameworks in cloud computing such as MapReduce on large

978-0-7695-4959-0/13 $25.00 © 2013 IEEE
DOI 10.1109/ACSAT.2012.70

cluster [2], Azure MapReduce [5], Cloud MapReduce [7]. Section V contains comparison among the architectures and performance. Section VI present future work. Finally we conclude in Section VII.

II. Challenges for MapReduce in the Cloud

As mentioned in the section I, MapReduce frameworks perform much better in brittle environments than other tightly coupled distributed programming frameworks, due to their excellent fault tolerance capabilities. However, cloud environments provide several challenges for MapReduce frameworks to harness the best performance.

- **Storage for Data** : Clouds offers a variety of storage options, such as off-instance cloud storage like Amazon S3, mountable off-instance block storage like Amazon EBS, visualized instance storage and many more. The choice of best suited storage to the particular MapReduce deployment plays an important role. Because the performance of data intensive applications rely a lot on the storage location and on the storage bandwidth.
- **Storage for metadata** : MapReduce frameworks also need to maintain metadata informations for ensuring good scalability and the accessibility to avoid single point of failures.
- **Communication consistency and scalability**: Cloud infrastructures have the tendency to exhibit inter-node I/O performance fluctuations(due to shared network, unknown topology) that could affect the intermediate data transfer performance of MapReduce applications.
- **Consistency of performance** : Clouds are implemented as shared infrastructures operating using virtual machines. Itś possible for the performance to fluctuate based on the load of the underlying infrastructure services.
- **Reliability** : Nodes failure are to be expected whenever large numbers of nodes are utilized for computations. MapReduce framework can recover jobs from worker node failure but it becomes disastrous when master node(node which stores metadata, which handles job scheduling queue etc) fails.
- **Choice of suitable instance type** : Its important for the performance of mapreduce job to select best instance in terms of both configuration and monetary terms as cloud offers a variety of them.

III. Performance Factors

This section focuses to answer the following one question:

- What factors affect the performance of Mapreduce?

To answer the question, we consider the impact of the architectural design of MapReduce, including programming model, storage-independent. In particular, we identify some factors that affect the performance of MapReduce from [2], [6] and [8] :

- **I/O mode** : While MapReduce is independent of the underlying storage system, it still requires the storage system to provide I/O interfaces for scanning data. We identify two types of I/O modes: direct I/O and streaming I/O. It has been seen that direct I/O outperforms streaming I/O by 10% [6] .
- **Scheduling** : The scheduling algorithm, which determines the assignment of map tasks to the available nodes affects performance of MapReduce. Existing block-level scheduling incurs considerable overhead in MapReduce. The more data blocks to schedule, the higher the cost the scheduler will incur [9].
- **Fault tolerancy** : The performance of MapReduce framework depends highly on its fault tolerance capability. The architecture should be resilient to any kind of failure.

IV. Survey on Different MapReduce framework for Cloud Computing

MapReduce was first proposed by Google in 2003 to cope with the challenges of processing an exponentially growing amount of data. In the same year the technology was invented, Google's production index system was converted to MapReduce.

A. MapReduce architecture for Google Large Clusters

1) Execution Overview: When a user program calls the MapReduce function [2] the following sequence of actions occurs:

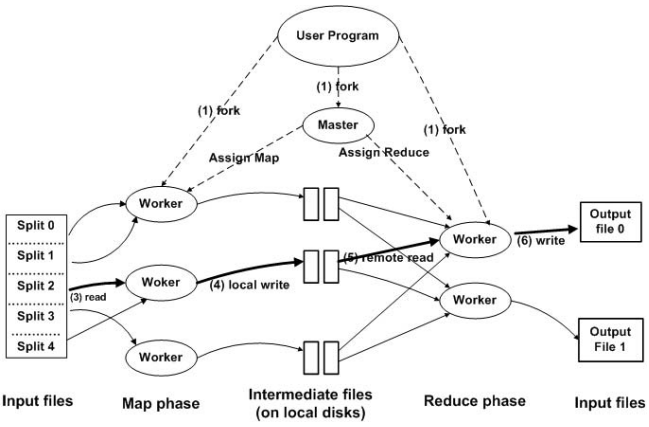

Figure 1. Execution Overview [2]

- When user invoke the MapReduce function, the MapReduce library in the user program first splits the input files into pieces. It then starts up many copies of the program on a cluster of machines as shown in figure 1. Each cluster consists of hundreds or thousands of machines.
- Among the copies of the program one copy is special, the master. The rest are workers that are assigned work

by the master. Master picks the idle worker to assign each one a map task or reduce task.

- A mapworker reads the contents of the corresponding input split and parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory, local disk.
- The buffered pairs partitioned into regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master which forward these locations to the reduce workers. Reduce worker receiving these location from the master it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.
- The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function.When all map tasks and reduce tasks have been completed, the master wakes up the user program.

After successful completion, the output of the mapreduce execution is available in the R output files.

2) Performance Analysis: Since the Mapreduce is designed to help processing very large amounts of data using hundreds or thousands of machines, the library must tolerate machine failures gracefully.

Resilient to Worker Failure

MapReduce is resilient to large scale worker failures. The master pings every worker periodically. Failure of receiving no response from a worker in a certain amount of time, the master marks the worker as failed. When workers complete Map tasks are reset back to their initial idle state and become eligible for schedulings. Similarly, any map task or reduce task in progress on a failed worker is also reset to idle and becomes eligible for rescheduling.

3) Drawbacks:

- **Master Failure** : If the master fails then the above implementation aborts MapReduce computation. So the impact of master failure is disasterous.
- **Patent Risk** : MapReduce is a core technology in Google. By using it, Google Engineers are able to focus on their core algorithms rather than being bugged down by parallelization details. As a result, Mapreduce greatly increases their productivity it is no surprise that Google would patent such a technolodgy to maintain its competitive advantage. The MapReduce patent covers the Google implementation descrived above. A totally different architecture will poses a minimal risk w.r.t the MapReduce Patent which is important for enterprise customer who are more concern about potential risk.
- **Architectural Exploration** : Google only described one implementation of MapReduce programming model. There is no guarantee that this the best one as it suffer from master failure.We need to explore the tradeoffs of using a different one with a completely different architecture.
  So we need other framework for MapReduce on cloud computing.

B. MapReduce framework, Azure MapReduce for Microsoft Azure Cloud

Azure MapReduce, novel MapReduce runtime built using the Microsoft Azure Cloud infrastructure. Azure MapReduce [5] architecture successfully leverages the high latency, eventually consistent, yet highly scalable Azure infrastructure services to provide an efficient, on demand alternative to traditional MapReduce clusters.

1) Architecture: Azure MapReduce uses

- **Azure Queues** for map and reduce tasks scheduling
- **Azure tables** for metadata and monitoring data storage
- **Azure blob storage** for input, output and intermediate data storage and the Window Azure Compute worker roles to perform the computations.

When a user program calls the MapReduce function the following sequence of actions occurs:

- Client driver is used to submit the Map and reduce tasks to the worker node using Azure queues. Users can utilize the client API to generate a set of map tasks.

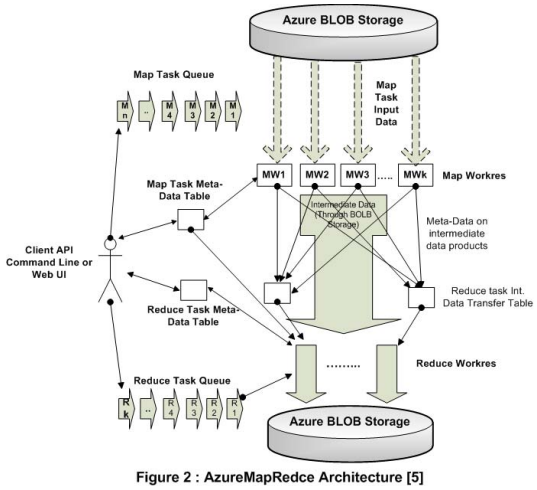

Figure 2. AzureMapReduce Architecture [5]

- Then the Map workers whose number is configured by users poll and dequeue map task scheduling messages from the scheduling queue, which were enqueued by the client API in figure 2. Map tasks upload the generated intermediate data to the Azure Blob Storage and put the key-value pair meta-data information to the

correct reduce task table.The scheduling messages contain the meta-data needed for the Map task execution, such as input data file location, program parameters, map task ID, and so forth.

- Users configure the number of Reduce tasks. There is an associated Azure Table corresponding each reduce task, containing the input key-value pair meta-data information generated by the map tasks. Reduce tasks fetch intermediate data from the Azure Blob storage based on the information. Each Reduce task starts processing the reduce phase; when all the map tasks are completed, and after all the intermediate data products bound for that particular reduce task is fetched. In the Azure MapReduce, each reduce task will independently determine the completion of map tasks based on the information in the map task meta-data table and in the reduce task meta-data table. After completion of the processing, reduce tasks upload the results to the Azure Blob Storage and update status in the reduce task meta-data table.

2) Performance Analysis:

- **Fault Tolerance:** In Azure Map Reduce, map and reduce tasks delete messages from the queue only after successful completion of the tasks. If a task fails or is too slow processing, then the message will reappear in the queue after the timeout. In this case, it would be fetched and re-executed by a different worker. This is made possible by the side effect-free nature of the MapReduce computations, which allows it to ignore the data communication failures.
- **No Master Failure:** As there is no master that distribute the tasks between works so there is no chance of having a failure of master which would led to the abortion of MapReduce Computing.

3) Drawbacks:

- **Fixed queue service time** : Azure Queue service currently doesn't allow for dynamic changes of visibility timeouts, which would allow for richer fault tolerance patterns. Currently Azure allows a maximum of 2 hours for queue message timeout, which is not enough for Reduce tasks of larger MapReduce jobs, as the Reduce tasks typically execute from the beginning of the job till the end of the job.
- **Lack of locality optimization** : When Map worker dequeue task from the scheduling queue it does not consider that if the related input data is locally available. It downloads the task data from blob storage before process and uploads the intermediate data on blob storage again. As the number of map tasks is much higher than the number of reduce tasks in the usual case, the effects will be manifold. First, it will increase the map task process time which will affect the overall performance. Second, network congestion may occur.
- **Lack of direct data passing between Map task and Reduce task** : In this system, Map task and Reduce do not pass data between themselves. This affects the overall performance too.
- **Table dependacy of metadata** : Though table based metadata makes Azure MapReduce free from single master and immune to master failure, it has few drawbacks. Metadata is saved in disks instead of cache which makes it persistent but slow to access and modify. It is also accessed by remote connections which only possible with slow performing streaming I/O. So. It will affect the overall performance if the system runs many map and reduce workers

C. Cloud MapReduce (CMR) for Amazon Cloud OS

Cloud MapReduce (CMR), which implements the MapReduce programming model on top of the Amazon cloud OS [7]. CMR is a demonstration that it is possible to overcome the cloud limitations and simplify system design and implementation by building on top of a cloud OS.

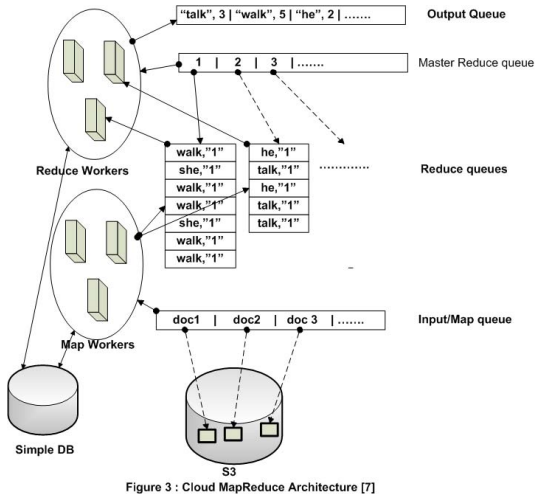

Figure 3. Cloud MapReduce Architecture [7]

1) Architecture: CMR's architecture is shown in the figure 3. There are several Queues:

- **One Input Queue** : holds the input to mapreduce computation
- **One Output Queue** : holds the result of mapreduce computation
- **One Master Reduce Queue** : holds many pointers, one for each reduce queue.
- **Many Reduce Queue** : holds the intermediate keyvalue pairs from the map function.

When a user program calls the MapReduce function the following sequence of actions occurs:

- In CMR, map workers poll the input queue for work and it dequeus a key-value pair and invokes the user

defined map function to process it. The ouput of this map funtion is intermediate key value pairs.

- The MapReduce framework then collects these 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.
- CMR uses the network to transfer the intermediate key-value pairs as soon as they are available, thus it overlaps data shuffling with map processing.
- When map workers finish their jobs, the reduce workers start to poll work from the master reduce queue. A reduce worker dequeues a message and responsible process 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 and thus the whole process go on. The reduce worker collects the outputs and write them to the output queue.

2) Performance Analysis: CMR has several highly desirable properties. Such as:

- **Incremental Scalability** : In Cloud MapReduce user can launch a number of servers at the beginning and also can launch additional servers in the middle of a computation if it thinks the progress is too slow.The new servers can automatically figure out the current job progress and poll the queues for work to process [7].
- **Symmetry and Decentralization** : There are no master or slave nodes every node has the same responsibility which simplifies system provisioning, configuration and failure recovery.
- **Heterogeneity** : The computing nodes could have varying computation capacity. The faster nodes would do more work than the slower nodes. In addition, the computing nodes could be distributed geographically.
- **More efficient data shuffling** : CMR uses queue as the intermediate point between Map and Reduce, which enables Map to push data to reduce. This design is similar to what is used in parallel database so it inherits the benefits of efficient data transfer as a result of pipelining.

3) Drawbacks: **Lack of locality optimization** : One drawback of the current CMR implementation is that it does not employ any locality optimization as it uses the network exclusively for I/O, bypassing all local storage. Such an architecture would encounter network bottleneck eventually in todays cloud infrastructure, when the network links between the computing nodes and the cloud services are saturated.

V. Comparison of Architecture and Performance

In the previous sections we have tried to explore different MapReduce architecture and their performances. At the very first we have discussed the generalized map reduce architecture which is very simple and powerful interface. It is highly scalable. This architecture is highly resilient to worker failure but not resilient to master failure. Moreover this architecture is the core technology in Google so it posses maximal risk with respect to the MapReduce patent. Moreover there is no scope for architectural exploration with a fixed architecture. So we have felt the need to explore completely different architectures.

Secondly we have discussed the Azure MapReduce architecture. It is fault tolerant and it resolves the problem of the generalized architecture by not incorporating the master in the architecture. Thus it became master failure resilient. But this architecture has some limitations regarding queue message timeout.Currently Azure allows a maximum of 2 hours for queue message timeout, which is not enough for reduce tasks of larger MapReduce jobs. Also in contrast to Amazon simple queue service, Azure queue service currently doesnt́ allow for dynamic changes of visibility timeouts, which would allow for richer fault tolerance patterns.

Thirdly we have described the Cloud MapReduce architecture, which has incremental scalability and also incorporate the heterogeneity. One drawback of the current CMR implementation is that it does not employ any locality optimization. It uses the network exclusively for I/O, bypassing all local storage.

Now in this section we are going to have a quick glance of the pros and cons of all the architecture that we have analyzed.

In the table I, we have give a quick glimpse of their architecture and in table II clearly highlighted the performance factors and in table III give the idea of overall pros and cons of every architecture and at the same time it also emphasized the need of evaluation of different architecture.

VI. Future Work

We have seen so far that the MapReduce programming model focuses on data transformation. The default sort-merge grouping algorithm is not efficient. There are some cretain pitfall of this kind of algorithm. So there is provision to explore different effective grouping and scheduling algorithm which can eventually improve the overall performance. In future a common platform for assesesing the performance of all the above architecture may be developed.

VII. Conclusion

The utility computing model introduced by cloud computing combined with the rich set of cloud infrastructure services offers a very viable alternative to traditional

Criteria	Generalize MapReduce	AzureMapReduce	CloudMapReduce
Existance of master	One master	No master	No Master
Number of reduce worker	Decided by master	User have the ability to configure	Decided by user
Existance of Queue	No Queue exist	Two types of Queues	Four types of Queues
Existance of table	No table	Two types of Tables for Metadata	No table
Storage	Local IDE disk	AzureBlob Storage	Donot store use network to transfer

Table I
COMPARISONS OF ARCHITECTURE

Factors	Generalize MapReduce	AzureMapReduce	CloudMapReduce
I/O Mode	streaming I/O	streaming I/O	streaming I/O
Scheduling	block level locality based	global simple queue based	Simple Queue Service(SQS) based
Fault tolerancy	Not resilient to master failure	Resilient to worker failure	Resilient to worker failure

Table II
COMPARISONS OF PERFORMANCE FACTORS

Name of Architecture	Pros	Cons
Generalize MapReduce [2]	Silmple and powerful interface Enables automatic parallelization Resilient to worker failure	Not resilient to master failure Patent Risk Lack of Architectural Exploration
AzureMapReduce [5]	Fault Tolarent Resilient to Master failure	Queue message timeout not enough Donnot allow for dynamic changes Lack of locality optimization Indirect data parsing Table dependacy of metadata
CloudMapReduce [7]	Incremental Scalability Symmetry and Decentralization Heterogeneity More efficient data shuffling between Map and Reduce	Donot employ locality optimization

Table III
COMPARISONS OF PERFORMANCE

servers and computing cluster. In this paper, we explored the challenges presented by cloud environments to execute MapReduce computation and how to overcome them through different architecture. At the same time we tried to explore different novel decentralized controlled frameworks like Azure MapReduce which fulfill the needs Azure users, Cloud MapReduce which is for Amazon Cloud OS and analysis their pros and cons in order to have a performance evaluation. We hope that the techniques we discussed are general enough that would be encouraged others to use them in other applications. Our survey would definitely open an promising approach to improve MapReduce performance for Cloud Computing.

REFERENCES

[1] M. Armbrust, A. Fox, G. Rean, A. Joseph, R. Katz, A. Konwinski, L. Gunho, P. David, A. Rabkin, I. Stoica and M. Zaharia, *Above the clouds:A Berkeley View of Cloud Computing*. Feb 2009.

[2] J. Dean and S. Ghemawat, *MapReduce: simplified data processing on large clusters* . CACM 51, 2008.

[3] J. Dean and S. Ghemawat, *MapReduce: a flexible data processing tool*. Communications of the ACM 53, Jan 2010.

[4] C. Gong, J. Liu, Q. Zhang, H. Chen and Z. Gong, *The characteristics of cloud computing*. International Conference on Parallel Processing Workshops (39th ICPPW'10), 2010.

[5] T. Gunarathne, T. L. Wu, J. Qiu, and G. Fox, *Mapreduce in the clouds for science*. CloudCom, IEEE, pp. 565–572, 2010.

[6] D. Jiang, B. C. Ooi, L. Shi and S. Wu, *The performance of mapreduce: An in-depth study*. PVLDB 3, pp. 472–483, 2010.

[7] H. Liu and D. Orban, *Cloud mapreduce: A mapreduce implementation on top of a cloud operating system*. CCGRID, IEEE, pp. 464–474, 2011.

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz and I. Stoica, *Improving mapreduce performance in heterogeneous environments*. OSDI, 2008.

[9] J. Ekanayake, S. Pallikara and G. Fox, *MapReduce for Data Intensive Scientific Analysis* pp. 277-284.