Enhancing the Inter-Cloud Data Transfer using Cache Manager

Ms. Lekshmi Mohan
Department of Computer Science and Engineering
NCERC , University of Calicut Thrissur, India

Mr. Siva Shankar S.
Department of Computer Science and Engineering
NCERC , University of Calicut Thrissur, India

Abstract

Now a day the importance of "Big Data" analytics is increasing. And the amount of data available over the internet has increased. Here the Hadoop MapReduce has an important role. it is very difficult to scale the inter cloud data transfer. Due to the latency it is very hard to transfer the data through multiple clouds. Here a framework is used; the data is processed in the internal cloud (IC) and external cloud (EC). These features are used to realize the inter-cloud MapReduce. This framework is used to meet the job deadlines as this meets the service level objective(SLO). Inorder to reduce the execution time cache manager is introduced to BStream. By using this we can save the data for the future and the future request for the data can be served faster.
Keywords: Cache Manager, Data Transfer, Hadoop

I. INTRODUCTION

In the recent years big data is a very popular term. It is mainly used to describe the exponential growth and availability of both structured and unstructured data in a traditional database. Volume, velocity and variety are the three v's of the big data. Variability and complexity are the dimensions. We can quickly solve the complex problems using big data and sophisticated analytics in a distributed, in memory and parallel environment. High performance analytics also make it possible to analyse all variable data to get precise answer for hard to solve problems and uncover new growth opportunities and manage unknown risks. All while using IT resources more effectively.

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. It is used to perform the real-time "Big Data" analytics like fraud detection etc. These applications have to meet some service level objective as the deadline and they will experience high variable input load. If there is any fail in meeting the service level objective will lead to some penalties like the revenue loss etc.

Cloud bursting is an application deployment model in which an application runs in a private cloud or data centre and bursts into a public cloud. The advantage of such a hybrid cloud deployment is that an organization only pays for extra compute resources when they are needed. It provides an alternative to over provisioning by offloading the excess load from the internal cloud(IC) to an external cloud (EC). The main examples are amazon EC2 Rackspace etc. There are some difficulties in scaling the Hadoop MapReduce. The first one is the Hadoop being a batch processing system they takes the entire input data before the start of the computation. The batch processing using the Hadoop in EC incurs huge startup latencies as the inter-cloud data transfer latencies are at least an order of magnitude higher than that within a data centre. So we have to wait for the entire input to be uploaded and after that processing it. Then the net difficulty is because of the tight coupling in the shuffle phase in MapReduce with reducers. As shuffle involves all node communication and the shuffling of the data from EC to reducers in IC takes longer due to the difference between inter cloud and intra clod bandwidth. The shuffle phase begins only after the reducers have started. This elongated shuffle phase not only delays job completion but also causes idle CPU cycles within reducers.

The newer version of Hadoop is called the yarn. Here in the yarn the fixed partitioning of the slots are not maintained. YARN is Yet Another Resource Negotiator in big data applications one of the large scale distributed operating system used is yarn. The start of reducers are delayed and it is possible to use these slots for performing map tasks.so by reducing the job completion time we can improve the utilization of the slots. In an inter cloud setting with elongated shuffle. It is not possible to delay the start of reducers in IC.so the shuffle phase in the EC can be further optimized by doing the reduce operation that decreases the data download size.

In this BStream there is a cloud bursting framework that overcomes the above difficulties and it incorporates various optimizations. BStream architecture uses the stream processing engine storm in EC and yarn in IC. When the data arrives, using the storm both the map and reduce operations are executed on the incoming stream of data in EC. By overlapping processing with input transfer, BStream minimizes the startup latencies in EC. When the reduce operation in EC is executed. It decreases the downloading size. Shuffle phase is optimized using the checkpointing strategies that download intermittent reduce output from EC to reducers in IC. Shuffle from EC can also be decoupled from YARN reducers, thereby allowing the reducers to start much later. Thus by using the stream processing technique and check pointing strategies, BStream enables pipelined uploading

processing and downloading of data.. Presently we are considering the jobs meeting the deadlines whose input is initially present in IC.

At first the Hadoop reducer will consume whatever checkpoint data is available. And then the Storm topology in EC is busy for a much longer duration, and thereby improving performance. The next step the rest of the output is processed. Inorder to optimize the BStream the cache manager is introduced and the request is analysed in the cache manager. The cache manager searches the request from the mapping phase in the database. If the data is present in the cache manager then the data is transferred to the map phase. The intermediate data for the request is transferred to mapping phase. If the data is not present in the cache manager means then there is no response to the map phase

Hadoop MapReduce does not offer any guarantees on the completion time of jobs. Recently many works have proposed that guarantees deadlines for MapReduce jobs.

A. One Cluster Deadline Management

A cost model is proposed by Kamal et al to determine whether job deadline can met. Any job whose deadline cannot be met is rejected. In this cost model, it assumes that the map phase, shuffle phase and reduce phase execute sequentially, thereby ignoring the overlap between map and shuffle phase. A frame work called ARIA (Automatic Resource Inference and Allocation) is proposed by Verma et al. uses job profiling and analytical model to schedule deadline based MapReduce jobs. ARIA determines the number of map and reduces slots required to meet job deadlines and considers the overlap between map and shuffle phase. However ARIA does not estimate the reduce slowstart parameter which can enable better utilization of slots as ARIA assumes that the reducers are started right from the beginning of execution. None of the above approaches utilizes resources from EC for meeting deadlines.

B. Multi-Cluster Deadline Management

We classify the survey related work into non-Hadoop based frameworks and Hadoop based frameworks.

1) Non-Hadoop based frameworks:

There are different approaches for partitioning MapReduce job across clusters. In the Figure 1b the MapReduce job is partitioned according to the policy where MapReduce jobs are run locally in each cluster and their results aggregate using a "Global-Reduce" phase [3]. The input data is partitioned across different MapReduce clusters according to their compute capacity. The input out/put sizes are assumed to be negligible. Like Aneka [4] and CometCloud [5] work like [6] [7] have proposed deadline-aware scheduling for MapReduce on top of different cloud platforms. In both of them, the job is partitioned according to Figure. 1 which incurs high network latencies when the shuffle data size is huge thereby increasing the duration of reduce. In [8] the authors use a generalized reduction API that integrates map, combine and reduce operation into a single operation(local reduction) at the node-level to reduce overheads of shuffling, grouping, and storing large intermediate data. A global reduction is finally performed to aggregate all the node level results. This is to complete the job as early as possible. The above mentioned work was extended to incorporate deadline or the cost constraints. Some works extended to support stream processing.

2) Hadoop based frameworks:

In [9] Cardosa et al. find out that the efficiency of running MapReduce across two clouds in different configurations , when they depend on the nature of the job, one of the configurations performs better. They use the batch processing. We have shown that the batch processing in EC shows significant start up latencies. Few work support streaming reads within maps form remote HDFS. Thus the input data transfer from IC overlapped by processing within maps in EC. While this approach decreases the startup latency, it increases the overhead during task failures in EC as the incoming data is not stored in EC. Before the map is performed on the local block each incoming data block is replicated [11]. The above works partition the job according to policy shown in Figure. 1 where only maps are executed in EC. According to the nodes compute capability and link speed in [10] the size of the input split to each node is varied.. In order to minimize network overheads, But their placement algorithm ignores the final destination where the output of reduce is required. Thus, it may end up running reduce task on a node with a slow out link. .

II. BSTREAM

A MapReduce job can be split in different ways across multi clouds. There are mainly four types of inter-cloud MapReduce approaches. In the first one the map tasks are distributed across IC and EC. This results in the huge data transfer overheads during shuffle operation. In the second one this problem overcomes by splitting the MapReduce job into different sub jobs and executing them separately on IC and EC. Here a final MapReduce job is responsible for the merging of results at the end. Due to this approach it introduces another overhead of launching an additional job for performing job for global reduction. In the next approach this problem is avoided by distributing both map and reduce task across multiple clouds. The problem of this approach is the difference in inter-clod and intra-cloud latencies can lengthen the shuffle phase, thereby wasting lots of compute cycles of the nodes. So in BStream we adopt an approach given in the below Figure 1. Here in the output of MapReduce job in EC is directly transferred to the reducers in IC. This model will determine the start time of reducers in IC considering the overheads associated with inter-cloud and intra-cloud transfer. This minimizes the wastage of compute cycles during shuffle phase without incurring the overhead of global reduction.

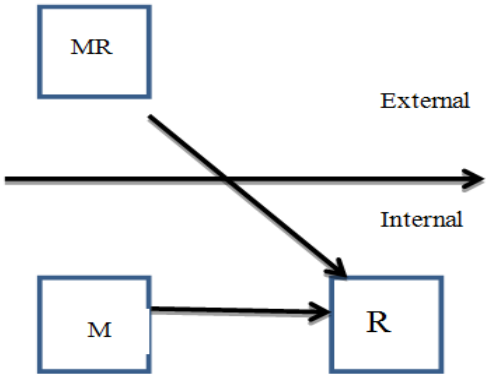

Fig. 1: Inter-Cloud MapReduce Approach

A. Working of BStream

BStream couples stream-processing in EC with batch processing in IC to realize inter-cloud MapReduce. Batch processing is more efficient in the IC since the input is already loaded in the HDFS. Stream processing is preferred in EC as it can perform computation on the incoming data as and when it arrives.

The Figure 2 shows the system architecture. The user submits the MapReduce job to the controller along with its deadline (step 1). The controller uses the estimator to determine the resource allocation (step 2). The estimator refers the job profile database and uses the analytical model (refer Section V) to estimate the resource allocation in IC (step 3). If the estimated resource allocation returned to the controller by the estimator (step 4) exceeds the available resources in IC, then the controller initializes the burst coordinator (step 5). It passes the values for (computed by the estimator) number of maps to be burst and the time to start bursting to the burst coordinator. The controller submits the MapReduce job to the Hadoop framework (step 6). The burst coordinator submits the Storm topology to Nimbus (step 7). It sequentially sends burst control signal for each data block to one of the data nodes (step 8a). The data node starts bursting data to the Kafka7 message queue (step 8b). The Storm spouts start consuming messages from Kafka in parallel (steps 8c). These messages (tuples) are processed through the Storm topology and the output is produced from the reduce bolt during commit8. The burst coordinator simultaneously updates the status of data transfer in Zookeeper (step 8d) that is referred by Storm to detect the end of data transfer (step 8e). The output from the reduce bolt is written to LevelDB9 in EC (step 9a). Two different checkpointing strategies are proposed to transfer this output to the reducers in IC (step 9b). Zookeeper is also used to store the reducer configuration in IC. This is used by the checkpoint controller to transfer the Storm output to appropriate reducer nodes in IC (step 9c). Hadoop AM is responsible for updating the Zookeeper with reducer configuration changes (step 9d). These updates are consumed by the checkpoint controller, which uses them for subsequent transfers from EC (step 9e). The reducers in IC merge the output from Storm and Hadoop to produce the final job output (step 10).

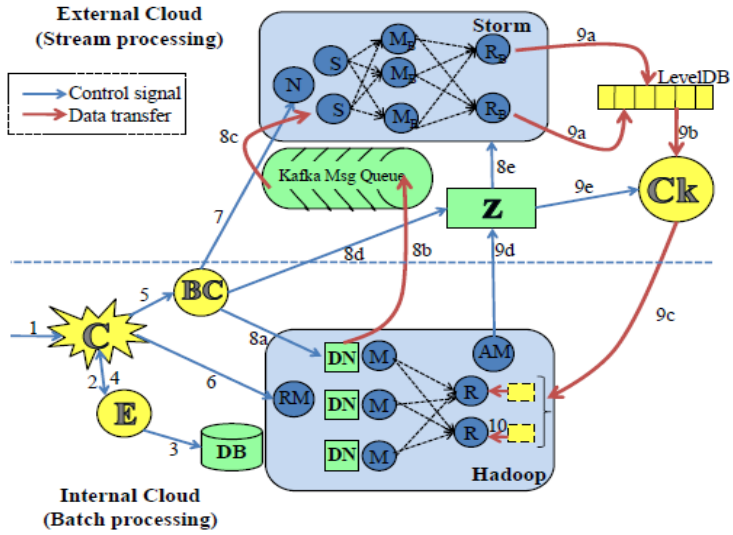

Fig. 2: BStream: Cloud Bursting Framework for MapReduce

B. Job Estimation

Profiling builds a job profile consisting of parameters discussed. The profile parameters such as map execution time, map selectivity, output to input ratio remain consistent across recurring datasets from same data source. On each unique data source an offline profiling is done once for every unique job. We run the job in IC with reduce slowstart as 1 and measure the profile parameters. The reduce slowstart is set as 1 so it ensures that reducers start only after all the maps are complete. Therefore, there is no waiting time within a reducer slot during the shuffle phase. Thus the measured profile parameters are free from any extraneous delays. Storm job profiling determines the minimum number of workers to be provisioned in EC such that data processing rate matches upload rate.

Because of the increase in the number of storm workers the processing time of a job decreases due to increased parallelism. However, processing rate is limited by upload rate to EC. When the number of workers required matching processing rate with arrival rate exceeds the maximum number of workers in EC, then additional processing delay is obtained.

When a job is submitted to the framework, the estimator retrieves the profile parameters from the profile database and uses the analytical model to determine the resource allocation plan. The controller takes responsibility of executing the job according to the resource allocation plan.

C. Functioning of YARN

The modified YARN is to burst map tasks to EC and to combine the output from EC in the Hadoop Reducers. When a job is submitted Hadoop runtime registers the file-split information of that job with the burst coordinator. The file splits are transferred as 256kb messages to the Kafka message queue. The outputs from the EC are directly transferred to reducer nodes in IC. The AM tries to maintain this reducer configuration by specifying hints while requesting for containers from RM. The burst coordinator stores in Zookeeper the initial set of nodes where the Hadoop reducers are likely to run. If the location of the reducer changes, then AM updates the Zookeeper. The new reducer reads the missed checkpoint data from HDFS If the reducer configuration changes. Each Hadoop reducer node runs a separate merger thread that merges the checkpoint data (as it arrives) with the previous output.

D. Modifications in storm

The storm is modified to implement the notation of job completion. The state change of the zookeeper will indicate the end of stream. The burst coordinator is responsible for updating the state changes. Once the output is downloaded from the EC the storm topology will be removed.

E. Checkpointing

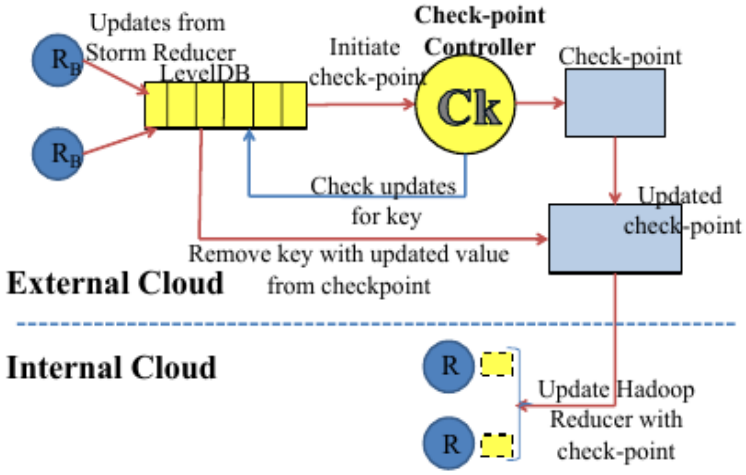

Fig. 3: Checkpointing Strategy

The time spent in downloading the output from EC can contribute a significant portion to the overall completion time of the burst job if the output to input ratio is high. In checkpointing, we take a snapshot of the current output of a reducer and transfer the snapshot data to IC. In the level DB the partial outputs are stored. The data output rate from Storm reducers is higher than the rate at which data can be downloaded. A key can get update from storm reducers during checkpoint transfer. That updated key will retransmit in the next checkpoint. To reduce the probability of retransmission, the checkpoint controller employs continuous updating strategy, where it checks for updates before sending a key to reducers in IC. The retransmission overhead is reduced if the updated keys are not transmitted as a part of the current checkpoint. it will be transmitted as part of the next checkpoint If the key does not receive any further updates. Since the network is kept busy throughout, there is no loss in terms of bandwidth utilization. If a certain key gets updated after it has been transmitted, then that key and the difference between its output value and checkpoint value are transmitted as part of the next checkpoint. The next checkpoint also includes keys that newly arrived in

the interval between the two checkpoints. The probability of updates or addition of new keys keeps varying depending up on the characteristics of the dataset.

F. Cache Manager

A cache manager holds the refernce to the cache and enhances and manages their criterion and lif cycle. Cache is a component that saves the data and for future purpose we can access the data very faster. The data stored in cache might be the result of an earlier computation. Cache hit occurs when the requested data can be found in the cache and the cache miss occurs when it cannot. The most request can be served from the cache the faster the system performs.

Here the request is trasferred to the cache manger. After the request is getting trasfered to the cache manger the request gets analyzed. The cache manager seraches the request from the mapping phase in the database. If the data is present in the cache manger means it is transferred to the mapping phase.if the data is not present in the cache manger means it will not respond to the mapping phase. So by using the cache manger we can get the data faster at any time and the execution time will be reduced.

III. PERFORMANCE EVALUATION

The performance of this technique is compared with the existing technique and a graph is plotted on the basis of execution time and is shown below as Figure 4.

Fig. 4: Execution Time Comparison

IV. RESULT

From the above graph, can be concluded that execution time in external cloud is many times higher than that of internal cloud. That means internal cloud implementation saves time compared to that of external cloud and thus can use internal cloud implementation for obtaining better performances.

V. CONCLUSION AND FUTURE SCOPE

In this paper, we have proposed a cache manger to the BStream to reduce the execution time of the submitted job. BStream architecture to meet the job deadlines across cloudsand it extends hadoop to multiple clouds by combining stream processing in EC and batch processing in IC. By using the cache manager the performance of the BStream is improved. If the data is present in the cache mamager means it will transfer the data to the mapping phase and if not it will not respond to the mapping phase.

As a part of our future work we can extend the framework to multiple jobs. We can extend the model by using the reducer in IC is executed by two partial steps. In the first step the reducer consumes whatever checkpoint data is available. The rest of the output from EC is processed as a part of the second step. This keeps the storm topology in EC busy for a much longer time, thereby improving performance.

ACKNOWLEDGEMENT

Authors would like to thank the authors of ARIA, CometCloud, Aneka for making their reference papers available. And also we extending our sinciere gratitude to the CSE department of Nehru College of Engineering and Reserch Center for offering us the full support.

REFERENCES

[1] Kc K and Anyanwu K, "Scheduling hadoop jobs to meet deadlines," in Proc. 2nd IEEE Int'l Conf. on Cloud Comput. Technology and Science (CLOUDCOM '10), 2010, pp. 388–392.
[2] Verma A, Cherkasova L, and Campbell R H, "ARIA. Automatic Resource Inference and Allocation for Mapreduce environments," in Proc. 8th ACM Int'l Conf. on Autonomic Comput. (ICAC '11), 2011, pp. 235–244.
[3] Luo Y, Guo Z, Sun Y, Plale B, Qiu J, and Li W W, "A hierarchical framework for cross-domain mapreduce execution," in Proc. 2nd ACM Int'l Wksp. on Emerging Computational Methods for the Life Sciences (ECMLS '11), 2011, pp. 15–22.
[4] Vecchiola C, Calheiros R N, Karunamoorthy D, and Buyya R, "Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka," Future Generation Comp. Syst., vol. 28, no. 1, pp. 58–65, 2012.
[5] Kim H and Parashar M, "Cometcloud: An autonomic cloud engine," in Cloud Computing: Principles and Paradigms. Wiley, 2011, ch. 10, pp. 275–297.
[6] Michael M, Rodrigo C N, and Buyya R, "Scaling mapreduce applications across hybrid clouds to meet soft deadlines," in Proc. 27th IEEE Int'l Conf. on Advanced Information Networking and Applications (AINA '13), 2013, pp. 629–636.
[7] Hegde S, "Autonomic cloudbursting for mapreduce framework using a deadline based scheduler," Master's thesis, Univ. of Rutgers, NJ, May 2011. [Online]. Available:http://nsfcac.rutgers.edu/people/parashar/Papers/samprita thesis.pdf.
[8] Bicer T, Chiu D, and Agrawal G, "A framework for data-intensive computing with cloud bursting," in Proc. IEEE Int'l Conf. on Cluster Comput., 2011, pp. 169–177.
[9] Cardosa M, Wang C, Nangia A, Chandra A, and Weissman J, "Exploring mapreduce efficiency with highly-distributed data," in Proc. 2nd ACM Int'l Wksp. on MapReduce and its Applications (MapReduce '11), 2011, pp. 27–34.
[10] Heintz B, Wang C, Chandra A, and Weissman J, "Cross-phase optimization in mapreduce," in Proc. IEEE Int'l Conf. on Cloud Eng. (IC2E '13), 2013.
[11] Kim S, Won J, Han H, Eom H, and Yeom H Y, "Improving hadoop performance in intercloud environments," SIGMETRICS Perform. Eval. Rev., vol. 39, no. 3, pp. 107–109, Dec. 2011.