Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 4, April 2015, pg.691 – 696

RESEARCH ARTICLE

Efficient Map Reduce Model with Hadoop Framework for Data Processing

Suhaib M Ansari[1], Saikiran Chepuri[2], Vaibhav Wadhai[3]

[1]SITE & VIT University, India

[2]SITE & VIT University, India

[3]SITE & VIT University, India

[1] es_em31@yahoo.co.in; [2] saikiranchepuri35@gmail.com; [3] vibs.wadhai@gmail.com

*Abstract: **Big data refers to the large-scale distributed applications that work on unprecedentedly large data sets. The Map-Reduce [12] framework and the Apache's Hadoop [13] [17], the software system for big-data applications. An observation regarding these applications is that they generate a large amount of intermediate data, and these abundant information is thrown away after the processing finish. By taking into the consideration I am proposing some ways to efficiently work on big data applications. The techniques I am using here are Data Cleaning in Hadoop, Push Model and caching. These three methods will help in providing efficient work on Hadoop. It will save a lot of time while working on the large data sets. The already implemented Map-Reduce Framework [11] is being modified so as to get the required result.***

Keywords: Caching, Push-Model, Data Cleaning, Map-reduce, DRF

Introduction

The two mechanisms Map-Reduce [12] and the Apache's Hadoop [13] are being intensely used for the large data sets. The difficulty of managing the large data sets have been a considerable headache for the organization in processing the data, but Map-reduce have envisioned in handling this Big-data in an effective way. Applications specify the computation in terms of a mapper and a reducer function working on partitioned data items.

Map-Reduce provides a uniform framework for implementing large-scale scattered computation on unprecedentedly large-scale data set. Though, there is a restriction of the system, i.e., the inefficiency in incremental processing. Incremental processing refers to the applications that incrementally grow the input data and continuously apply computations on the input in order to generate output. Some issues which are to be rectified are that there are potential duplicate computations being performed in this process, the Map-Reduce does not have the mechanism to identify such duplicate computations. For this I am implementing a Caching [15] mechanism which will fix this issue. The mechanism can be implemented by using the ehcache method which enable us speed up the already computed input.

The other thing is about the data processing, while it takes place the temporary data which we called as the context memory. There are two things present in the context i.e. (key, value), which is processed first and stored onto the mapper. We are using the process of Data cleaning for removing the temporary data which is present and not removed while map reduce task is performed. This task will not be able to clean and leaves many temporary data in memory which causes execution time to be run slowly and getting more time to clean the temporary data. It will affect a lot for providing efficiency in the system.

As we already know about that in the file systems comes the Map-Reduce engine, which consists of one Job Tracker, to which client applications submit Map Reduce jobs. The Job Tracker pushes work out to available Task Tracker nodes in the cluster, striving to keep the work as close to the data as possible.

If a Task Tracker fails or times out, that part of the job is rescheduled. The Task Tracker on each node spawns off a separate Java Virtual Machine process to prevent the Task Tracker itself from failing if the running job crashes the JVM. A heartbeat is sent from the Task Tracker to the Job Tracker every few minutes to check its status. The Job Tracker and Task Tracker status and information is exposed by Jetty and can be viewed from a web browser. If the Job Tracker failed on Hadoop 0.20 or earlier, all ongoing work was lost. Hadoop version 0.21 added some check pointing to this process. The Job Tracker records what it is up to in the file system. When a Job Tracker starts up, it looks for any such data, so that it can restart work from where it left off.

LITERATURE SURVEY

A widespread assessment on incremental calculation is given in Ref. [6]. Google Big table [6] is a distributed storage system for managing structured data built on top of the Google File System (GFS) [14]. It is able to efficiently handle incremental processing with the structure information in the data. Google Percolator [8] is an incremental processing platform, which achieves much faster web index processing compared to the previous Map Reduce like batch-processing system. It is used in Google's new web page indexing engine, which achieves 50% fresher web indexing than the old system. Ram cloud [6] is a distributed computing platform, where all data is kept in RAM instead of on disk. Unlike Hadoop, Ram cloud focuses on the computation and processing performed on small data objects. Dryad [1], is a distributed programming model that is targeted at the same application scenarios as Map Reduce. Unlike Map-Reduce simple two-phase execution model, Dryad employs a Directed Acyclic Graph (DAG) based model. Dryad is thus able to provide a more natural representation of many real-world problems. However, from an implementation point of view, such a design causes excessive complexity for application developers to implementation with Dryad, which substantially hinders its adoption. DryadInc [1] is an extension to Dryad to reuse identical work to accelerate processing our work focuses on dynamically identify redundant computation in Map Reduce job. Memory cached[19] is a distributed caching system designed as an object accessing layer between the application and underlying relational database.

The cache manager could utilize Memory cached to accelerate query response because the size of cache item is generally small. Scheuermann et al. [7] studied how to optimize the performance of distributed storage with disks. This work does not address the data sharing problem identified in this paper. This mechanism is orthogonal to ours and could be integrated straightforwardly. Performance optimization in data-intensive applications with Map Reduce is an active research topic. Herodotou et al. [9] proposed a intelligent cluster sizing algorithm for data-intensive analytics applications. Wu et al. [14] studied the query optimization problem in using Map Reduce to do online query processing. Both Wu et al.'s work and our work aim to accelerate processing by removing redundancy in the computing process. Their work is focused on a higher layer than ours. Although it has a similar problem as ours, it focuses on programming models instead of actual implementation. The input data is modeled as a stream. The system answers query by utilizing the continuing coming data stream. Their methodology is similar to ours in using actual profit to make management decisions.

Network levitated merge algorithm [2] has been incorporated for improving the performance. . A novel network-levitated merge algorithm is introduced to merge data without repetition and disk access. In addition, a full pipeline is designed to overlap the shuffle, merge, and reduce phases. Our experimental results show that Hadoop-A significantly speeds up data movement in Map Reduce and doubles the throughput of Hadoop.

Proposed System:

The motivation over this system is to use intermediate results for further execution of the applications, where we don't want to execute results which are previously processed. This will be achieved by Caching mechanism. Also the Data cleaning mechanism which enables us to clean the already present content memory to fasten the execution process. The Push Model is also the strategy to work which enables the job tracker to push the heart beat to the task tracker in order to work directly.

ARCHITECTURE

Data Cleaning:

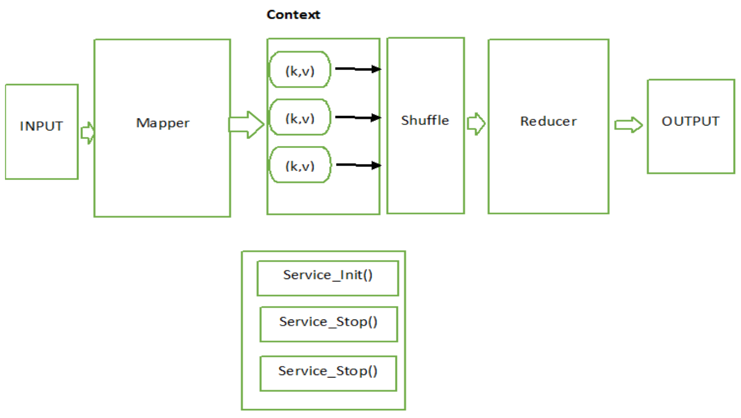


Fig 1. Data Cleaning Process

Push Model:

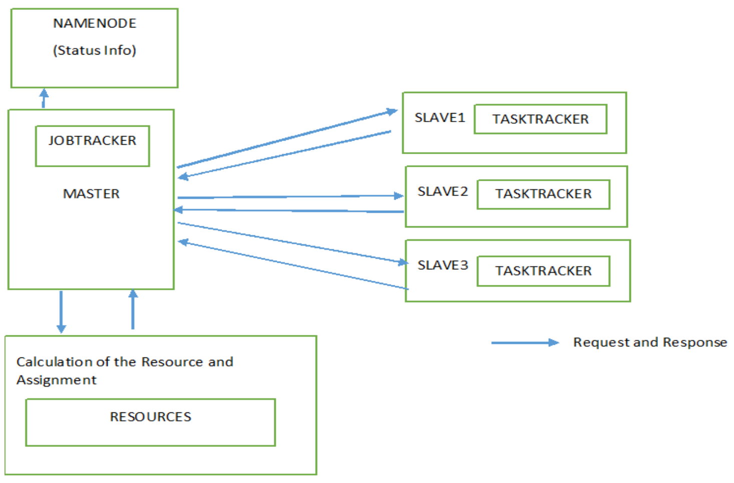


Fig 2. Push Model

In the Push model the use of DRF(dominant resource fairness) is used in order to allocate the resources while assigning the job to it. The status information of what is going on is saved in the Namenode. Jobtracker pushes the information to the task tracker to work out the job in the tasktracker. So by this we save the time complexity.

Caching:

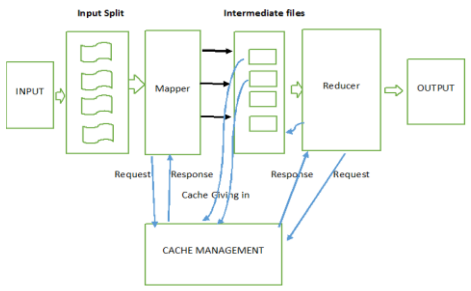


Fig 3. Caching Mechanism

The work has been implemented on Hadoop platform with Caching being done with help of ehcache methodology which enables the system to keep the already implemented input's output saved in the cache management. By this we can reduce number of complex operation in the mapper and the reducer phase thereby getting the output directly from the cache management.

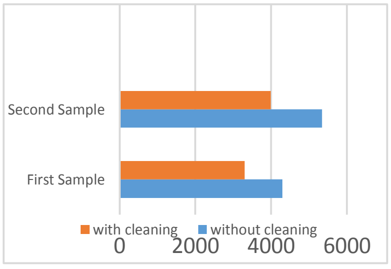
EVALUATION & RESULTS:

The evaluation that I have done based on the two ways are being assessed and the results are being taken, it shows a significant change in the execution time in processing the data. The two processes i.e. data cleaning and push model are allowing us a significant change in the execution of the data in the system. In Hadoop framework the change in this execution time is a very significant achievement. The Two processes are being formulated below:

1) Data Cleaning:

The process is being done and the results are being evaluated and the results are showing a significant change in the execution of the data. The execution which before used to take more time, now is being done in a very less time.

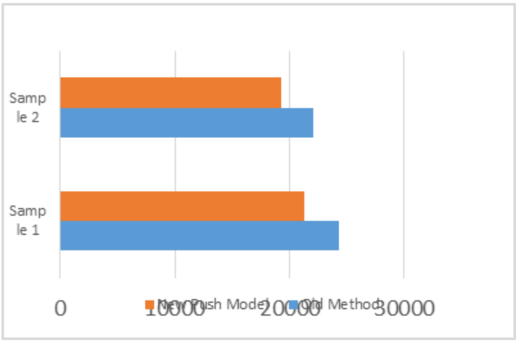
The basic change in this model with the previous model is about the context. The context contains the key and value of the input. The already used framework used to store the key and value in the mapper and then was directing it to the reducer, but now we are transferring the key and value to the reducer without saving it on the mapper, which saves a lot of time in the execution of the process.



The results shown in the graph is shows the data processing with data cleaning approach. The blue bar show the data before cleaning and the other shows data processing with data cleaning. The time which is shown is in milliseconds.

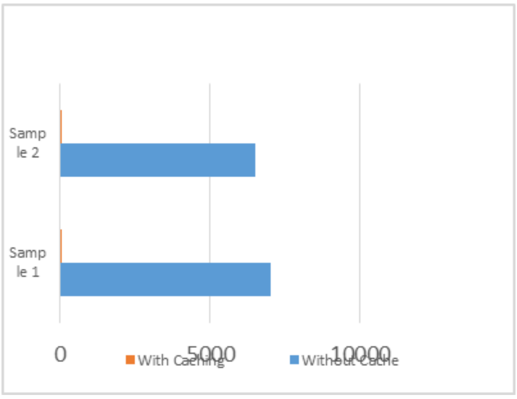
2) Push Model:

Here comes the concept of job tracker and the task tracker. The job tracker assigns the jobs to the task tracker. Before the mechanism which is there is that the task tracker when completing the work used to generate a heartbeat to the job tracker to get assigned with other jobs.
Now we are using a time interval based job assigning system, which assigns the jobs to the task tracker when notified but in a different manner. The jobs time interval is decided by the resource and the cluster capacity. The job tracker checks for the resource availability in some time interval and when the task tracker is free it is assigned the job. The status information will be stored in the name node.



The above graph is also taken from the observation done by the implementation of the push model. It depicts the execution time of the already used model and the new push model. There is a slight change in the execution time. As the data can be big, for data in GB and TB it can save us a lot of time for executed the input in the Hadoop.

3) Caching:



This mechanism allows us not to compute the result of the already computed input. Whenever the data which already has generated the output, we should not again compute it to gather the same result again. So for this I am using an ehcache method which enables us to search for the inputs output and not compute it and save the time of computation of the mapper and reducer phase and directly generate the output. The output is generated from the cache management which contains all the information of the data processed. This helps us in saving a lot of execution time.
The above bar graph depicts the execution time of the without cache model and with cache mode. The lines of the with cache model are not quite visible ass it is taking negligible amount of the time to execute the already executed input.

CONCLUSION

Time is a very precious thing in this world and processing big data i.e. gigabytes and terabytes of data takes a long time. So we should try to find out ways finding out the best possible ways in order to decrease the time complexity. The Analysis which I have done on the data which I have used to process has shown a change in the execution time. The time taken before by the system was comparatively more to the previous model used. By this we are able find out that the performance of the system can increase to a good extent.

FUTURE WORK

The future work related to this project can be regarded provided some other ways related to Hadoop framework to provide some more efficient ways to reduce the time complexity of the system.

ACKNOWLEDGEMENT

We are very much thankful to our guide Prof. Prabhavathy P who is currently working as Assistant Professor at VIT University, SITE department for timely and exclusive access to gain information on Data mining and Data Processing Issues.

REFERENCES

1. L. Popa, M. Budiu, Y. Yu, and M. Isard, Dryadinc: Reusing work in large-scale computations, in Proc. of HotCloud’09, Berkeley, CA, USA, 2009.
2. Weikuan Yu, Member, IEEE, Yandong Wang, and Xinyu Que, Design and Evaluation of Network-Levitated Merge for Hadoop Acceleration.
3. Y. Chen, S. Alspaugh, and R.H. Katz, “Interactive Query Processing in Big Data Systems: A Cross Industry Study of MapReduce Workloads,” Technical Report UCB/EECS-2012-37, EECS Dept., Univ. of California, Berkeley, Apr. 2012.
4. B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy, “A Platform for Scalable One-Pass Analytics Using MapReduce,” Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’11), pp. 985996, 2011.
5. G. Ramalingam and T. Reps. A categorized bibliography on incremental computation, in Proc. of POPL ’93, New York, NY, USA, 1993.
6. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data, in Proc. of OSDI’2006, Berkeley, CA, USA, 2006.
7. P. Scheuermann, G. Weikum, and P. Zabback, Data partitioning and load balancing in parallel disk systems, The VLDB Journal, vol. 7, no. 1, pp. 48-66, 1998.
8. D. Peng and F. Dabek, Largescale incremental processing using distributed transactions and notifications, in Proc. of OSDI’ 2010, Berkeley, CA, USA, 2010.
9. H. Herodotou, F. Dong, and S. Babu, No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics, in Proc. of SOCC’2011, New York, NY, USA, 2011.
10. S.Wu, F. Li, S.Mehrotra, and B. C. Ooi, Query optimization for massively parallel data processing, in Proc. of SOCC’2011, New York, NY, USA, 2011.
11. http://www01.ibm.com/software/data/infosphere/hadoop/mapreduce/
12. https://www.mapr.com/blog/how-write-mapreduce-program.
13. https://www.mapr.com/services/mapr-academy/Developing-Hadoop-Applications
14. Amazon web services, http://aws.amazon.com/, 2013.
15. Memcached—A distributed memory object caching system, http://memcached.org/, 2013.
16. http://www.dummies.com/how-to/content/tracking-jobtracker-and-tasktracker-in-hadoop-1.html.
17. Apache Hadoop Project, http://hadoop.apache.org/, 2013.