International Journal of Soft Computing 11 (5): 312-318, 2016
ISSN: 1816-9503

Efficient Big Data Analytics With Optimized Parallel Processing

S. Sravanthi and K. Thirupathi Rao
Department of Computer Science and Engineering, KL University, Vaddeswaram, India

Abstract: Now a days the word MapReduce is synonymous with big data processing. Different flavors of it is available over Apache Hadoop being at the core of big data processing. For well versed Java developers there is the direct interaction with the core there is Hive for the SQL proficient one's and there is Pig for procedural language aware developers. What ever the wrapper being used the core implementation of hadoop is simply is to divide the processing into two disjoint phases. One being the Map function and the other being the reduce function. So far many big data processing implementations are driven with the idea of equal distribution of workloads across processing nodes. We propose a dynamic distributed algorithm that is a processing aware job scheduler that assigns data processing nodes work load based on their prior performance throughputs. Extensive simulations using a 2.4 GB weather temperature conversion datasets demonstrates that our proposals can significantly reduce processing costs while prioritizing working nodes better compared to previous approaches.

Key words: Mapreduce, distributed programming, apache hadoop, big data, Hadoop Distributed File System (HDFS)

INTRODUCTION

The term MapReduce was first coined by Google in 2004 when they first published a study (Dean and Ghemawat, 2004) that specifies the implementation required to initiate a paradigm to process large data quickly in a distributed manner. The efficiency of MapReduce is in its ability to deploy and manage programs across domains to build a cohesive unit that can process data and acquire results effectively. Apache Hadoop developed by doug cutting which happens to be a pioneer open source MapReduce implementation. It's flexibility at the core to interact with any of the procedural languages such as C++, Java, Python, Ruby, SQL like via Hive, Piglets to initiate MapReduce has always been its positive feature. It's implementational prototypes such as Nutch search engine project in 2005 captured everyone's attention but the real deal was when it processed a terabyte of data in 209 sec using a 10,000-core Hadoop cluster at Yahoo in April 2008.

Relational Database Management System (RDBMS) aided with Structured Query Language (SQL) are often used to analyze tabular data, they are more suited when the volume of the data is small (<100 MB) and the data in question is continuously updated. The MapReduce approach supplements RDBMS systems especially with regard to structure, format and volume of data and the only difference is the data in question is stored once and accessed many times. The Hadoop based MapReduce approach is very helpful the when the volume of the data is huge and it is unstructured.

The growing trend is to use RDBMS systems for dynamic events and activities, for computationally expensive tasks like batch processing of files, using non relational data processing tools such as NoSQL Database, 2007 or Hadoop etc is more beneficial. The term BigData (Franks, 2012) is often associated with Hadoop as companies like Facebook, Yahoo pioneers of MapReduce technology demonstrated its's power in processing large volumes of datasets.

In this study, we analyze and highlight the performance differences between having hadoop implementations with or without a processing aware job scheduler. The data set used for processing is a 2.4 GB weather temperature conversion sets and processed in Hadoop Single Node Cluster mode. The performance parameters such as hadoop chunck size, processing nodes, processing time are considered to validate the performance difference of a processing aware job scheduler and a plain job scheduler. With out a processing aware job scheduler the plain hadoop implementations fared poorly with respect to the performance parameters. Besides the weather temperature conversion datasets similar datasets from other scientific domains such as environmental analysis (Stepisnik and Kosec, 2011), energy analysis (Wang *et al*., 2013), bioinformatics (Taylor, 2010), simulation-al data (Kosec and Sarler, 2013) can also be used to validate the proposed concept.

Corresponding Author: S. Sravanthi, Department of Computer Science and Engineering, KL University, Vaddeswaram, India

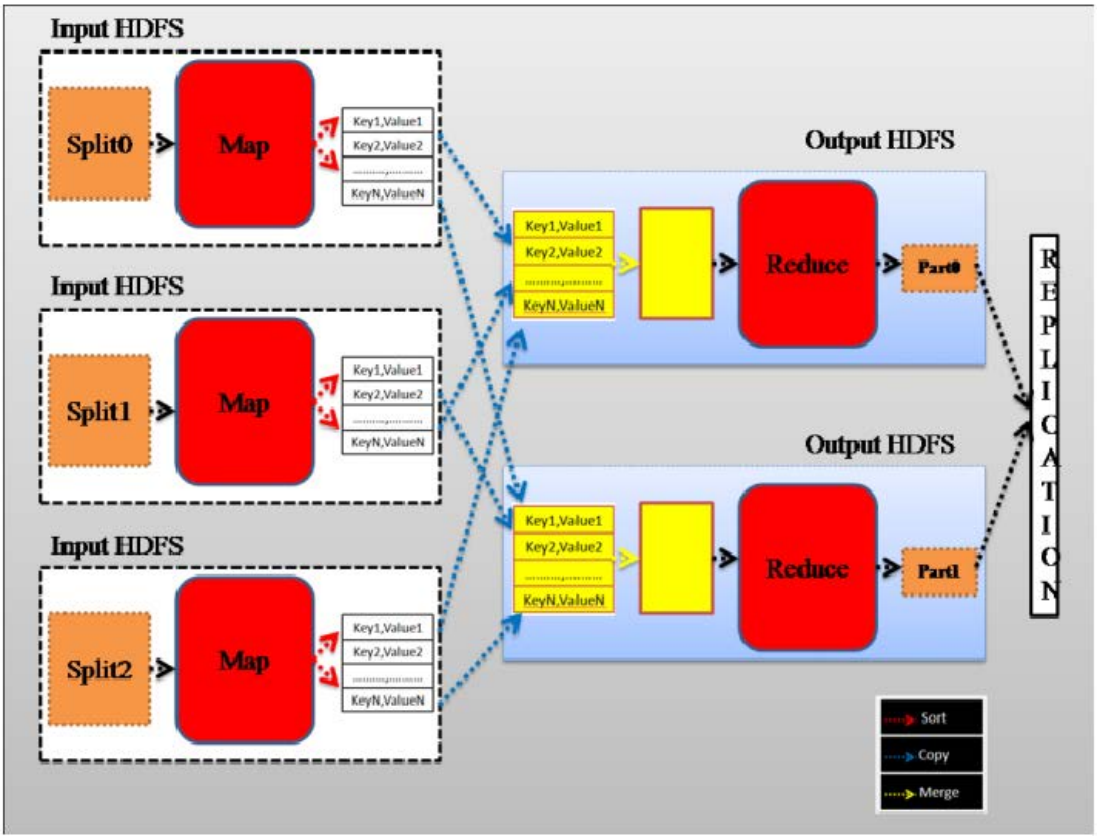

Fig. 1: A snapshot of MapReduce process of data flow in a typical Hadoop setup

Literature review

MapReduce model: It is a computational domain that implements distributed processing of big data sets. Most used tasks with respect MapReduce on big data are as follows:

- Distributed globally search a regular expression and print (grep)
- Count frequencies
- Graph structure representation of web artefacts
- Term-frequency
- Inverse document frequency

The four fundamental stages of any MapReduce model are as follows:

- Input splits
- Split based key value pair generation often called map
- Iterative sort, copy, merge operations on map
- Summing up similar keys with their respective
- Values often called reduce

Flow representation of the above stages are represented in Fig. 1. The problem of optimizing the job scheduler for an efficient distributed processing in MapReduce paradigm was researched earlier. We shall discuss some such methods. Most previous approaches concentrated on utilizing the Hadoop MapReduce platform for their respective domains but some focused on performance improvement within the platform itself especially with respect to tweaking the data transmissions and work allocations between the Hadoop entities.

Moniruzzaman and Hossain (2013) is the first to explore the possibility of performance improvement in the execution of a job by modifying network usage in such a way that it upholds high network resource utilization at reduced network congestion.

Palanisamy *et al.* (2011)'s prototype named Purlieus demonstrated the efficency of modifying the default MapReduce job assignment system especially using amazon AWS clouds by using bridge machines between cloud instances and local machines. These partitions reduces overall network congestion using time

synchronized shuffling between cloud instances and local machines. But no research was implemented to optimize the shuffling module to reduce network traffic loads in MapReduce tasks. Palanisamy *et al.* (2011)'s research with respect to shuffling of data partitions among intermediate workers(data nodes) was picked up by Ibrahim *et al.* (2010). The default configuration of hadoop is the implementation of a hash map based data structure that would produce in coherent and in-consistent results which require further rounding-off key value pairs to synchronize the input and output data. Ibrahim *et al.* (2010) overcame the rounding off process with a new scheme called fairness-aware hash-map approach that has a cache to store and audits all the intermediate distributions to guarantee a balanced load distribution among worker nodes. Fan *et al.* (2014) is the first ever to suggest modifying work allocation scheduler itself and in that regard a scheduler algorithm was developed for data (key value pairs) distributions to enhance load sharings among worker nodes. Hsueh *et al.* (2014) improved Liya's scheme load balancing with an efficient load balancing scheme of their's and compared both methods to evaluate skewness performance of MapReduce jobs of large datasets.

All researches so far have been implemented with respect to task reduction, optimizing shuffling, partitions, generation of key value pairs, local aggregation, cross domain mappers, network aggregation etc but not on the job scheduler itself which happens to be the current scope of our study.

Unfortunately, all above research focuses on load balance at reduce tasks, ignoring the network traffic during the shuffle phase. In addition to data partition, many efforts have been made on local aggregation in mapper combining and in-network aggregation to reduce network traffic within MapReduce jobs.

MATERIALS AND METHODS

System model: The following node clusters will be deployed in a usual Hadoop big data processing system. Our proposal of optimizing the job allocation tasks will be initiated at Job Tracker for load aware job assignment among data nodes. Client submits job for execution JobTracker One per Hadoop cluster coordinates job Scheduler TaskTracker One per cluster node executes individual map/reduce tasks MapTask or ReduceTask at Child (Data Nodes) application code architectural representation of the proposed system is as follows in Fig. 2.

Proposed work: The default capacity scheduler of Hadoop was intended to permit smooth sharing of chunks

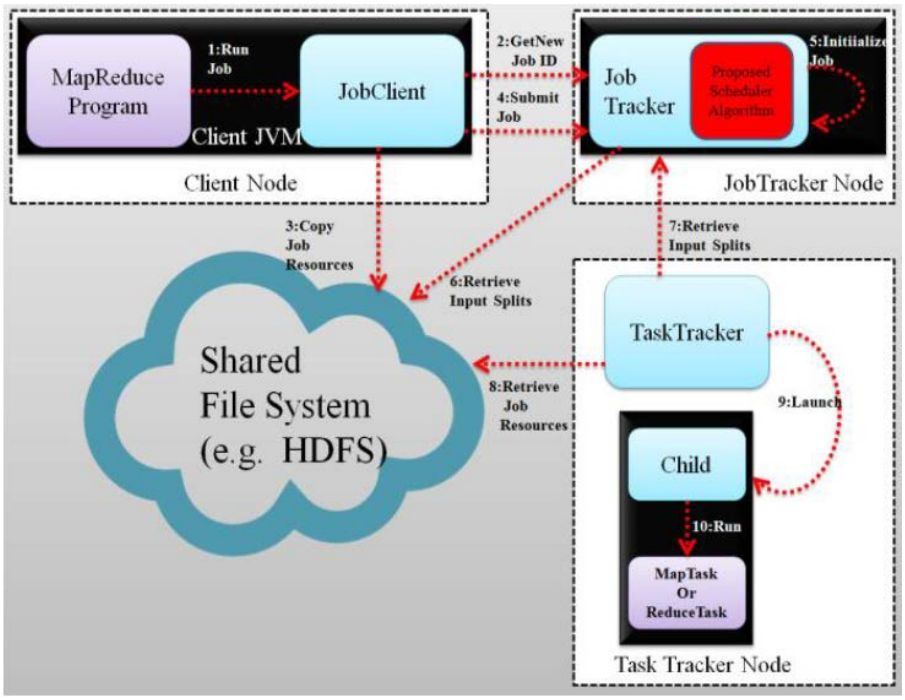

Fig. 2: An architecture representation of node clusters in a typical hadoop big data processing system and the proposed scheduler optimization implemented at job tracker entity

Let:

- $JobClasses1 = \{C_1, \ldots, C_n\}$, where C_i represents a class of jobs whose minimum share > 0, and share similar arrival rates, estimated execution rates, and priorities.
- $JobClasses2 = \{C'_1, \ldots, C'_n\}$, where C'_i represents a category of all the jobs in the system with similar arrival rates, estimated execution rates, and priorities.

Upon receiving a message (either a "job arrival" from a user or a "heartbeat" from a free resource) from the Hadoop system:

- If incoming message = job (say J) arrival
  1. Get the estimated execution time of J from the Task Scheduler component.
  2. Check if J fits in the current job classification
     (a) If J fits in any class (say C_i)
        i. add J to the queue of C_i
     (b) Else
        i. update the job classification by adding a new class for J (say C_j)
        ii. add J to the queue of C_j
        iii. find a matching of classes and resources using an optimization approach, which results in two sets of suggested classes for each resource as follows:

$$SC_R = \{C_1, \ldots, C_n\}, \text{ where } C_i \in JobClasses1$$
$$SC'_R = \{C'_1, \ldots, C'_n\}, \text{ where } C'_i \in JobClasses2$$

- If incoming message = heartbeat from a free resource (say R)
  – For each free slot in the resource R (say S_R):
  1. Look for a job where

$$job.user.min\_share - job.user.currentshare > 0$$

and

$$job.class \in SC_R$$

and

$$((job.user.min\_share - job.user.currentshare) \times priority) \text{ is maximum}$$

  2. If no job is found, look for a job where

$$job.class \in SC'_R$$

and

$$(job.user.currentshare/priority) \text{ is minimum}$$

  3. If a job is found (say $J_{selected}$), send S_R and $J_{selected}$ to the Task Scheduler. The Task Scheduler chooses a task from $J_{selected}$ and assigns it to S_R.
  4. If no job is found, S_R will left unassigned until the next heartbeat message.

Fig. 3: Proposed scheduler

of big data to be processed between Hadoop node clusters name node, data node, job tracker, task tracker, data node, job tracker, task tracker, etc) in an anticipated, quick and straight forward way using pre-defined or default configurations while utilizing the queues provided at each node.

We propose to consider data segregation and conglomeration for a MapReduce work with a motive to minimize total round trips and hence the aggregated system traffic between Hadoop clusters by assigning jobs to more resource centric workers instead of equal distributions. Specifically, we propose an appropriate job queue estimation calculation for big data applications by splitting the primary large data into process able chunks say 128MB (64MB Chunk Size for Hadoop) that can be fathomed and processed in parallel at data nodes. Algorithmic representation of the proposed scheduler is as shown in Fig. 3.

The proposed scheduler manages the data segments accumulation and assignments in a dynamic way based on the load awareness of processing nodes. Simulated results using Hadoop in pseudo distributed mode shows that our method can altogether decrease job allocation costs. It assures job allocation guarantees for queues and simultaneously provisioning elasticity for queues in the sense that limited capacity of a worker nodes determines less possible throughput and a more resource centric worker node determines more throughput. These queues at worker nodes can be harnessed by Job Tracker based on the temporal demand of the job at hand and previous

processing statistics of worker nodes. This results in significantly higher cluster utilization while still providing assured and reliable outputs for Hadoop workloads.

In the pseudo code, we shall define two set of Job classes, one containing the original workload allocations to data nodes and the other containing revised workload allocations to data nodes based on priority tweaks. The initial if clause of the pseudo code specifies the job request to the job tracker from the task tracker of data node which is the default setup. The execution time of the previous job is stored for prioritizing subsequent jobs. Our optimization tweak can be seen in the later if clause of the pseudo code with regard to a heart beat acknowledgement request combined with the previously stored execution time. This combined pseudo code extension to the default hadoop setup ensures performance improvement with respect to overall execution time of processing large data sets.

RESULTS AND DISCUSSION

We design the implementation on a standard machine with the configurations Intel (R) Pentium (R) D CPU 2.8 Ghz, 2 GB RAM, 3.2 bit Windows 7 operating system. The cluster and job configuration applied in our current weather temperature conversion data set are as follows. The following name node hdfs statistics snapshot validates our claim of using the 2.4 GB temperature conversion dataset (Fig. 4 and Table 1).

We have conducted rigorous evaluation on varying sizes of data sets across the native scheduler and optimized scheduler, the results show that the optimized job scheduler performs more efficiently than the hadoop native scheduler. We also analyzed several factors that may affect job performance, including differences between CPU bound and I/O bound workloads, relationship between task and job speedup and the efficiency of sorting optimization and trade-offs but the considerable and noticeable difference is observed in the processing time. A graphical plotter describes the performance differences of using native scheduler based approach and an optimized load aware job scheduler with respect to execution time (Fig. 5).

Table 1: Hadoop configuration table

Job	Temperature conversion
Compression	Disabled
Dfs.block.size	128 MB
dataset size	2.4 GB
Data nodes cluster size	3
Hadoop version	1.0.3
Slots	Map, Reduce
Virtual machine instance	1
Virtual machine tool	VMWare Cloudera-CentOS

Contents of directory /user/cloudera

Goto : /user/cloudera go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
.Trash	dir				2015-10-14 15:57	rwx------	cloudera	cloudera
.staging	dir				2016-01-20 01:41	rwx------	cloudera	cloudera
1901	file	1.2 GB	3	128 MB	2016-01-19 01:46	rw-r--r--	cloudera	cloudera
1902	file	1.2GB	3	128 MB	2016-01-19 01:47	rw-r--r--	cloudera	cloudera
MaxTemp.jar	file	3.89 MB	3	128 MB	2016-01-19 01:39	rw-r--r--	cloudera	cloudera
maxtemp1	file	1.2 GB	3	128 MB	2015-10-13 15:43	rw-r--r--	cloudera	cloudera
maxtempout	dir				2015-10-13 14:38	rwxr-xr-x	cloudera	cloudera
maxtempout1	dir				2016-01-20 01:41	rwxr-xr-x	cloudera	cloudera
project	dir				2016-01-20 01:24	rwxr-xr-x	cloudera	cloudera
sravmaxtemp	dir				2015-10-14 15:36	rwxr-xr-x	cloudera	cloudera
sravtemp	dir				2015-10-13 16:55	rwxr-xr-x	cloudera	cloudera
wcoutput	dir				2015-09-25 12:53	rwxr-xr-x	cloudera	cloudera
wordcount	dir				2015-09-25 12:32	rwxr-xr-x	cloudera	cloudera

Fig. 4: Name node big data statistics

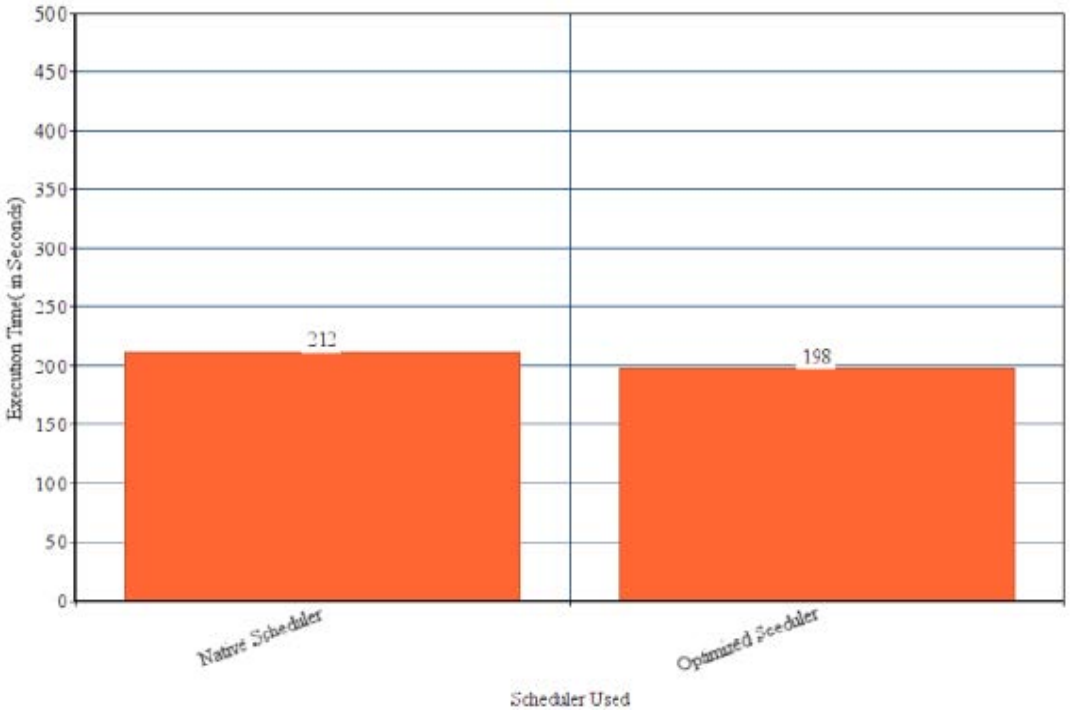

Fig. 5: Performance evaluation between native scheduler based and an optimized load aware job scheduler

CONCLUSION

In this study, we investigated the performance bottle necks in hadoop big data processing and identified a potential scope for improvement area with respect to scheduling at job tracker instance. For streamlining data segmentation, accumulation and assignment in MapReduce framework and to minimize system traffic and round trip times during job allocations for enormous big data applications utilizing processing nodes processing capabilities using an adaptive scheduler. We proposed an optimized two way job scheduler that considers each job's execution time of a data node and based on that decides the size of data that needs to be assigned for processing of same data node. To manage the substantial scale of big data sets of around 2.4 GB, we deployed a dispersed calculation to take care of job scheduling through pseudo distribution set up of hadoop architecture. A possible future work might be to consider dynamic expanding big data than static fixed size data sets for processing. That could be really helpful for more real time needs of various organizations. The obtained results show that our proposition can adequately decrease system execution time under different configurations settings especially with respect to size of the data.

REFERENCES

Dean, J. and S. Ghemawat, 2004. MapReduce: Simplified data processing on large clusters. Proceedings of the 6th Symposium on Operating Systems Design and Implementation, December 6-8, 2004, San Francisco, CA., USA., pp: 137-150.

Fan, L., B. Gao, X. Sun, F. Zhang and Z. Liu, 2014. Improving the Load Balance of Mapreduce Operations Based on the Key Distribution of Pairs. Cornell University Library, Ithaca, New York, USA.,.

Franks, B., 2012. Taming the big Data Tidal Wave: Finding Opportunities in Huge Data Streams with Advanced Analytics. Vol. 49, John Wiley & Sons, New York, USA.,.

Hsueh, S.C., M.Y. Lin and Y.C. Chiu, 2014. A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys. Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing, Vol. 152, January 20-23, 2014, Australian Computer Society Inc, Darlinghurst, Australia, ISBN: 978-1-921770-34-0, pp: 3-9.

Ibrahim, S., H. Jin, L. Lu, S. Wu and B. He *et al*., 2010. Leen: Locality fairness-aware key partitioning for mapreduce in the cloud. Proceeding of t he 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), November 30-December 3, 2010, IEEE, Wuhan, China, ISBN: 978-1-4244-9405-7, pp: 17-24.

Kosec, G. and B. Sarler, 2013. Solution of a low Prandtl number natural convection benchmark by a local meshless method. Int. J. Numer. Meth. Heat Fluid Flow, 23: 189-204.

Moniruzzaman, A.B.M. and S.A. Hossain, 2013. NoSQL database: New era of databases for big data analytics-classification, characteristics and comparison. Int. J. Database Theor. Appl., 6: 1-14.

Palanisamy, B., A. Singh, L. Liu and B. Jain, 2011. Purlieus: Locality-aware resource allocation for MapReduce in a cloud. Proceedings of the 2011 International Conference on for High Performance Computing, Networking, Storage and Analysis, November 12-18, 2011, ACM, New York, USA., ISBN: 978-1-4503-0771-0, pp: 1-58.

Stepisnik, U. and G. Kosec, 2011. Modelling of slope processes on karst modeliranje pobocnih Procesov na Krasu. Acta Carsologica, 40: 267-273.

Taylor, R.C., 2010. An overview of the Hadoop-MapReduce-HBase framework and its current applications in bioinformatics. BMC. Bioinf., 11: S1-S12.

Wang, L., J. Tao, R. Ranjan, H. Marten and A. Streit *et al*., 2013. G-Hadoop: Map reduce across distributed data centers for data-intensive computing. Future Gener. Comput. Syst., 29: 739-750.