International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 4, Issue 4, July - August 2015 **ISSN 2278-6856**

Analysis of Dynamic Data Placement Strategy for Heterogeneous Hadoop Cluster

Richa Jain[1], Amit Saxena [2], Dr. Manish Manoriya[3]

Research Scholar[1], Professor[2], Professor[3]
Truba Institute of Engg.and Information Technology, Bhopal

ABSTRACT
MapReduce has become a very important distributed process model for large scale data-intensive applications like Web Data and Data Mining. Hadoop is an open source implementation of MapReduce is wide used for large data processing which requires low time response. This paper, address the matter of approach to place data across nodes in an exceedingly way that every node contains a balanced processing load. Given an Data intensive application running on a Hadoop MapReduce cluster, our Data placement theme adaptively balances the number of knowledge hold on in every node to realize improved data-processing performance. Experimental results show that our Data placement strategy will forever improve the MapReduce performance by rebalancing information across nodes before playacting a data intensive application in an exceedingly heterogeneous Hadoop cluster. It is necessary for data placement algorithms to partition the input and intermediate data supported the computing capacities of the nodes within the cluster. This Hadoop implementation assumes that each node in an exceedingly cluster has an equivalent computing capability which the tasks are data-local, which can increase further on top of and scale back MapReduce performance. This paper proposes a dynamic data placement algorithm to resolve the unbalanced node employment downside. The planned technique will dynamically adapt and balance data hold on in every node supported the computing capability of every node in an exceedingly heterogeneous Hadoop cluster. The planned algorithm will scale back data transfer time to realize improved Hadoop performance. The experimental results show that the dynamic information placement policy will decrease the time of execution and improve Hadoop performance in an exceedingly heterogeneous cluster.

Keywords:-Hadoop, MapReduce, Heterogeneous, Data Placement

1.INTRODUCTION
Massive amounts of data are being generated a day in an exceedingly style of domains starting from scientific applications to social networks to retail. The stores of data on that recent businesses trust are already large and increasing at an unexampled pace. Organizations are capturing data at deeper levels of detail and keeping additional history than they ever have before. Managing all of the data is so rising mutually of the key challenges of the new decade. This deluge of data has to increase of parallel and distributed data management systems like parallel databases or MapReduce frameworks like Hadoop to research and gain insights from the data. Advanced analysis queries are run on these data management systems so as to spot fascinating trends, create uncommon patterns stand out, or verify hypotheses. In parallel databases, the queries generally incorporate multiple joins, cluster definitions on multiple attributes, and sophisticated aggregations. On Hadoop, the tasks have similar flavor with simplest of map-reduce programs being aggregation tasks that kind the premise of study queries. There have additionally been several makes an attempt to mix the quantifiability of Hadoop and declarative querying skills of relative databases. For fault tolerance, load equalization and availableness, these systems typically keep many copies of every information item (e.g., Hadoop classification system (HDFS) maintains a minimum of three copies of every information item by default. Our goal during this work is to point out a way to exploit this inherent replication in these systems to reduce the quantity of machines that are concerned in execution a question, known as the question span (we use the term question to denote each SQL queries and Hadoop tasks).

There ar many motivating reasons for doing this:
Minimize the communication overhead: question span directly impacts the entire communication that has got to be performed to execute a question. This is often clearly a priority in distributed setups (e.g., grid systems or multi-datacenter deployments); but even at intervals an data center, communication network is sold, and particularly cross rack communication data measure will be a bottleneck. HDFS, for example, tries to put all replicas of data item in an exceedingly single rack to reduce inter-rack data transfers. Our algorithms will be accustomed any guide these selections and cluster replicas of multiple information things on to one rack to boost network performance for queries that access multiple information things, that HDFS presently ignores. In cloud computing, the entire communication directly impacts the entire greenback price of execution the question.
Minimize the entire quantity of resources consumed: it's well-known that correspondence comes with vital startup and coordination overheads, and that we generally see sub-linear speedups as a results of these overheads and information skew. though the time interval of a question typically decreases in an exceedingly parallel setting, the entire quantity of resources consumed generally will increase with redoubled correspondence.
Our goal is to search out a layout that minimizes the common price over all queries. Our algorithms will optimize for load or storage constraints, or both. Our key

contributions embrace formulating and analyzing this problem, drawing connections to many issues studied within the graph algorithms literature, and developing economical algorithms for information placement. Additionally, we tend to examine the special case once every question accesses at the most two information things Our techniques are applicable in partition farms like MAID,PDC, or Rabbit, that utilize a set of a partition array as a workhorse to store widespread information in order that alternative partitions might be turned off or sent to lower energy modes. Vital work has been done on the converse downside of minimizing question response times or latencies. Declustering refers to the approach of investment correspondence within the partition scheme by spreading out blocks across completely different partitions in order that multiblock requests will be dead in parallel. In distinction, we try to cluster information things along to reduce the quantity of web sites needed to satisfy a fancy analytical question. Minimizing average question spans through replication and information placement raises 2 issues. First, will it adversely have an effect on load balancing? Focusing merely on minimizing question spans will result in a load imbalance across the partitions. However, we tend to don't believe this to be a serious concern, and that we believe total resource consumption ought to be the key optimisation goal. Most analytical workloads ar generally not latency-sensitive, and that we will use temporal scheduling (by suspending sure queries) to balance hundreds across machines. We are able to additionally simply modify our algorithms to include load constraints. A second concern is that the price of duplicate maintenance. However, most distributed systems do replication for fault tolerance, and thence we tend to don't add any further overhead. Secondly, AN increasing range of widespread applications became data-intensive in nature. within the past, the planet Wide net has been adopted as a perfect platform for developing information intensive applications, since the communication paradigm of the Web is sufficiently open and powerful. Representative data-intensive net applications embrace search engines, on-line auctions, webmail, and on-line retail sales. Data intensive applications like data processing and net compartmentalization have to be compelled to access ever-expanding information sets starting from some gigabytes to many terabytes or perhaps petabytes. Google, as an example, leverages the MapReduce model to method or so twenty petabytes of knowledge per day in an exceedingly parallel fashion. MapReduce is a beautiful model for parallel processing in superior cluster computing environments. The quantifiability of MapReduce is verified to be high, as a result of employment within the MapReduce model is divided into varied tiny tasks running on multiple machines in an exceedingly large-scale cluster.

MapReduce program directs file queries to a namenode, that successively passes the file requests to corresponding nodes within the cluster. Then, data nodes at the same time feed Map functions within the MapReduce program with great amount data. Once new application information are written to a go into HDFS, file fragments of the file are hold on multiple data nodes across the Hadoop cluster. HDFS distributes file fragments across the cluster, assumptive that every one the nodes have identical computing capability. Such a homogeneity assumption, which may probably hurt the Hadoop performance of heterogeneous clusters.

In native MapReduce, they create many assumptions. as an example, They assume that nodes in cluster will perform work roughly an equivalent rate, all tasks progress at a relentless rate throughout time. there's no price to launching a speculative task on a node that will otherwise have AN idle slot. Tasks within the same class (map or reduce) need roughly an equivalent quantity of labor. These motivates USA to develop data placement schemes which will perceptibly improve the performance of heterogeneous Hadoop clusters.

We observe that information section could be a determinative issue for MapReduce performance. To balance load, Hadoop distributes information to multiple nodes supported disc space availableness. Such data placement strategy is extremely sensible and economical for a solid atmosphere wherever nodes are identical in terms of each computing and disk capability. In solid computing environments, all the nodes have identical employment, indicating that no information must be enraptured from one node into another. In an exceedingly heterogeneous cluster, however, a high performance node will complete native processing quicker than a low-performance node. When the quick node finishes process data residing in its native disk, the node must handle unprocessed data in an exceedingly remote slow node. The overhead of transferring unprocessed data from slow nodes to quick peers is high if the number of enraptured data is large. AN approach to boost MapReduce performance in heterogeneous computing environments is to considerably scale back the number of knowledge enraptured between slow and quick nodes in an exceedingly heterogeneous cluster. To balance data load in an exceedingly heterogeneous Hadoop cluster, we tend to are motivated to analyze data placement schemes, that aim to partition an outsized information set into information fragments that are distributed across multiple heterogeneous nodes in an exceedingly cluster.

In this paper, we tend to introduce data placement mechanism within the Hadoop distributed classification system or HDFS to initial distribute an outsized data set to multiple nodes in accordance to the computing capability of every node. Additional specifically, we tend to implement an data reorganization algorithm additionally to an information distribution formula in HDFS. Data reorganization and distribution algorithms enforced in HDFS will be accustomed solve the information skew downside thanks to dynamic data insertions and deletions.

2.RELATED WORK

Data partitioning and replication plays a progressively necessary role in massive scale distributed networks like content delivery networks (CDN), distributed databases and distributed systems like peer-to-peer networks. Recent work has shown that considered placement of knowledge and replication improves the potency of question process algorithms. There has been some recent interest on up information colocation in massive scale process systems like Hadoop. Recent work by Eltabakh on CoHadoop is extremely near our work, wherever they supply AN extension for Hadoop with a light-weight mechanism that enables applications to regulate wherever data is hold on. They concentrate on data colocation to boost the potency of the many operations, together with compartmentalization, grouping, aggregation, columnar storage, joins, and sessionization. Our techniques are complimentary to their work. Hadoop++ is another closely connected work, wherever it exploits information pre-partitioning and colocation. there's substantial quantity of labor on duplicate placement that focuses on diminution of network latency and information measure. Neves et al. [9] propose a method for replication in CDN wherever they replicate information on to set of servers to handle requests in order that the traffic price within the network is reduced. There has been plenty of labor on dynamic/adaptive duplicate management , wherever replicas are dynamically placed, enraptured or deleted supported the read/write access frequencies of the information things once more with the goal of minimizing data measure and access latency.

Our work is completely different from many alternative works on data placement [12, 10, 8] wherever the information question employment is additionally sculpturesque as a hypergraph and partitioning techniques are accustomed drive data placement selections. Ferhatosmanoglu et al. propose victimisation replication at the side of declustering for achieving best parallel I/O for spatial vary queries. Finally, there's a lot of work on finding tiny separators in data placement strategies. many theoretical results on better-known concerning this downside. We tend to discuss these connections in additional detail later once we describe our planned algorithms.

3.PROBLEM DEFINITION

DATA PLACEMENT ALGORITHMS

We present to algorithms for data placement with replication, with the goal to reduce the common question span, rather than ranging from scratch.

In a cluster wherever every node contains a native disk, it's economical to maneuver processing operations to nodes wherever application information are set. If information aren't domestically out there in an exceedingly process node, data have to be compelled to be migrated via network interconnects to the node that performs the data process operations. Migrating an outsized quantity of knowledge results in excessive network congestion, that successively will deteriorate system performance. HDFS permits Hadoop MapReduce applications to transfer process operations toward nodes storing application data to be processed by the operations.

In a heterogeneous cluster, the computing capacities of nodes might vary considerably. A high speed node will end process data hold on in an exceedingly native disk of the node quicker than low speed counterparts. when a quick node completes the process of its native computer file, the node should support load sharing by handling unprocessed information set in one or additional remote slow nodes. Once the number of transferred data thanks to load sharing is extremely massive, the overhead of moving unprocessed information from slow nodes to quick nodes becomes a vital issue moving Hadoop's performance. To spice up the performance of Hadoop in heterogeneous clusters, we tend to aim to reduce data movement between slow and quick nodes. This goal will be achieved by data placement scheme that distributes and stores data across multiple heterogeneous nodes supported their computing capacities. Data movement will be reduced if the quantity of file fragments placed on the disk of every node is proportional to the node's processing speed.

To achieve the simplest I/O performance, one might create replicas of input file of a Hadoop application in an exceedingly approach that every node in an exceedingly Hadoop cluster contains a native copy of the input file. Such an data replication theme will, of course, minimize data transfer among slow and quick nodes within the cluster throughout the execution of the Hadoop application. The data replication approach has many limitations. First, it's terribly high-priced to form replicas in an exceedingly large-scale cluster. Second, distributing an outsized range of replicas will prodigally consume scarce network data measure in Hadoop clusters. Third, storing replicas needs immoderately great amount of disk capability, that successively will increase the price of Hadoop clusters.

Although all replicas will be made before the execution of Hadoop applications, vital efforts should be create to scale back the overhead of generating replicas. If the data replication approach is used in Hadoop, one must address the matter of high overhead for making file replicas by implementing a coffee overhead file replication mechanism. as an example, Shen and Zhu developed a proactive low-overhead file replication theme for structured peer to look networks. Shen and Zhu's theme could also be incorporated to beat this limitation.

To address the on top of limitations of the data replication approach, we tend to ar that specialize in information placement methods wherever files are divided and distributed across multiple nodes in an exceedingly Hadoop cluster while not being duplicated. Our data placement approach doesn't need any comprehensive theme to alter data replicas.

Placement management mechanism, two algorithms are enforced and incorporated into Hadoop's HDFS. The primary algorithm is to distribute file fragments to heterogeneous nodes in an exceedingly cluster . Once all file fragments of AN input data needed by computing nodes are out there in an exceedingly node, these file fragments are distributed to the computing nodes. The second data placement algorithm is employed to reorganize file fragments to unravel the data skew downside. There two cases during which file fragments should be organized. First, new computing nodes are else to an existing cluster to possess the cluster dilated. Second, new data is appended to an existing input data. In each cases, file fragments distributed by the initial data placement formula will be non continuous.

4.INITIAL DATA PLACEMENT

The initial data placement formula begins by first dividing an outsized input data into variety of even sized fragments. Then, the data placement formula assigns fragments to nodes in an exceedingly cluster in accordance to the nodes' processing speed. Compared with low-performance nodes, superior nodes are expected to store and method additional file fragments. allow us to contemplate a MapReduce application and its input data in an exceedingly heterogeneous Hadoop cluster. In spite of the nonuniformity in node process power, the initial data placement theme must distribute the fragments of the input file in order that all the nodes will complete process their native data at intervals nearly an equivalent time.

In our experiments we tend to determined that the computing capability of every node is sort of stable certainly tested Hadoop applications, as a result of the time interval of those Hadoop applications on every node is linearly proportional to computer file size. As such, we are able to quantify every node's process speed in an exceedingly heterogeneous cluster employing a new term known as computing magnitude relation. The computing magnitude relation of a computing node with relation to a Hadoop application will be calculated by identification the applying. it's price noting that the computing magnitude relation of a node might vary from application to application.

5.DATA DISTRIBUTION

Input file fragments distributed by the initial information placement formula may be noncontinuous thanks to the subsequent reasons:

(1) new data is appended to existing input file;

(2) data blocks are deleted from the prevailing input file; and

(3) new data computing nodes are else into AN existing cluster.

To address this dynamic data load equalization downside, we tend to enforced an data distribution formula to reorganize file fragments supported computing ratios.

The data distribution procedure is delineate because the following steps.

First, like initial data placement, data concerning the topology and disc space utilization of a cluster is collected by the data distribution server. Second, the server creates 2 node lists: an inventory of nodes during which the quantity of native fragments in every node exceeds its computing capability and an inventory of nodes which will handle additional native fragments attributable to their high performance. The primary list is termed over utilised node list; the second list is termed as under-utilized node list. Third, the information distribution server repeatedly moves file fragments from over-utilized node to underutilized node till the data load are equally distributed. Method of migrating data between a try of AN over-utilized and an underutilized nodes, the server moves file fragments from a supply node within the over utilised node list to a destination node within the underutilized node list. Note that the server decides the quantity of bytes instead of fragments and moves fragments from the supply to the destination node. The on top of data migration method is perennial till the quantity of native fragments in every node matches its speed measured by computing magnitude relation.

The Data distribution formula

1. Get the Network Topology, calculate the computing magnitude relation and utilization
2. Build and sort two lists: under-utilized node list and over-utilized node list
3. choose the supply and destination node from the separate lists
4. Transfer data from supply node to destination node
5. Repeat step 3, 4 till any list is empty

6.PROPOSED DYNAMIC DATA PLACEMENT POLICY

In a heterogeneous cluster, the computing capability for every node isn't an equivalent. Moreover, for various varieties of job, the computing capability magnitude relation of nodes are not an equivalent. Therefore, a Dynamic Data Placement (DDP) strategy is given consistent with the categories of jobs for adjusting the distribution of knowledge blocks.

The planned algorithm, namely DDP, consists of two main parts: the primary phase is performed once the computer file are written into the HDFS, and also the second part is performed once employment is processed.

7.IMPLEMENTATION METHOD

RATIO TABLE

When Hadoop starts, a RatioTable is formed within the NameNode, that is employed to work out the allocation magnitude relation of knowledge blocks in nodes once the Data is written into the HDFS, and is employed to work out whether or not the information blocks should be reallocated once the task is dead. The RatioTable records the categories of jobs and also the computing capability magnitude relation of every node. The computing capability of every node relies on the common execution time of one task in this node. The NameNode calculates the computing capability magnitude relation for every node consistent with the task execution time come by the heartbeat message of every DataNode.

STAGE1

When data are written into the HDFS, NameNode 1st checks the RatioTable. These data are accustomed verify whether or not this sort of job has been performed. If the RatioTable contains a record of this job, the fresh written data are going to be allotted to every node in accordance with the computing capability that records within the RatioTable. If the RatioTable has no record of this job, the data are going to be equally distributed to the nodes within the cluster, and also the NameNode can add a brand new record of this sort of job within the RatioTable. every node's computing capability are going to be set at one for this sort of job.

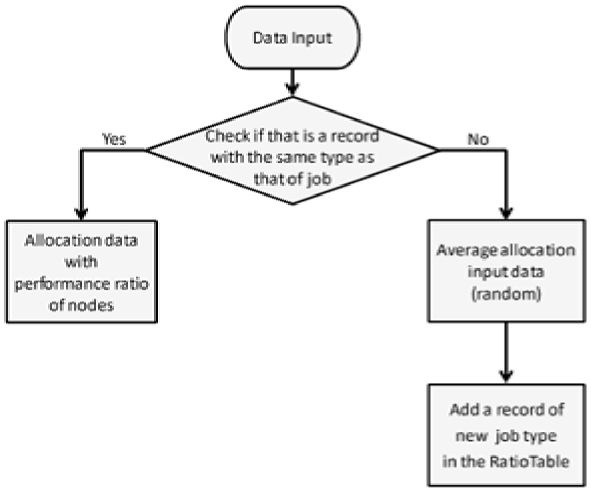
Data Input
Check if that is a record with the same type as that of job
Yes
No
Allocation data with performance ratio of nodes
Average allocation input data (random)
Add a record of new job type in the RatioTable

Fig.1. The flow chart of stage1.

If the performed job is TeraSort, the NameNode can check the RatioTable and notice no record of TeraSort. during this case, the data are going to be equally distributed to the 3 nodes, a record for TeraSort is then created for the RatioTable, and also the computing capability among Node A, Node B and Node C for TeraSort to be set at 1:1:1. Fig.1 shows the flow chart of Algorithm1(Stage one).

STAGE 2

Stage 2 starts once the task begins execution; because the job starts execution, every node can 1st receive the primary batch of tasks. once the task finishes execution in every DataNode, all DataNodes can come the task execution time to the NameNode. The NameNode calculates the computing capability magnitude relation of this job for every node consistent with those execution time. However, every node contains a completely different range of task slots. in an exceedingly DataNode, the tasks in task slots will be processed in parallel. This causes the computing capability magnitude relation that's calculated supported the task execution time to be inaccurate. Therefore, the task execution time needs calculative the magnitude relation of computing capability for the nodes, and also the range of task slots should be thought-about. Therefore, the computing capability adopts the common time needed to finish one task.

The NameNode can use the Tt(X) of every node X to calculate the computing capability magnitude relation of every node. when the Name Nodecalculates the computing capability magnitude relation, it'll compare the record of the RatioTable. If the computing capability magnitude relation and record ar an equivalent, then it'll not be transferred to any data blocks. However, if they're not same, then data blocks are going to be transferred consistent with the new magnitude relation calculated by the NameNode, and also the NameNode can modify the record at the RatioTable. The transferred information blocks are processed within the background, and also the Hadoop job doesn't have to be compelled to wait till the information transfer is completed to be dead.

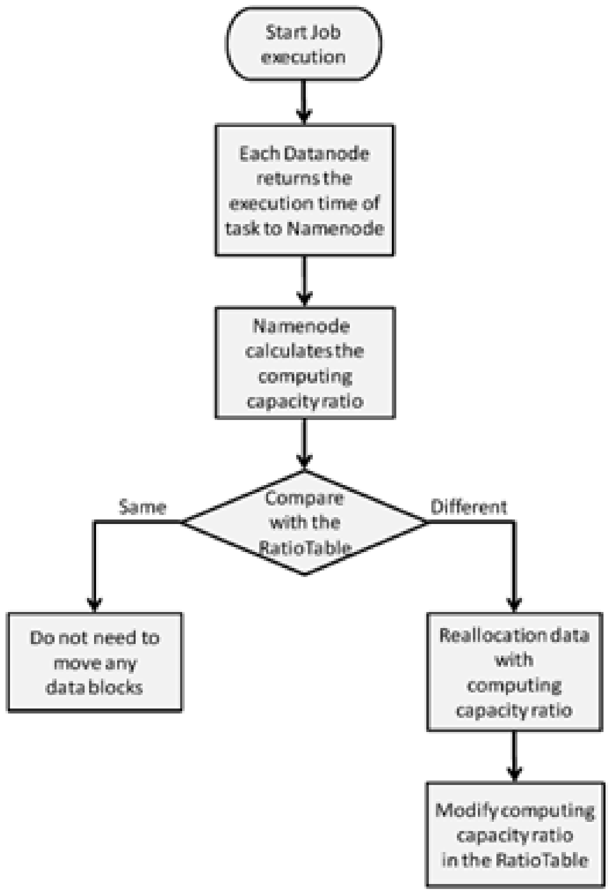
Start Job execution
Each Datanode returns the execution time of task to Namenode
Namenode calculates the computing capacity ratio
Compare with the RatioTable
Same
Different
Do not need to move any data blocks
Reallocation data with computing capacity ratio
Modify computing capacity ratio in the RatioTable

Fig.2 is that the flow chart of Algorithm2(stage2).

Table 1 Five Nodes in a Hadoop Heterogeneous Cluster

Sno	Node	Node Type	CPU Type	CPU Speed	RAM (GB)	Cache (L1) MB	Network Bandwidth	HardDisk	OS
1	Node A	Master	Core 2 Due		2	1	10 Mbps	1 TB	Fadora
2	Node B	Slave	Core 2 Due		4	1	10 Mbps	1 TB	Fadora
3	Node C	Slave	Core 2 Due		2	1	10 Mbps	1 TB	Fadora
4	Node D	Slave	Core 2 Due		1	1	10 Mbps	1 TB	Fadora
5	Node E	Slave	Core 2 Due		1	1	10 Mbps	1 TB	Fadora

9.RESULT

WordCount and Grep are two varieties of jobs run to guage the performance of the planned algorithm in an exceedingly Hadoop heterogeneous cluster. WordCount and Grep are MapReduce applications running on a Hadoop cluster. WordCount is, application used for reckoning the words within the input data, and Grep is employed to look for normal expressions. In every spherical, ten jobs are run at the same time, and every job processes completely different computer file severally, during which the dimensions of all input file is 1 GB. The experimental data are run in ten rounds during which every round runs ten jobs, to average the execution time.

First, the execution time of WordCount and Grep are measured for every node to perform completely different sizes of knowledge. Fig.3 shows that the common execution time of one GB and 2GB WordCount job severally. Fig.4 shows the common execution time of one GB and a couple of GB Grep job severally. As shown in Fig.3 and Fig.4, the performed information size doesn't have an effect on the computing capability magnitude relation between nodes. The execution time of every node is proportional to the information size.The data shown in Fig.3and Fig.4 ar adopted to calculate the computing magnitude relation of every node for 2 varieties of jobs.

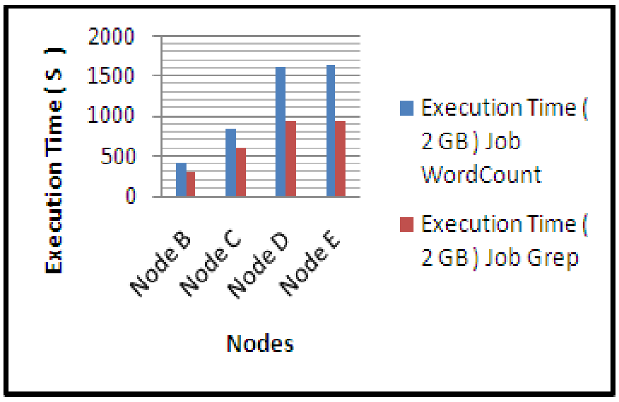

Fig.3. ExecutiontimeofWordCount&amp; Grep job (one GB) on every node.

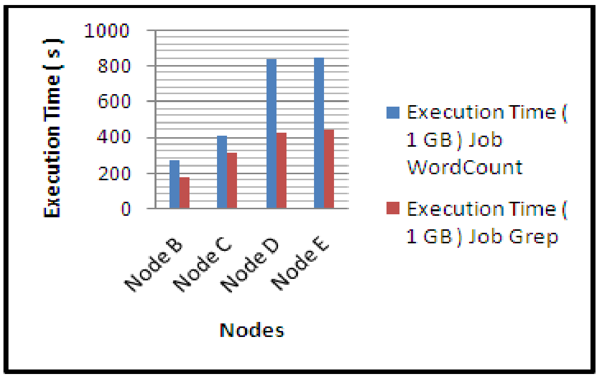

Fig.4.ExecutiontimeofWordCount&amp; Grep job (a pair of GB) on every node.

According to the experimental results, in spite of the dimensions of knowledge to be processed, every node’s computing capability magnitude relation is maintained. For a WordCount job, supported the common time of 1 job execution, Node B is fourfold quicker than NodeD, and NodeC is 2 times quicker than Node D.
However, once Hadoop perform jobs, it spends beyond regular time to start out up and finish off and tasks aren't solely map tasks however also includes reducing tasks. The planned
algorithm in the main involves the a part of the map tasks to optimize. However, consistent with the task execution time, calculative the computing magnitude relation might cause slight variations. Therefore, completely different jobs are thought-about to live the common time needed to finish one task.

Fig.5 shows the common time needed to finish one WordCount task &amp; Grep task with 64MB Block Size . As shown in Fig.5, the execution time offset of every node for execution the native. In spite of computing capability, the time offset of execution native tasks depends on the dimensions of an information block.
When the dimensions of an data block is about to be larger, the time offset of execution an area task is far larger. Consistent with the experimental results, the task execution time is proportional with the data block size. Therefore, once calculative the computing capability magnitude relation of every node, the task execution time is adopted. The setting of an data block size doesn't have an effect on the computing magnitude relation of every node. Therefore, the experimental information block size involves victimisation the Hadoop default sixty four MB.
The computing capability magnitude relation of the common time needed to finish one task. This magnitude relation is calculated consistent with the experimental results shown in Fig.5. For a WordCount job, the common time needed to finish one task, Node B is four and a 4 times quicker than Node C, Node B is a pair of times quicker than Node C. For a Grep job, Node B is 3 and a 4 times quicker than Node C , Node B is a pair of times quicker than Node C. This magnitude relation are slightly completely different from the magnitude relation calculated with job execution time, as a result of the task isn't solely map task additionally includes alternative actions. Therefore, we tend to adopt the magnitude relation of records to treat because the computing capability magnitude relation.
Figs.7 represent the common execution time of a Word-Count and a Grep job, compared with the planned formula and another 3 data allocation methods. during this experiment, we tend to used the dimensions of every data as a pair of GB, and also the data block size is about at 64 MB. every spherical executes ten jobs, and perform a complete of 5 rounds during which every job processes completely different information files. Finally, the common execution time of 1 job is employed for analysis.
As shown in Fig.7, the planned DDP algorithm is compared with the Hadoop default strategy, the simplest case data distribution, and also the worst case data distribution. The simplest case of distribution represents all of the task process native data during which no information are needed to be transferred. Therefore, data blocks are allotted in accordance with the magnitude relation recorded in. as an example, suppose that a WordCount job form of data should be written into the HDFS. These data will be divided into seventeen data blocks, and also the magnitude relation recorded in is 4.5, 2, 1, 1. Therefore, if data blocks are allotted consistent with the magnitude relation, then Node B is appointed 9 information blocks, Node C is appointed to four blocks, Node C and Node E are every appointed 2 blocks. The worst case indicates that every one of data blocks are targeting an equivalent node in order that alternative nodes playacting tasks should wait till information blocks are transferred from the node storing all data blocks, and

also the node computing capability should be the slowest. Therefore, the worst case is that the information blocks being placed in Node D or Node E.

Fig.7 shows the various varieties of information placement policy impacts on WordCount job execution time. The common execution time of DDP is closed to the best case. even though victimisation our approach the primary time to perform this sort of job doesn't follow computing capabilities to allot data blocks, it should lead to a small delay at the start of execution, however the planned formula can change the situation of knowledge blocks once job playacting, and when this, the data of this job kind to be written into the HDFS are going to be allotted consistent with the computing magnitude relation. Therefore, the common execution time of the planned algorithm and also the best case are nearly identical. As shown in Fig.8, the comparison with Hadoop default strategy, DDP will scale back execution time or so 14.5%. As shown in Fig.8, DDP compared with the worst case will improve by or so 24.7%.

Fig.8 shows the comparison of various information placement policies on Grep job execution time. For the Grep job, when victimisation the planned algorithm, the common execution time is additionally very near the simplest case. As shown in Fig.8, compared with the Hadoop, the DDP will scale back the execution time by or so 23.5%. As shown in Fig.8, compared with the worst case, the DDP will improve by approximately 32.1%.

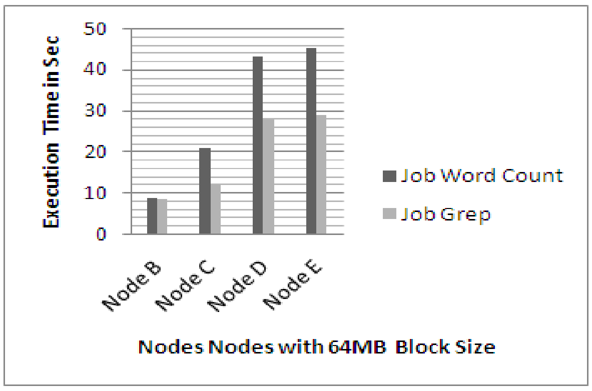

Fig 5. Time needed to execute with sixty four MB Block Size of HDFS

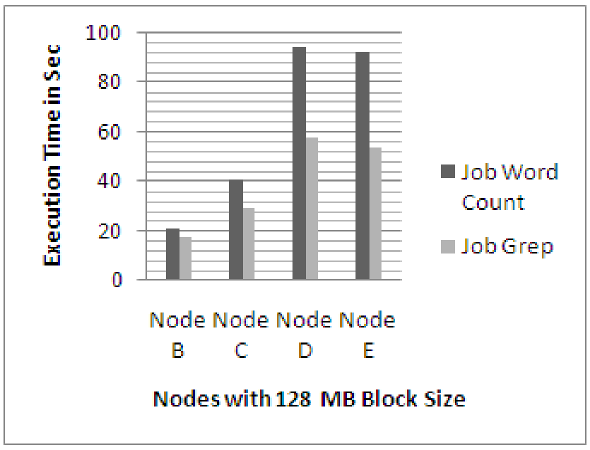

Fig. 6 Time needed to execute with 128 MB Block Size of HDFS

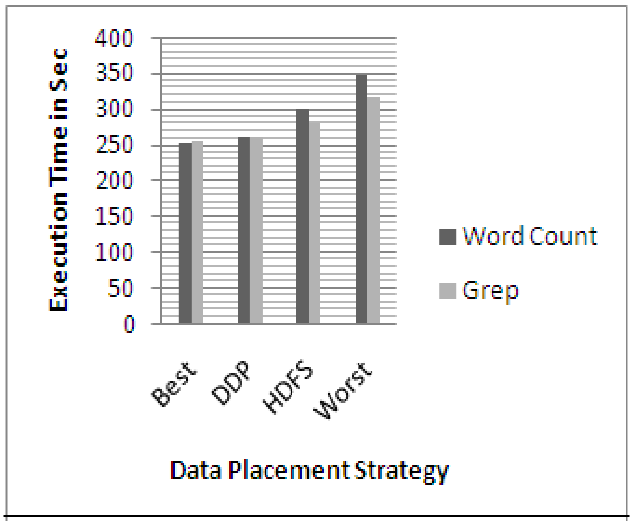

Fig 7 Time to Execute WordCount & Grep with Different Placement Strategy

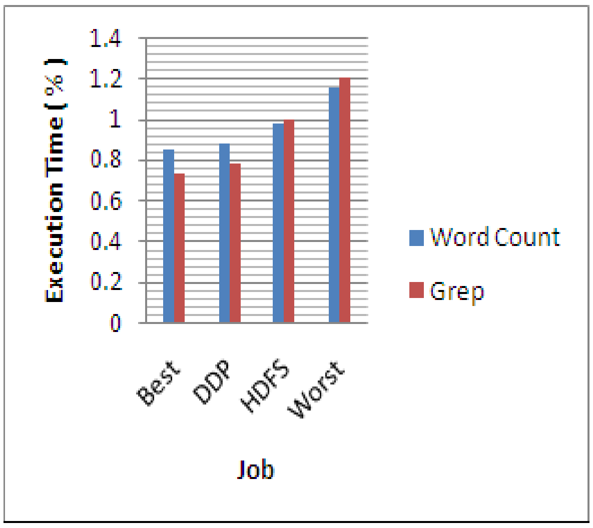

Fig 8 The percentage of execution time of a job compare with Hadoop default strategy.

10.CONCLUSION

In this paper, we tend to solve the combined downside of knowledge placement and replication, analyzed the issues of data placement and duplicate choice for this metric, and player connections to many well-studied speculative ideas. We tend to used these connections to develop a series of algorithms to unravel this downside, and our in depth experimental analysis over many datasets incontestible that our algorithms may result in forceful reductions in average execution. We tend to are going to extend our add many completely different directions. As we tend to mentioned earlier, we tend to believe that temporal programing algorithms will be accustomed correct the load imbalance that will result from optimizing for question span alone; though analysis tasks ar typically not latency sensitive, there are still typically deadlines that require to be glad. we tend to decide to study a way to incorporate such deadlines into our framework. we tend to are going to study a way to with efficiency track changes within the question employment nature on-line, and the way to adapt the replication selections on-line.

11.FUTURE WORK

Future directions of our analysis embrace applying machine learning approaches to check the resource usage patterns of jobs and model their power usage characteristics to with efficiency schedule them on the cluster. Upon submission of employment, create use of the information concerning data set to predict the data layout for its data blocks and optimally schedule the tasks creating use of knowledge section. i'd additionally prefer to study the performance of our algorithm once given with differing kinds of workloads like processor intensive, memory intensive, disk intensive and network intensive. We tend to decide to notice best values of MapReduce configuration parameters in order that our formula will give even higher energy savings. I additionally decide to create use of multiple heuristics like interval and minimum replication issue of a file, whereas rebalancing the cluster to avoid overhead on the network. Our future work can take another factors into thought supported our current work, e.g., a way to pre-partition the data nodes into completely different teams dynamically and attain energy-efficient block placement consistent with their period of time utilization and employment state.

REFERENCES

[1]. Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/.

[2]. Hadoop, http://hadoop.apache.org/.

[3]. Hadoop Distributed File System, http://hadoop.apache.org/docs/stable/hdfs_design.html.

[4]. Hadoop Yahoo, http://www.ithome.com.tw/itadm/article.php?c=49410&s=4.

[5]. T.Chao, H.Zhou, Y.He, and L.Zha. A Dynamic MapReduce Scheduler for Heterogeneous Workloads. IEEE Computer Society, 2009.

[6]. Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou, „„Workload Characterization on a Production Hadoop Cluster: A Case Study on Taobao,‟ ‟ in Proc. IEEE IISWC, 2012, pp. 3-13.

[7]. Q. Chen, D. Zhang, M. Guo, Q. Deng, S. Guo, SAMR: a self-adaptive MapRe-duce scheduling algorithm in heterogeneous environment, in: 2010 IEEE 10th International Conference on Computer and Information Technology (CIT), IEEE, 2010, pp.2736–2743

[8]. Chao Tian, Haojie Zhou, Yongqiang He, and Li Zha. “A Dynamic MapReduce Scheduler for Hetergeneous Workloads”, Eighth International Conference on Grid and Cooperative Computing, 2009. GCC '09. pp. 218-224, Aug. 2009.

[9]. D. Borthakur, K. Muthukkaruppan, K. Ranganathan, S. Rash, J.-S. Sarma, N. Spiegelberg, D. Molkov, R. Schmidt, J. Gray, H. Kuang, A. Menon, A. Aiyer, Apache Hadoop goes realtime at Facebook, in: SIGMOD ’11, Athens, Greece, June 12–16, 2011.

[10]. Q. Chen, D. Zhang, M. Guo, Q. Deng, S. Guo, SAMR: a self-adaptive MapRe-duce scheduling algorithm in heterogeneous environment, in: 2010 IEEE 10th International Conference on Computer and Information Technology (CIT), IEEE, 2010, pp.2736–2743.

[11]. B. He, W. Fang, Q. Luo, N. Govindaraju, T. Wang, Mars: a MapReduce framework on graphics processors, in: ACM 2008, 2008, pp.260–269.

[12]. G. Lee, B.G. Chun, R.H. Katz, Heterogeneity-aware resource allocation and scheduling in the cloud, in: Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud, vol.11, 2011.

[13]. C. Tian, H. Zhou, Y. He, L. Zha, A dynamic MapReducescheduler for heteroge neous workloads, in: Eighth International Conference on Grid and Cooperative Computing, GCC’09, IEEE, 2009.

[14]. M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving MapReduce performance in heterogeneous environments, in: Proc. OSDI, San Diego, CA, December 2008, pp.29–42.