Asian Journal of Information Technology 15 (15): 2645-2655, 2016
ISSN: 1682-3915

Efficient Map Reduce Task Scheduling and Micro-Partitioning Mechanism for Optimizing Large Data Analysis

[1]A. Kamaleswari and [2]P. Thangaraj
[1]Department of Computer Application, Bannari Amman Institute of Technology, Sathyamangalam, Erode, Tamil Nadu, India
[2]Department of Computer Science and Engineering, Bannari Amman Institute of Technology, Sathyamangalam, Erode, Tamil Nadu, India

Abstract: Virtualization is a key technology to enable cloud computing. A vast mass of popular content is transferred frequently across network links in the cloud. In network-level Redundancy Elimination (RE) techniques, it minimizes traffic flow on bandwidth-constrained network paths by eliminating the transmission of repetitive byte sequences. In previous research, the protocol is independent of redundancy elimination which cannot eliminate duplicate packets from within arbitrary network flows. In emerged cloud we require a potent technique to improve the performance of network links in the face of frequent data. Our proposed research present the packet reduction technique is used for finding and removes the duplicate packets in a network environment. In addition, the data itself can be moreover large to store on a single machine. In order to reduce the time it takes to process the data and to have the storage space to store the data, we introduce a new approach called map reduce method. In this approach, it has to separate the workload among computers in a network. As an outcome, the performance of map reduce robustly determined on how equal it distributes this workload among the computer. In map reduce, workload allocation depends on the algorithm that separating the data. To avoid the issues of uneven distribution of data we use data sampling. By using the partitioning mechanism, the partitioning is done on the data which depends on how huge and representative the sample is and on how well the samples are examined. In addition to that we use partitioning methods to divide the workload into small tasks that are dynamically scheduled at runtime based on deadline. To improve the accuracy in scheduling, we propose a novel method called deadline constraints based task scheduling algorithm in map reduce. This method allows the user to specify a job's deadline and attempts to formulate the job to be completed before the deadline. This method is simple and efficient systems with high-throughput, low-latency task schedulers and proficient data materialization.

Key words: Map reduce, redundancy elimination, partitioning, deadline, scheduling

INTRODUCTION

Cloud computing has significantly transformed the method various critical services are delivered to customers for instance, the software, platform and infrastructure as a service models and by the same time has raised new problems to data centers. The outcome is a complete, innovative generation of large scale infrastructures, carrying an unprecedented level of workload and server consolidation that require new programming representations, management procedures and hardware platforms. Simultaneously, it provides remarkable capabilities to the mainstream market, thus, offering opportunities to build new services that need large scale computing. Hence, data analytics is one of the more important fields that can advantage from next generation data center computing. The intersection among cloud computing and next generation data analytics services indicates towards a future in which large amounts of data are available and users will be capable to process this data to create high value services. As a result, building new models to implement such applications and mechanisms to maintain them are open challenges. An instance of a programming representation specially well-suited for large-scale data analytics is map reduce, initiated by Google in 2004.

At the present time cloud computing is developing its services to data-intensive computing on distributed platforms like map reduce, dryad and hadoop. In namely distributed models on clouds, physical machines are

Corresponding Author: A. Kamaleswari, Department of Computer Application, Bannari Amman Institute of Technology, Sathyamangalam, Erode, Tamil Nadu, India

virtualized and a huge range of Virtual Machines (VMs) form a virtual cluster. Execution time aims of map reduce jobs play a significant function in achieving higher revenues or utility for the content suppliers. Depending on whether or not a task is allocated to a node by its input data (local task or non-local task), the execution time of the task capacity differs considerably. However, to boost locality when a task is scheduled, the computing node with the equivalent data must have free computing slots to execute the task. If not the task has to be allocated to a remote node which needs remote data transfer from another node for the task to execute. The challenge that requires to be considered is how efficiently, we can schedule the jobs such that both data locality and deadline conditions are fulfilled.

Map reduce workloads commonly include a very huge number of small computations executing in parallel. High levels of computation partitioning and moderately small individual tasks are a planning point of map reduce models (Dean and Ghemawat, 2008). At the same time as it was innovatively used mainly for batch data processing, it utilize has been expanded to distribute, multi-user environments in which submitted jobs may have execute in different priorities (Zaharia *et al*., 2010). This transform builds scheduling even more relevant. Task selection and slave node assignment direct a job prospect for development and thus control job performance.

One of the propose objectives of the map Reduce framework is to enhance data locality over working sets (Polo *et al*., 2010a) in an try to minimize network bottlenecks and improve overall system throughput. Data locality is attained when data are saved and developed with the same physical nodes. Failure to develop locality is one of the famous shortcomings of most multi-job map reduce schedulers while placing tasks from various jobs on the similar nodes will contain a negative achieve on data locality (Polo *et al*., 2010b).

All together, there is a development towards the adoption of heterogeneous hardware and hybrid systems in the computing industry. Heterogeneous hardware will be controlled to enhance both performance and energy consumption, developing the best features of every platform. For instance, a map reduce framework facilitated to run on hybrid systems has the possible to have considerable impact on the future of several fields, involving financial analysis, healthcare and smart cities-style data management. Map reduce offers an easy and a suitable method to expand massively shared data analytics facilities that utilize all the computing power of these large-scale services. A large cluster of hybrid a lot of core servers will carry workload consolidation approaches one action closer in future data centers.

Further, map reduce recommended as a service in the cloud provides an attractive usage model for enterprises. A map reduce cloud service will permit enterprises to cost-effectively analyse large volumes of data without creating large infrastructures of their own. Using Virtual Machines (VMs) and storage hosted in the cloud, enterprises can basically create virtual map reduce clusters to analyse their data.

A significant challenge for the cloud provider is to maintain multiple virtual map reduce clusters executing simultaneously, a diverse set of jobs on distributed physical machines. Concretely, every map reduce job produces various loads on the distributed physical infrastructure computation load: number and size of each VM (CPU, memory), storage load: amount of input, output and intermediate data and network load: traffic generated during the map, shuffle and reduce phases. The network load is of special concern with map reduce as large volumes of traffic can be produced in the shuffle phase when the output of map tasks is transferred to reduce tasks. As every reduce task requires to read the output of all map tasks, an unexpected explosion of network traffic can considerably deteriorate cloud performance. This is particularly true when data has to traverse a greater number of network hops as going across racks of servers in the data center.

To minimize network traffic for map reduce workloads, we dispute for improved data locality for mutually map and reduce phases of the job. The aim is to minimize the network distance among storage and compute nodes for mutually map and reduce processing for map phase, the VM executing the map task must be close to the node that stores the input data (favourably local to that node) and for reduce phase, the VMs executing reduce tasks should be close to the map-task VMs which produce the intermediate data utilized as reduce input. Enhance the data locality in this manner is valuable in two ways it minimizes job execution times as network transfer times are large components of total execution time and it minimizes cumulative data center network traffic. At the same time as map locality is well understood and executed on map reduce systems, minimize locality has surprisingly received little concentration in spite of its important potential impact.

In a map reduce permitted computing cloud, a map reduce cluster is setup by different VMs organized in the cloud data center. We transfer to the VMs that host the map reduce nodes as the map reduce instances and the VMs that host another cloud application as the non-map reduce instances. To minimize the maintenance cost of the cloud data center, VMs' workload is combined by correctly grouping well-suited VMs collectively and allocating them to the appropriate physical servers. But, combining the VM workload will warm up the competition

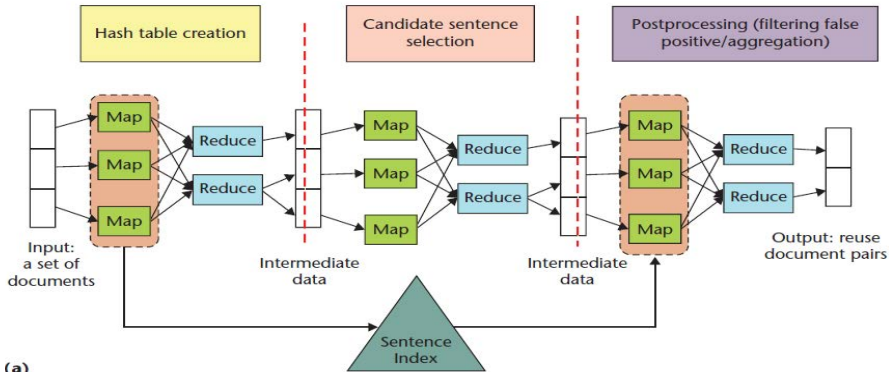

Fig. 1: Map reduce-based parallelization of the reuse detection workflow

for hardware resource between VMs. As the parallel computing environment of the map reduce model dictates that the performance of the map reduce applications depends on the slowest map reduce instances in the cloud data center and it is also very responsive to the I/O bandwidth (e.g., disk I/O, network bandwidth) competition between other co-located non-map reduce instances. Consequently, map reduce instances must be hosted by homogeneous and remote VMs that have contributed I/O bandwidth engaged separately from other non-map reduce VMs. Therefore, it may not be useful for the cloud providers to create such remote VMs while the reserved I/O bandwidth cannot be distributed between other VMs. An innovative VM consolidation method specifically designed for map reduce allowed computing clouds which consolidate VMs based on the features of the map reduce model is required.

The early map reduce framework was planed for off-line data processing. Further, at present it is popularly functional in heterogeneous, sharing and multi-user models. The study of the map reduce scheduling algorithm mainly in four parts: the data locality of the map reduce tasks. It is the result of the data allocation to task scheduling; fault-tolerant scheduling and expectation execution time in a heterogeneous model; resource sharing: for the Hadoop cluster how to distribute the computing resources through scheduling the user groups; resource aware scheduling algorithm. It is based on the significance of the cluster resources, namely memory, disk IO, network and other factors; real-time scheduling. It is the study of the map reduce real-time scheduling environment. At the present time, the map reduce scheduling algorithms mainly involve FIFO (First Input First Output), LATE (Longest Approximate Time to End) (Zaharia *et al*., 2010), fair scheduler (Zaharia *et al*., 2010) and capacity scheduler. Fundamental features such as data locality, user priority, fault-tolerant and fairness are all considered with these algorithms.

Furthermore, few algorithms have considered the user's job deadline constraints such as in the flexible cloud computing environment or online service system (Alexandraki and Paterakis, 2005). During the job deadline, we can build a model to advance the reliability of the task remaining time estimating in the heterogeneous environment, formulate the use jobs can be completed before the deadline.

The remarkable growth in the computer networks increase the network bandwidth available to users has resulted in the generation of large volumes of data. For a single computer the volume of data produced is too large so that the computation of the data obtains more time and also increases the storage space. In order to minimize the computation time and reduces the storage space the workload is shared on two or more computers. In addition to that allocation of workloads depends upon the node capacity so that we realize the effective scheduling. The essential property that some of the content on the internet is highly admired results some data being frequently transferred across the network. Our proposed systems challenge to improve the efficiency of the network by removing these redundant transfers.

Literature review: Candan *et al*. (2011) proposed a rank loud, a scalable, ranked media, query-processing system. Rank loud avoids waste by intelligent partitioning the data and allocating it on available resources to minimize the data replication and indexing overheads and to prune superfluous low-utility processing.

As shown in Fig. 1, the last phase of the operation involves post processing to eliminate false positives. When attempting to parallelize the underlying workflow using a map reduce based framework, however, we have seen that the benefits of using more servers drop quickly for high target-recall rates. This drop in the scalability is primarily due to the number of unnecessary candidates that are being generated and processed. In short, in many

large-scale media-processing applications such as content-overlap analysis to be effective, available resources must be allocated to pay the most attention to the most promising and relevant (that is highest-utility) words, sentences, paragraphs or other media features. To avoid waste and achieve the scalability needed for large-scale media processing and mining, data-processing systems must employ data partitioning and resource allocation strategies that can prune unpromising data objects from consideration without having to use available resources to enumerate results that will be eventually eliminated.

Josh Rosen and Bill Zhao the research, we will explore the challenges of skew and stragglers, survey existing techniques to mitigate them and explore how fine-grained micro-tasks can effectively mitigate skew. Partitioning skew is caused by uneven map output partition sizes/record counts which may be caused by poor choices of partitioning functions. We introduce our approach for avoiding skew and stragglers during the reduction phase. The key technique is to run a large number of reduce tasks, splitting the map output into many more partitions than reduce machines in order to produce smaller tasks. These tasks are assigned to reduce machines in a "just-in-time" fashion as workers become idle, allowing the task scheduler to dynamically mitigate skew and stragglers.

Running many small tasks lessens the impact of stragglers for the work that would have been scheduled with slow nodes when using coarse-grained tasks can now be performed by other idle workers. With large tasks, it can be more efficient to exclude slow nodes rather than assigning them any study. By assigning smaller units of study, jobs can derive benefit from slower nodes. Micro-tasks can also help to mitigate skew. Increasing the number of hash partitions generally leads to smaller, more-even partitions because there is a lower probability of collision in the key's partitioning function. This does not provide the same partitioning quality guarantees as a system that manually partitions the most expensive keys but it has a high probability of producing even partitions. For inputs containing few distinct keys, fine-grained partitioning may result in many empty reduce tasks that receive no data. These empty reduce tasks are unproblematic, since they can be easily detected and ignored by the scheduler. Jobs with few distinct keys are the most sensitive to partitioning skew, since there may not be enough other study to mask the effects of a straggling task created by a key collision in the hash partitioning function. For jobs with large numbers of distinct keys, the impact of key collisions is small. Liu H, Orban D describes how we implement Cloud mapreduce using the Amazon cloud OS. We start with the high level architecture and then delve into detailed implementation issues we have encountered. We use the Word Count application as an example to describe our implementation. We use four infrastructure services that Amazon provides today. We use EC2 APIs to spawn up new virtual machines (also called instances) to process new map reduce jobs. We store our input and possibly output data in S3. By leveraging the distributed nature of S3, we can achieve higher data throughput since data come from multiple servers and communications with the servers potentially all traverse different network paths. We also use SQS which is a critical component that allows us to design mapreduce in a simple way. A queue serves two purposes. First, it is a synchronization point where workers (a process running on an instance) can coordinate job assignments. Second, a queue serves as a decoupling mechanism to coordinate data flow between different stages. There are several SQS queues: one input queue, one master reduce queue, one output queue and many reduce queues. As its name implies, the input queue holds the inputs to the map reduce computation. The map reduce framework collects the output key value pairs from the map function, then writes them to the reduce queues. A reduce key maps to one of the reduce queues through a hash function. A default hash function is provided but the users could also supply their own. Once the map workers finish their jobs, the reduce workers start to poll work from the master reduce queue. Once a reduce worker dequeues a message, it is responsible for processing all data in the reduce queue indicated by the message.

Edward Bortnikov introduces a map reduce system can simultaneously run multiple jobs competing for the node's resources and traffic bandwidth. These conflicts cause slowdown in the execution of tasks. The duration of each phase and hence the duration of the job is determined by the slowest or straggler, task. We use the slowdown metric the ratio between the task's execution time and the median running time of a sibling task in the same job to characterize stragglers.

We address the above shortcomings by introducing slowdown predictor a novel machine learned oracle component that detects potential or existing bottlenecks in MR clusters based on patterns mined from historical performance data. The oracle exposes a simple API: given a task-node pair ht, ni, it produces an estimate for the slowdown of t running on n. In this context, both the task and the node are modeled as feature vectors assembled from MR-level and system metrics. Slowdown prediction can serve both the scheduling and the speculative execution scenarios. Consider, for example, augmenting Hadoop's capacity scheduler with a predictor oracle. The

scheduler manages a pool of execution slots at each node and assigns the free slots to tasks waiting for execution. In its current form, it will always use a slot, should a task be waiting for it. With a predictor's help, the scheduler can avoid creating task to- node assignments for which the slowdown estimate is high. For example, it can prevent allocating congested hardware to resource-savvy tasks (despite the existence of free slots), hence avoiding the emergence of stragglers.

The predictor abstraction generalizes the currently existing heuristics while being far better amenable to tuning for specific workloads. In this study, we demonstrate a specific predictor implementation which is trained on a Hadoop performance dataset collected at Yahoo!. Our evaluation shows that slowdown can be effectively predicted, especially for mappers, thanks to the dominance of large periodic jobs in production environments.

Gonzalez *et al.* (2012) to address the challenges of power-law graph computation, we introduce the power graph abstraction which exploits the structure of vertex-programs and explicitly factors computation over edges instead of vertices. As a consequence, power graph exposes substantially greater parallelism, reduces network communication and storage costs and provides a new highly effective approach to distributed graph placement. We describe the design of our distributed implementation of power graph and evaluate it on a large EC2 deployment using real-world applications. We introduced the power graph abstraction which exploits the Gather-Apply-Scatter Model of computation to factor vertex-programs over edges, splitting high-degree vertices and exposing greater parallelism in natural graphs. We then introduced vertex-cuts and a collection of fast greedy heuristics to substantially reduce the storage and communication costs of large distributed power-law graphs.

Hammoud *et al.* (2012) investigate the problems of data locality and partitioning skew in Hadoop. We propose Center-of-Gravity Reduce Scheduler (CoGRS), a locality-aware skew-aware reduce task scheduler for saving mapreduce network traffic. In an attempt to exploit data locality, CoGRS schedules each reduce task at its center-of-gravity node which is computed after considering partitioning skew as well. mapreduce, this study explores the locality and the partitioning skew problems present in the current Hadoop implementation and proposes Center-of-Gravity Reduce Scheduler (CoGRS), a locality-aware skew-aware reduce task scheduler for mapreduce. CoGRS attempts to schedule every reduce task, R, at its center-of-gravity node determined by the network locations of R's feeding nodes and the skew in the sizes of R's partitions. The network is typically a bottleneck in mapreduce-based systems. By scheduling reducers at their center-of-gravity nodes, we argue for reduced network traffic which can possibly allow more mapreduce jobs to co-exist on the same system. CoGRS controllably avoids scheduling skew, a situation where some nodes receive more reduce tasks than others and promotes pseudo-asynchronous map and reduce phases.

Hammoud and Sakr (2011) describe Locality-Aware reduce task Scheduler (LARTS), a practical strategy for improving map reduce performance. LARTS attempts to collocate reduce tasks with the maximum required data computed after recognizing input data network locations and sizes. LARTS adopts a cooperative paradigm seeking a good data locality while circumventing scheduling delay, scheduling skew, poor system utilization and low degree of parallelism. We propose a novel strategy, LARTS which applies data locality to reduce task scheduling in mapreduce. We empirically analyze Hadoop's performance and network traffic. We observe that the process of interleaving the execution of map tasks with the shuffling of partitions employed by native Hadoop improves performance but increases network traffic. We show how LARTS manages to maintain the advantage of the interleaving process besides diminishing network traffic.

Hindman *et al.* (2011) propose Mesos, a thin resource sharing layer that enables fine-grained sharing across diverse cluster computing frameworks by giving frameworks a common interface for accessing cluster resources. The main design question that Mesos must address is how to match resources with tasks. This is challenging for several reasons. First, a solution will need to support a wide array of both current and future frameworks, each of which will have different scheduling needs based on its programming model, communication pattern, task dependencies and data placement. Second, the solution must be highly scalable as modern clusters contain tens of thousands of nodes and have hundreds of jobs with millions of tasks active at a time. Third, the scheduling system must be fault-tolerant and highly available as all the applications in the cluster depend on it.

This approach would be for Mesos to implement a centralized scheduler that takes as input framework requirements, resource availability and organizational policies and computes a global schedule for all tasks. While this approach can optimize scheduling across frameworks, it faces several challenges. The first is complexity. The scheduler would need to provide a sufficiently expressive API to capture all frameworks'

requirements and to solve an on-line optimization problem for millions of tasks. Even if such a scheduler were feasible, this complexity would have a negative impact on its scalability and resilience. Second as new frameworks and new scheduling policies for current frameworks are constantly being developed it is not clear whether we are even at the point to have a full specification of framework requirements. Thirdly, many existing frameworks implement their own sophisticated scheduling and moving this functionality to a global scheduler would require expensive refactoring.

Slagter *et al*. (2013) mapreduce is a programming model developed as a way for programs to cope with large amounts of data. It achieves this goal by distributing the workload among multiple computers and then working on the data in parallel. From the programmers perspective mapreduce is a relatively easy way to create distributed applications compared to traditional methods. It is for this reason map reduce has become popular and is now a key technology in cloud computing. Programs that execute on a map reduce framework need to divide the work into two phases known as map and reduce. Each phase has key-value pairs for both input and output. To implement these phases, a programmer needs to specify two functions: a map function called a mapper and its corresponding reduces function called a reducer. When a map reduce program is executed on Hadoop, it is expected to be run on multiple computers or nodes. Therefore, a master node is required to run all the required services needed to coordinate the communication between mappers and reducers. An input file (or files) is then split up into fixed sized pieces called input splits. These splits are then passed to the mappers who then work in parallel to process the data contained within each split. As the mappers process the data, they partition the output. Each reducer then gathers the data partition designated for them by each mapper, merges them, processes them and produces the output file. Task scheduling is an important process as described by Jawad (2006).

MATERIALS AND METHODS

Mapreduce computing framework: As a distributed computing framework on commercial computer, one of the map reduce most important benefits is that it offers an abstraction that hides many system level details from programmer. It processes data by separating the progress into two parts: map and reduce. Each map function obtains a split file as its input data which locates in the distributed file system and includes the key-value data. The split file can be co-location with the map function or not. If the split file and the map function don't in the same node, then the system will transmit the split file to the map function. This process will delay the execution of map task. Figure 2 show that the map function is concerned to each input key-value pair and produces an arbitrary number of intermediate key-value pairs. The process of map task involves:

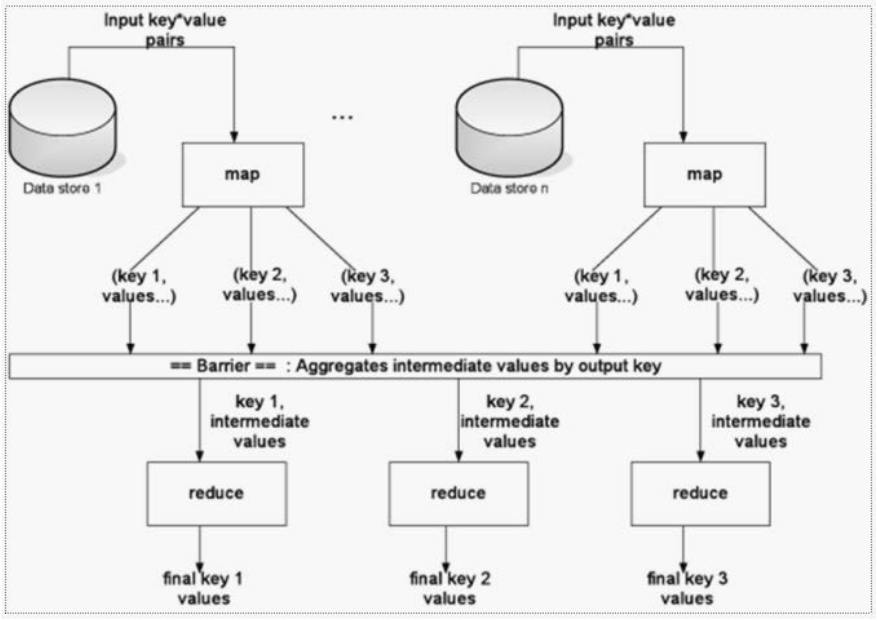

Fig. 2: A map reduce computation

- Read: Reading the input split and creating the key-value pairs
- Map: Executing the user-provided map function
- Merge and write: collecting the map output into a buffer and partitioning. Writing the buffer to disk as a spill file. Merging the file spills into a single map output file. Merging might be performed in multiple rounds

And the procedure of reduce task includes:

- Shuffle: Copying the map output from the mapper nodes to a reducer's node and decompressing if needed. Partial merging may also occur during this phase
- Merge: Merging the sorted fragments from the different mappers to form the input to the reduce function
- Reduce: Executing the user-provided reduce function
- Write: Writing the output to HDFS (Hadoop Distributed File System)

The reduce function is applied to all values that associated with the same intermediate key and generates output key-value pairs as the final result. In the map reduce framework, map or reduce codes can be moved among the cluster nodes and the data can be transferred from a node to another. If the code and data on the same node, we call this "data locality". The cost of migrating code is extremely lower than migrating data. So, the ideal situation is to move the code, not data.

Duplicate packet reduction based on map reduce framework: A packet is then split up into fixed sized pieces called input splits. These splits are then overtaken to the mappers who then work in parallel to process the data included within every split. As the mappers process the data, they partition the output. Each reducer then gathers the packet partition designated for them by each mapper, merges them, processes them and produces the output file.

It is the partitioning of the packet that determines the workload for each reducer. In the map reduce model, the workload should be balanced in order for resources to be utilized effectively. Similarly each partitioning packets are matched and redundant packet are removed in map reduce. In this study two algorithms is proposed. First algorithm is task partitioning; it is significant then that the partition role evenly shares task pairs between reducers for suitable workload distribution. Second algorithm is DTSM, partitioned task are scheduled based on the deadline constraints.

Task partitioning algorithm:

Input:
IT: Set of Input Tasks
i: index in the partition array
Partition size: Size of the partition array
Partition count: Total number of partitions
Key count: Number of prefixes in the partition array
k: Partition number
Output: PT: a set of partitioned tasks
Create IT by extracting n samples from source data.
For each Task T in IT
tc = Code (T)
If (partition[tc] == FALSE)
Key Count[tc] = 1
Else if (partition[tc] == TRUE)
Key Count[tc] = keyCount[tc]+1;
End if
End if
Partition[tc] = TRUE
End for
Fori = 0 to partitionSize
Totalkey = totalkey+key Count [i]
End for
Split Size = totalkey/partitionCount
Fori = 0 to partition Size
If (partition [i] ==TRUE)
Split = split + key Count[i]
If (split >split Size)
k = k +1
Split = key Count [i]
End if
Pt_k add (partition [i])
End if
End for

In this research, the scheduling algorithm sets dual deadlines: map and reduce-deadline. And reduce-deadline is presently the users' task deadline. In direct to get map-deadline; we require knowing the map task's time proportion on the task's execution time. In a cluster with range of resources, map slot and reduce slot number is determined. For an arbitrary given task with deadline constraints, the scheduler has to plan reasonable with the remaining resources in direct to promise that all tasks can be completed before the deadline constraints.

We build a model to compute the remaining time of all running map or reduce tasks. In order to obtain the slot constraint of running task, we must calculate the remaining task execution time.

T = (M, R, A, D) means a map reduce task. M, R, A and D denotes map task set, reduce task set, task arrived time and task deadline constraints, respectively. $N \leq n_1, n_2, n_3, ..., n_n>$ is the set of nodes.

CM (T, n) denotes the completed task T's map task set which run on the node and CR (T, n) means the

already completed reduce task set which ran on the node; TM_m denotes the task M_m 's (M_mCM (T, n)) completion time; Tr_r denotes the task R_r's (R_rCR (T, n)) completion time. The average completion time of task T's map tasks which run on the node should be calculated by all the maptasks that run on the node's completion time divided the completed task T's map task set which run on the node. So:

$$Mean_{TM}(T, n) = \left(\sum_{(M_m)\in CM(T,n)}\right)(TM_m)/\left(\left|CM(T, n\right|\right)$$

And the average completion time of task T's reduce tasks' which run on the node should be calculated by all the reduce tasks that run on the node's completion time divided the completed task T's reduce task set which run on the node. So:

$$Mean_{TR}(T,n) = \left(\sum_{(R_r)\in CR(T,n)}\right)(TR_r)/\left(\left|CR(T,n\right|\right)$$

We use UM(T) and UR(T) to denote waiting task set of the task T's map and reduce task type, respectively. Mm_m (T, n) denotes the completion time of task T's map task which runs on the node. So, the completion time of a running task equal to the running time that have spent and the remaining execution time. That is:

$$MM_m(T,n) = RTM_m + CTM_m$$

Where:

CTM_m = The m-th map task's running time that have spent

RTM_m = The m-th map task's remaining execution time

So, the m-th map task's remaining execution time of task T is:

$$RTM_m = MM_m(T,n) - CTM_m$$

The completion time of a certain tasks which run on the nodes belonging to the same capacity level, will tend to be the same. We know that Mm_m (T, n) is approximately, equal to Mean_TM (T, n):

$$RTM_m = Mean_{TM(T,n)} - CTM_m$$

In the same way, we can get the running reduce task's remainingexecution time:

$$RTR_r = Mean_{TR(T,n)} - CTR_r$$

where, CTR_r denotes the running time that have spent. Because the value of RTM_m or RTR_r which is based on the tasks' mean time, so we can not get the first map or reduce task's remaining time value. In the experiment we simply treat it as an infinity value in the beginning. Now, we can calculate the sum of the remaining map and reduce task's execution time of T which running on the nodes:

$$SM(T,n) = \sum_{1\le i\le m} RTM_m$$

$$SR(T,n) = \sum_{1\le i\le r} RTR_r$$

Uses empirical data to quantize the value of P_m and P_r. We use a data sample from user's input data and run the code on the data and then we get the map and reduce task's execution time T_m and T_r. So:

$$P_m = \frac{T_m}{T_m + T_r}$$

And:

$$P_r = \frac{T_r}{T_m + T_r}$$

Now, we can calculate the map-deadline by P_m:

$$D_m = A + (D - A) * P_m$$

where, A and D means the task's arrived time and deadline, respectively. According to map-deadline, we can acquire the current map task's slot number it needs and with reduce-deadline, we can get the current reduce task's slot number it needs. We estimate the time needed for the remaining task on the lowest level node. By this way, we can find the emergency degree of current task and the minimum map slot requirement of task T can be computed as follows:

$$\delta_T^m = \frac{\sum_{1\le n\le N} SM(T) + UM(T_n)\times MM(T,1)}{\left|D_m - \text{Current Time}\right|}$$

Similarly, the minimum reduce slot number requirement of task T is:

$$\delta_J^r = \frac{\sum_{1\le n\le N} SR(T,n) + UR(T)\times MR(J,1)}{\left|D - \text{Current Time}\right|}$$

The scheduling strategy of DTSM is based on δ_T^m and δ_T^r. The minimum map and reduce slot number required of task T can be denoted as δ_T^m and δ_T^r, respectively. The

symbol reflects that map tasks should be scheduled at present in order to meet task T's map-deadline as well as to meet the reduce-deadline (task deadline) δ_T^r reduce tasks should bescheduled. In the scheduling process, we take δ_T^m and δ_T^r as the basic criteria of priority allocation.

Algorithm 2 DTSM algorithm:

```
Collection assignTask(TaskTracker t )
M = t.freeMapSlots
R = t.freeReduceSlots
PTk=Ø
T =Ø
Compute the slot requirement for tasks in each partitioner PTk
Update tasks' priority in each partitioner PTk
Resort the tasks in each partitioner PTk
For each taskϵPTk do
Count = 0
If count <task.MapSlotRequirementCount
If job exists waiting map task and M≠Ø
Task t = task.obtainMapTask()
T = T∪{t}
Remove a map slot in M
Count++
Else break
Else break
For each taskϵPTk do
Count = 0
If count<task.ReduceSlotRequirementCount
If task exists waiting map task and R = Ø
Task t = task.obtainReduceTask()
T = T∪{t}
Remove a Reduce slot in R
Count++
Else break
Else break
Return T
```

But there are some special cases must be considered: at the beginning of the job be submitted, there is no data available, so the scheduler can't calculate approximately, the required slots or the execution time of tasks. In this case, the job's precedence is over than the others. In some situations, jobs may have previously missed their deadline. The approach we use is the same as the previous case: set such jobs' tasks with the highest priority. Algorithm 2 proposes the DTSM method. The input parameter t which including the free map slots and reduce slots, represents a request to enquire for waiting tasks. And the return value is a collection T which holds the tasks to be assigned to task tracker.

The mapreduce scheduling flows in cloud is as follows: the scheduling manages a each partitioner; the work node which is called task tracker asks the scheduler for tasks periodicity When a request arriving, the scheduler decides which task to be assigned to the task tracker according the scheduling algorithm. In algorithm 2, we firstly compute δ_T^m and δ_T^r for each task in partitioner and resort the task's order (see lines 6-8). Secondly, on lines 9-21 and lines 22-34, we schedule the waiting map and reduce tasks correspondingly. Finally, return the task collection T to the task trackter and the task tracker will run the tasks.

RESULTS AND DISCUSSION

In our experimental cloud setup simulation is performed by using cloudsim Ex. Using cloudsim EX we create the 700 host machines and 1000 virtual machines with different requirements. Each virtual machine runs the more number of tasks. We compare the proposed task partitioning based deadline aware scheduling strategy with the with existing technique as Multilevel Queue Scheduler (MQS) in terms of performance time, performance map, CPU utilization rate and the throughput. We have used CloudSim Ex to implement our proposed methodology which is based on mapreduce concepts.

Performance time: This metric defines the overall time taken to process the task partitioning and the task scheduling process with the concern of deadline. The time taken to process the entire workload by the proposed methodology and the existing methodology is shown in the following graph.

The corresponding values of total time for the number reduces are the plotted in Table 1. Figure 3 indicates the comparison graph of total time taken to schedule the tasks for different number of reducers. In the x axis, number of reducers is plotted whereas in the y axis total time taken to process the scheduling is plotted.

Performance map: The performance map defines the mapping performance. The mapping performance defines the percentage of mapping is performed by the number map task. The improved mapping of the proposed methodology than the existing methodology is shown in the following graphical representation (Fig. 4).

This graph indicated the performance improvement obtained for the corresponding mapping metric than the existing methodology. This Fig. 4 shows, in the x axis number of map tasks are plotted and in the y axis percentage of map tasks are plotted. The exact values of percentage of mapping are shown in Table 2.

Table 1: Total time values

	Total time	
Number of reducers	MQS	TPA-DTSM
1	550	400
2	1000	900
3	1600	1400
4	2000	1800
5	2800	2100
6	3300	2600
7	3900	3000
8	4400	3400
9	5000	3900

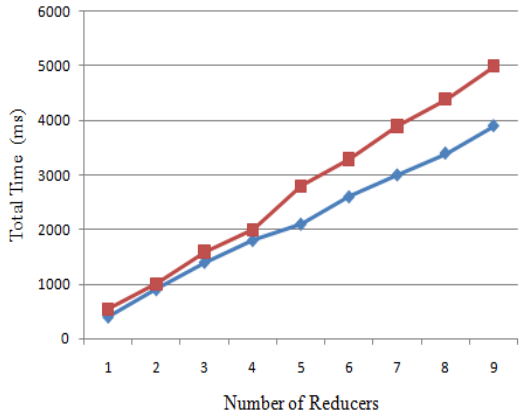

Fig. 3: Total time comparison

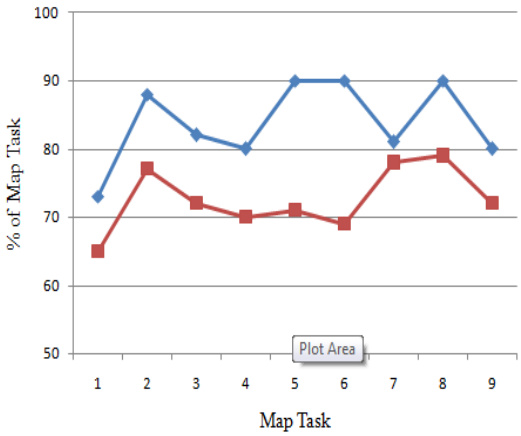

Fig. 4: Mapping performance comparison

Table 2: Mapping performance

	Map (%)	
Number of map task	MQS	TPA-DTSM
1	65	73
2	77	88
3	72	82
4	70	80
5	71	90
6	69	90
7	78	81
8	79	90
9	72	80

CPU utilization: CPU utilization refers to a computer's usage of processing resources or the amount of work handled by a CPU. Actual CPU utilization varies depending on the amount and type of managed computing tasks. The CPU utilization comparison of proposed methodology and the existing methodology is shown in Fig. 5.

This graph plots the CPU utilization range of processing the number of tasks submitted in the

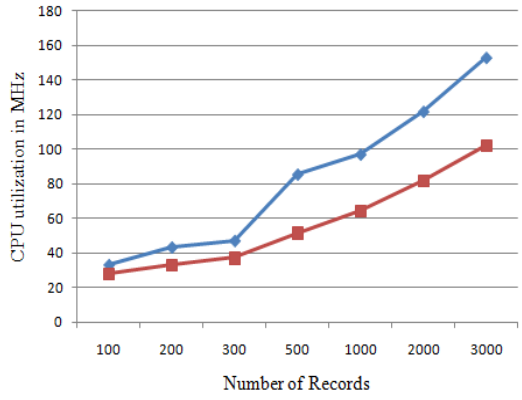

Fig. 5: CPU utilization

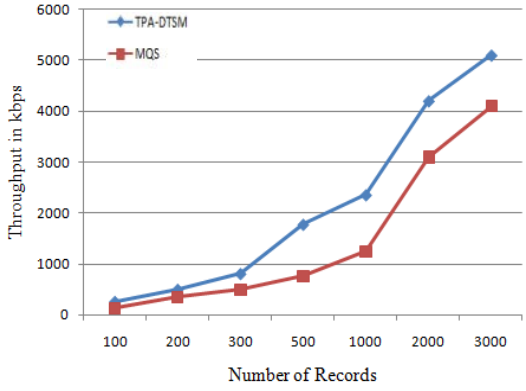

Fig. 6: Throughput

Table 3: CPU utilization

	CPU utilization	
Number of records	MQS	TPA-DTSM
100	28.0	33.0
200	33.0	43.2
300	37.1	47.0
500	51.6	85.4
1000	64.4	97.0
2000	82.0	121.8
3000	102.1	153.0

mapreduce component. The x axis plots the number of records executed and the y axis plots the CPU utilization consumed. The exact values of CPU utilization taken for plotting the graph is indicated in Table 3.

Throughput: Throughput defined as the number of packets can be transmitted in a particular period of time. The through put of the proposed methodology is increased than the existing work and the graphical illustration of the throughput is shown in Fig. 6.

Figure 6 shows the throughput comparison of existing work with the proposed methodology in terms of different number of records. In the x axis, the numbers of

Table 4: Throughput comparison

	Throughput	
Number of records	MQS	TPA-DTSM
100	140	180
200	350	380
300	500	520
500	770	850
1000	1250	1790
2000	3100	3700
3000	4100	5100

are plotted and in the y axis throughput value obtained is plotted in terms of kbps unit. The exact values obtained for the throughput of existing and proposed methodology is given in Table 4.

CONCLUSION

In this study present the packet reduction technique is used for find and removes the duplicate packets in network environment. In order to reduce the time it acquires to process the data and to have the storage space to store the data, we introduce a new approach called map reduce method. In this approach, it has to separate the workload among computers in a network. By improving network performance, mapreduce programs can become more efficient at handling tasks by reducing the overall computation time spent processing data on each node. The proposed partition algorithm used for partitioning the task that decides the workload for each reducer. In the mapreduce framework, the workloads have to be balanced in categorize for resources to be utilized efficiently. The DTSM proposed in this study focuses on user's deadline constraints problem. For a random submitted task with deadline constraints, the scheduler must schedule reasonable with the remaining resources in arrange to promise that all jobs can be completed before the deadline constraints. If the data are partitioned equivalent between the nodes then the execution time for the overall map reduce job is minimized.

RECOMMENDATIONS

In the future, we would like to employ the proposed task scheduler architecture and achieve additional researches to determine performance using straggling or heterogeneous nodes. We extend to examine other profits of micro-tasks, involving the use of micro-tasks as a substitute to preemption when scheduling combinations of batch and latency-sensitive jobs.

REFERENCES

Alexandraki, A. and M. Paterakis, 2005. Performance evaluation of the deadline credit scheduling algorithm for soft-real-time applications in distributed video-on-demand systems. Cluster Comput., 8: 61-75.

Candan, K.S., J. Kim, P. Nagarkar, M. Nagendra and R. Yu, 2011. RanKloud: Scalable multimedia data processing in server clusters. IEEE. MultiMedia, 18: 64-77.

Dean, J. and S. Ghemawat, 2008. mapreduce: Simplified data processing on large clusters. Commun. ACM, 51: 107-113.

Gonzalez, J.E., Y. Low, H. Gu, D. Bickson and C. Guestrin, 2012. Powergraph: Distributed graph-parallel computation on natural graphs. Proceedings of the Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation, October 8-10, 2012, ACM, New York, USA., pp: 17-30.

Hammoud, M. and M.F. Sakr, 2011. Locality-aware reduce task scheduling for mapreduce. Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science, November 29-December 1, 2011, IEEE, Athens, Greece, ISBN: 978-1-4673-0090-2, pp: 570-576.

Hammoud, M., M.S. Rehman and M.F. Sakr, 2012. Center-of-gravity reduce task scheduling to lower mapreduce network traffic. Proceedings of the IEEE 5th International Conference on Cloud Computing, June 24-29, 2012, IEEE, Honolulu, Hawaii, ISBN: 978-1-4673-2892-0, pp: 49-58.

Hindman, B., A. Konwinski, M. Zaharia, A. Ghodsi and A.D. Joseph *et al*., 2011. Mesos: A platform for fine-grained resource sharing in the data center. NSDI., 11: 22-22.

Jawad, S.K., 2006. Task scheduling in distributed systems using a mathematical approach. Asia J. Inform. Technol., 5: 69-74.

Polo, J., D. Carrera, Y. Becerra and J. Torres, 2010. Performance-driven task co-scheduling for mapreduce environments. Proceedngs of the Symposium on Network Operations and Management, April 19-23, 2010, Osaka, pp: 373-380.

Polo, J., D. Carrera, Y. Becerra, V. Beltran, J. Torres and E. Ayguade, 2010. Performance management of accelerated mapreduce workloads in heterogeneous clusters. Proceedings of the 39th International Conference on Parallel Processing, September 13-16, 2010, USA., pp: 653-662.

Slagter, K., C.H. Hsu, Y.C. Chung and D. Zhang, 2013. An improved partitioning mechanism for optimizing massive data analysis using mapreduce. J. Super Comput., 66: 539-555.

Zaharia, M., D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker and I. Stoica, 2010. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. Proceedings of the 5th European Conference on Computer Systems, April 13-16, 2010, Paris, France.