Towards a Resource Aware Scheduler in Hadoop

Shiwali Mohan, Nitin Garegrat, Mark Yong

Computer Science and Engineering, University of Michigan, Ann Arbor

December 21, 2009

Abstract

Hadoop-MapReduce is a popular distributed computing model that has been deployed on large clusters like those owned by Yahoo and Facebook and Amazon EC2. In a practical data center of that scale, it is a common scenario that I/O- bound jobs and CPU-bound jobs, that demand complementary resources, run simultaneously on the same cluster. Although improvement exist on the the default FIFO scheduler in Hadoop, much less work has been done on scheduling such that resource contention on machines is minimized. In this work, we study the behavior of the current scheduling schemes in Hadoop on a locally deployed cluster Simpool owned by ACAL, University of Michigan and propose new resource aware scheduling schemes.

1 Introduction

Hadoop MapReduce [1, 2] is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes. Under such models of distributed computing, many users can share the same cluster for different purpose. Situations like these can lead to scenarios where different kinds of workloads need to run on the same data center. For example, these clusters could be used for mining data from logs which mostly depends on CPU capability. At the same time, they also could be used for processing web text which mainly depends on I/O bandwidth.

The performance of a master-worker system like MapReduce system closely ties to its task scheduler on the master. Lot of work has been done in the scheduling problem. Current scheduler in Hadoop uses a single queue for scheduling jobs with a FIFO method. Yahoo's capacity scheduler [3] as well as Facebook's fair scheduler [4] uses multiple queues for allocating different resources in the cluster. Using these scheduler, people could assign jobs to queues which could manually guarantee their specific resource share.

Most of the current work has concentrated on looking at the scheduling problem from the master's perspective where the scheduler on the master node tries to assign equal work across all the worker nodes. In this work, we concentrate on improving resource utilization when different kinds of workloads run on the clusters. In practical scenarios, many kinds of jobs often simultaneously run in the data center. These jobs compete for different resources available on the machine, jobs that require computation compete for CPU time while jobs like feed processing compete for IO bandwidth. he Hadoop scheduler is not aware of the nature of workloads and prefers to simultaneously run map tasks from the job on the top of the job queue. This affects the throughput of the whole system which ,in turn, influences the productivity of the whole data center. I/O bound and CPU bound processing is actually complementary [5]. A task that is writing to the disk is blocked, and is prevented from utilizing the CPU until the I/O completes. This suggests that a CPU bound task can be scheduled on a machine on which tasks are blocked on the IO resources. When diverse workloads run on this environment, machines could contribute different part of resource for different kinds of work.

The rest of the paper is organized as follows. Section 2 describes the related work of this article. Section 3 provides a brief introduction to Hadoop and Map/Reduce framework. Section 4 presents our analysis of Hadoop-0.18.3 on the Simpool cluster at University of Michigan. In section

5, we propose new scheduling paradigms and finally we conclude in Section 6 with some discussion on future work.

2 Previous Work

The scheduling of a set of tasks in a parallel system has been investigated by many researchers. Many scheduling algorithms have been proposed [6, 7, 8], [9, 10] focus on scheduling tasks on heterogeneous hardware, and [6, 7, 8] focus on the system performance under diverse workload.

It is nontrivial to balance the use of the resources in applications that have different workloads such as large computation and I/O requirements [10]. [6] discuss the problem of how I/O-bound jobs affect system performance, and [5] shown a gang schedule algorithm which parallelize the CPU-bound jobs and IO-bound jobs to increase the utilization of hardware.

The scheduling problem in MapReduce has also attracted considerable attention. [11, 12] addressed the problem of how to robustly perform speculative execution mechanism under heterogeneous hardware environment. [13] derive a new family of scheduling policies specially targeted to sharable workloads.

3 Hadoop Framework

Hadoop is a open source software framework that dramatically simplifies writing distributed data intensive applications. It provides a distributed file system, which is modeled after the Google File System[14], and a map/reduce[1] implementation that manages distributed computation.

3.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project.

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per machine in the cluster, which manage storage attached to the machine that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

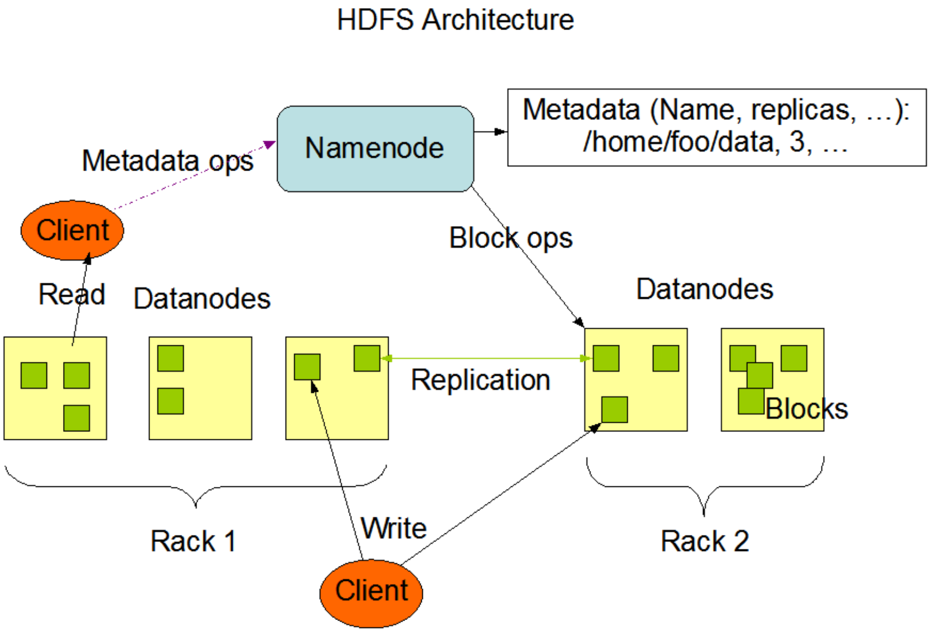

Figure 1: Hadoop Distributed File System architecture

3.2 Job Tracker and Task Tracker

MapReduce engine is designed to work on the HDFS. It consists of one Job Tracker, to which client applications submit MapReduce jobs. The Job Tracker pushes work out to available Task Tracker nodes in the cluster, striving to keep the work as close to the data as possible. With a rack-aware file-system, the Job Tracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network. If a Task Tracker fails or times out, that part of the job is rescheduled. If the Job Tracker fails, all ongoing work is lost.

This approach has some known limitations:

- The allocation of work to task trackers is very simple. Every task tracker has a number of available slots (such as "4 slots"). Every active map or reduce task takes up one slot. The Job Tracker allocates work to the task tracker nearest to the data with an available slot. There is no consideration of the current active load of the allocated machine, and hence its actual availability.
- If one task tracker is very slow, it can delay the entire MapReduce operation - especially towards the end of a job, where everything can end up waiting for a single slow task. With speculative-execution enabled, however, a single task can be executed on multiple slave nodes.

3.3 The Map-Reduce Programming Paradigm

Map/Reduce is a programming paradigm that expresses a large distributed computation as a sequence of distributed operations on data sets of key/value pairs. The Hadoop Map/Reduce framework harnesses a cluster of machines and executes user defined Map/Reduce jobs across the nodes in the cluster. A Map/Reduce computation has two phases, a map phase and a reduce phase. The input to the computation is a data set of key/value pairs.

In the map phase, the framework splits the input data set into a large number of fragments and assigns each fragment to a map task. The framework also distributes the map tasks across the cluster of nodes on which it operates. Each map task consumes key/value pairs from its assigned fragment and produces a set of intermediate key/value pairs. For each input key/value pair (K, V), the map task invokes a user defined map function that converts the input into a different key/value pair (K', V').

Following the map phase the framework sorts the intermediate data set by key and produces a set of $(K', V'*)$ tuples so that all the values associated with a particular key appear together. It also partitions the set of tuples into a number of fragments equal to the number of reduce tasks.

In the reduce phase, each reduce task consumes the fragment of $(K', V'*)$ tuples assigned to it. For each such tuple it invokes a user-defined reduce function that transmutes the tuple into an output key/value pair (K, V). Once again, the framework distributes the many reduce tasks across the cluster of nodes and deals with shipping the appropriate fragment of intermediate data to each reduce task.

4 Scheduling in Hadoop

4.1 Task Scheduling in Hadoop

As of v0.20.1, the default scheduling algorithm in Hadoop operates off a first-in, first-out (FIFO) basis. Beginning in v0.19.1, the community began to turn its attention to improving Hadoop's scheduling algorithm, leading to the implementation of a plug-in scheduler framework to facilitate the development of more effective and possibly environment-specific schedulers.

Since then, two of the major production Hadoop clusters – Facebook and Yahoo – developed schedulers targeted at addressing their specific cluster needs, which were subsequently released to the Hadoop community.

4.1.1 Default FIFO Scheduler

The default Hadoop scheduler operates using a FIFO queue. After a job is partitioned into individual tasks, they are loaded into the queue and assigned to free slots as they become available on TaskTracker nodes. Although there is support for assignment of priorities to jobs, this is not turned on by default.

4.1.2 Fair Scheduler

The Fair Scheduler [4] was developed at Facebook to manage access to their Hadoop cluster, which runs several large jobs computing user metrics, etc. on several TBs of data daily. Users may assign jobs to pools, with each pool allocated a guaranteed minimum number of Map and Reduce slots. Free slots in idle pools may be allocated to other pools, while excess capacity within a pool is shared among jobs. In addition, administrators may enforce priority settings on certain pools. Tasks are therefore scheduled in an interleaved manner, based on their priority within their pool, and the cluster capacity and usage of their pool.

As jobs have their tasks allocated to TaskTracker slots for computation, the scheduler tracks the deficit between the amount of time actually used and the ideal fair allocation for that job. As slots become available for scheduling, the next task from the job with the highest time deficit is assigned to the next free slot.

Over time, this has the effect of ensuring that jobs receive roughly equal amounts of resources. Shorter jobs are allocated sufficient resources to finish quickly. At the same time, longer jobs are guaranteed to not be starved of resources.

4.1.3 Capacity Scheduler

Yahoo's Capacity Scheduler [3] addresses a usage scenario where the number of users is large, and there is a need to ensure a fair allocation of computation resources amongst users. The Capacity Scheduler allocates jobs based on the submitting user to queues with configurable numbers of Map and Reduce slots.

Queues that contain jobs are given their configured capacity, while free capacity in a queue is shared among other queues. Within a queue, scheduling operates on a modified priority queue basis with specific user limits, with priorities adjusted based on the time a job was submitted, and the priority setting allocated to that user and class of job.

When a TaskTracker slot becomes free, the queue with the lowest load is chosen, from which the oldest remaining job is chosen. A task is then scheduled from that job. Overall, this has the effect of enforcing cluster capacity sharing among users, rather than among jobs, as was the case in the Fair Scheduler.

4.1.4 Scheduler Optimizations

Although the default scheduler is extremely simple, it does implement some measure of fault-tolerance through the use of speculative execution.

Speculative Execution

It is not uncommon for a particular task to continue to make progress, but to do so extremely slowly. This may occur for a variety of reasons – high CPU load on the node, temporary slowdown due to background processes, etc. In order to prevent a final straggler task from holding up completion of the entire job, the scheduler monitors tasks that are progressing slowly when the job is mostly complete, and executes a speculative copy on another node. If the copy completes faster, the overall job performance is improved; this improvement can be significant in real-world usage.

LATE Speculative Execution

The default implementation of speculative execution relies implicitly on certain assumptions, the two most important of which are listed below:

1. Tasks progress in a uniform manner on nodes
2. Nodes compute in a uniform manner.

In the heterogeneous clusters that are found in real-world production scenarios, these assumptions break down very easily. Zaharia et al [11] propose a modified version of speculative execution that uses a different metric to schedule tasks for speculative execution. Instead of considering the progress made by a task so far, they compute the estimated time remaining, which provides a far more intuitive assessment of a straggling tasks' impact on the overall job response time. They demonstrate significant improvements by LATE over the default speculative execution implementation.

4.2 Fine-Grained Resource Aware Scheduling

While the two improved schedulers described above attempt to allocate capacity fairly among users and jobs, they make no attempt to consider resource availability on a more fine-grained basis. Given the pace at which CPU and disk channel capacity has been increasing in recent years, a Hadoop cluster with heterogeneous nodes could exhibit significant diversity in processing power and disk access speed among nodes. Performance could be affected if multiple processor-intensive or data-intensive tasks are allocated onto nodes with slow processors or disk channels respectively. This possibility arises as the JobTracker simply treats each TaskTracker node as having a number of available task "slots". Even the improved LATE speculative execution could end up increasing the degree of congestion within a busy cluster, if speculative copies are simply assigned to machines that are already close to maximum resource utilization.

HADOOP-3759 [14] and HADOOP-657 [15] address this partially: 3759 allows users to specify an estimate for the maximum memory their task requires, and these tasks are scheduled only on nodes where the memory per node limit exceeds this estimate; 657 does the same for the available disk space resource. However, disk channel bandwidth is not considered, and JobTracker makes no attempt to load balance CPU utilization on nodes.

4.2.1 TaskTracker Resource Tracking

All scheduling methods implemented for Hadoop to date share a common weakness:

1. failure to monitor the capacity and load levels of individual resources on TaskTracker nodes, and
2. failure to fully use those resource metrics as a guide to making intelligent task scheduling decisions.

Instead, each TaskTracker node is currently configured with a maximum number of available computation slots. Although this can be configured on a per-node basis to reflect the actual processing power and disk channel speed, etc available on cluster machines, there is no online modification of this slot capacity available. That is, there is no way to reduce congestion on a machine by advertising a reduced capacity.

5 Hadoop Network Profiling on Simpool

To evaluate the performance and identify the limitations of the FIFO scheduler, we ran different benchmarks on the Simpool cluster. CPU utilization was used as the performance metric. The machine specifications of m35 class are given in Table 1.

Machine class	m35
Processor	2.4GHz P4 32-bit
RAM	1GB
OS	Ubuntu 9.04

Table 1: Machine Specifications of m35 cluster on Simpool at University of Michigan Ann Arbor

Hadoop-0.18.3 was set up on 6 machines of the m35 machine class of the cluster. The configuration of the set up is described in Table 2.

Master (Namenode)	m35-001
Slaves (Datanode)	m35-002, m35-003, m35-004, m35-005, m35-006

Table 2: Master and Slave set up on Cluster

We used the standard Sorting benchmark used widely for performance analysis in general. The data for the Sorting benchmark was generated using RandomWriter (Table 3), present in the examples directory available with the Hadoop package.

Number of hosts	6
Number of maps/hosts	11
Data in bytes/map	50MB
Running Time	126 secs

Table 3: RandomWriter runtime

Once the data was generated, Sorting job was submitted by m35-001 to sort this data. Although the simpool environment did not have much variability, but to address some random unexpected variability, the same Sort job was submitted 3 to 4 times.

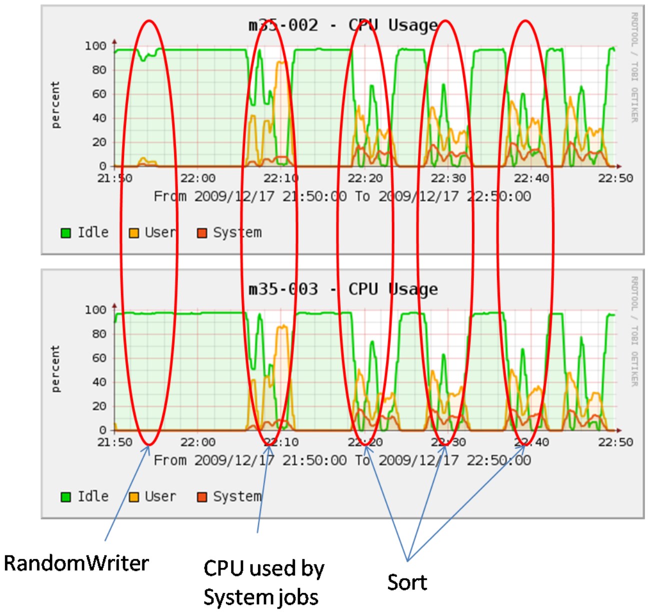

Figure 2: Trace obtained by running Random Writer and Sort benchmark

	Iteration 1	Iteration 2	Iteration 3
number of maps	11	11	11
number of hosts	6	6	6
Data in bytes/map	50 MB	50 MB	50 MB
Total maps	60	60	60
Total reduces	30	30	30
Running time	305 secs	294 secs	270 secs

Table 4: RandomWriter and Sort runtime

In the trace above, the peaks of CPU utilization (approximately 60%) when Sort is being executed correspond to Map jobs. The troughs (approximately 30%) on the other hand show the CPU utilized by Reduce jobs. There is not much variance between the different Sort jobs since there was no contention for CPU by other jobs and the environment was fairly stable.

After our preliminary analysis, 3 machines on the cluster submitted 3 Sorting jobs one after another (starting times within 10 sec between them).

The logs revealed that the default scheduler fails to be fair to all the 3 Sort jobs submitted. The current scheduler starts by submitting a bunch of Map jobs for Sort on m35-001 and then only after some time starts scheduling the Map jobs for the other Sort jobs.

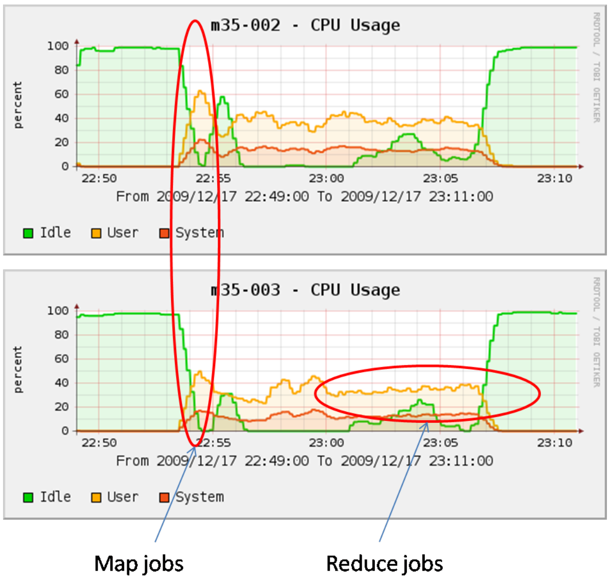

Figure 3: Trace obtained for Sort jobs submitted by m35-001, m35-002 and m35-003 (Sort (x3))

Iteration	1	2	3	4	5	6
# maps	11	11	11	11	11	11
# hosts	6	6	6	6	6	6
Data in bytes/maps	50MB	50MB	50MB	50MB	50MB	50MB
Total maps	60	60	60	10	10	3
Total reduces	30	30	30	10	10	3
Running time	452 sec	675 sec	825 sec	1039 sec	879 sec	1220 sec

Table 5: Sort (x3) runtime

To analyze how resource aware the current scheduler is, we artificially created CPU contention by running an CPU script that used 40% of the CPU on m35-003. Sort job was submitted by m35-001. CPU traces on m35-003 and m35-002 are shown in Figure 4.

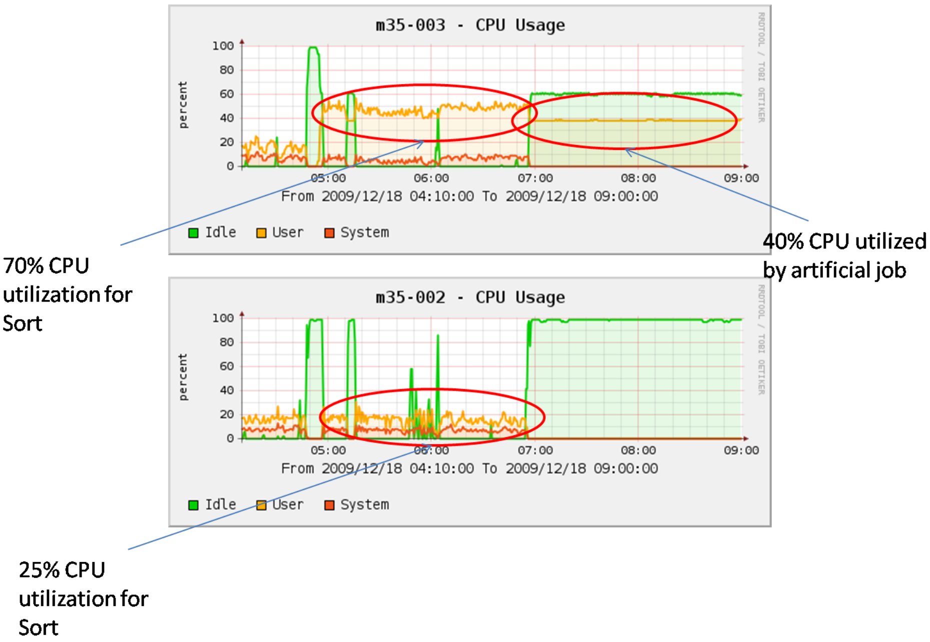

Figure 4: Trace obtained for Sort running with artificially created CPU contention

	Sort w/o cpu hit	Sort with cpu hit
# maps	11	11
#hosts	6	6
Data in bytes per map	50mb	50mb
Total Maps	60	60
Total Reduces	60	60
Running time	6010 sec	6268 sec

Table 6: Sort with CPU contention runtime

From the traces above, the scheduler completely ignores the fact that m35-003 is already loaded and m35-002 is not being utilized by any other job. Ideally, a scheduler should submit comparatively less number of jobs to m35-003 and more jobs to other machines.

6 Proposed Scheduler

We propose an improved scheduler that monitors per-node resource load levels at fairly high fidelity, then makes use of those metrics to compute actual forward capacity at each node.

6.1 TaskTracker Resource Monitoring

In our framework, each TaskTracker node monitors resources such as CPU utilization, disk channel IO in bytes/s, and the number of page faults per unit time for the memory subsystem. Although we anticipate that other metrics will prove useful, we propose these as the basic three resources that must be tracked at all times to improve the load balancing on cluster machines. In particular, disk channel loading can significantly impact the data loading and writing portion of Map and Reduce tasks, more so than the amount of free space available. Likewise, the inherent opacity of a machine's virtual memory management state means that monitoring page faults and virtual memory-induced disk thrashing is a more useful indicator of machine load than simply tracking free memory.

6.2 JobTracker Scheduling

We next propose two resource-aware JobTracker scheduling mechanisms that makes use of the resource metrics computed in the previous section.

6.2.1 Dynamic Free Slot Advertisement

Instead of having a fixed number of available computation slots configured on each TaskTracker node, we compute this number dynamically using the resource metrics obtained from each node. In one possible heuristic, we set the overall resource availability of a machine to be the minimum availability across all resource metrics. In a cluster that is not running at maximum utilization at all times, we expect this to improve job response times significantly as no machine is running tasks in a manner that runs into a resource bottleneck.

6.2.2 Free Slot Priorities/Filtering

In this mechanism, we retain the fixed maximum number of compute slots per node, viewing it as a resource allocation decision made by the cluster administrators at configuration time. Instead, we decide the order in which free TaskTracker slots are advertised according to their resource availability. As TaskTracker slots become free, they are buffered for some small time period (say, 2s) and advertised in a block. TaskTracker slots with higher resource availability are presented first for scheduling tasks on. In an environment where even short jobs take a relatively long time to complete, this will present significant performance gains. Instead of scheduling a task onto the next available free slot (which happens to be a relatively resource-deficient machine at this point), job response time would improved by scheduling it onto a resource-rich machine, even if such a node takes a longer time to become available. Buffering the advertisement of free slots allows for this scheduling allocation.

6.3 Energy efficient scheduling

On observing Figure 4 and Table 6 we observe that by making the scheduler resource aware, m35-002 can be put to sleep and assign all jobs if possible to m35-003. The total energy consumed by all the machines using this optimization will be less than the total energy consumed without using the proposed idea. It is important to observe that there will be total power reduction by using the proposed idea. But for a cluster we should consider energy instead of power, because power is not dependent on the runtime of the jobs on the machine. There is a possibility that the jobs submitted on a cluster can take a long time to complete, this will increase the energy but not necessarily the power. Therefore, coming up with an intelligent metric and finding the optimum threshold is the most challenging task for gaining energy benefits.

7 Conclusion and Future Work

This work analyzes the default FIFO scheduler of Hadoop on a local cluster Simpool. We propose two more efficient, resource aware scheduling schemes that minimize contention for CPU and IO resources on worker machines and can give better perform ace on the cluster.
Better scheduler schemes could be designed if resource utilization can be predicted by looking at the current resource usage of the task or by user supplied characteristics. Such a scheduler would be a learning scheduler that can classify tasks in CPU bound and IO bound categories and assign jobs as appropriate.

8 References

[1] Hadoop http://hadoop.apache.org/
[2] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Communications of the ACM, Volume 51, Issue 1, pp. 107-113, 2008.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] Hadoop's Capacity Scheduler http://hadoop.apache.org/core/docs/current/capacity_scheduler.html.
[4] Matei Zaharia, "The Hadoop Fair Scheduler" http://developer.yahoo.net/blogs/hadoop/FairSharePres.ppt
[5] Yair Wiseman and Dror G. Feitelson, "Paired Gang Scheduling," IEEE Transactions on Parallel and Distributed System, vol. 14, no. 6, June 2003
[6] M.J. Atallah, C.L. Black, D.C. Marinescu, H.J. Siegel and T.L. Casavant, "Models and algorithms for co-scheduling compute-intensive asks on a network of workstations," Journal of Parallel and Distributed Computing 16, 1992, pp.319–327
[7] D.G. Feitelson and L. Rudolph, "Gang scheduling performance benefitsfor fine-grained synchronization," Journal of Parallel and Distributed Computing 16(4),December 1992, pp.306–318
[8] J.K. Ousterhout, "Scheduling techniques for concurrent systems," in Proc. of 3rd Int. Conf. on Distributed Computing Systems, May 1982, pp.22–30.
[9]H. Lee, D. Lee and R.S. Ramakrishna, "An Enhanced Grid Scheduling with Job Priority and Equitable Interval Job Distribution," The first International Conference on Grid and Pervasive Computing, Lecture Notes in Computer Science, vol. 3947, May 2006, pp. 53-62
[10]A.J. Page and T.J. Naughton, "Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing," in 19th IEEE International Parallel and Distributed Processing Symposium, 2005.
[11] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, and Randy Katz Ion Stoica, "Improving MapReduce Performance in Heterogeneous Environments," Proceedings of the 8th conference on Symposium on Opearting Systems Design & Implementation
[12] E. Rosti, G. Serazzi, E. Smirni, and M.S. Squillante, "Models of Parallel Applications with Large Computation and I/O Requirements," IEEE Trans. Software Eng., vol. 28, no. 3, Mar.2002, pp. 286-307
[13]Parag Agrawal, Daniel Kifer, and Christopher Olston, "Scheduling Shared Scans of Large Data Files," PVLDB '08, August 2008, pp.23-28
[14] https://issues.apache.org/jira/browse/HADOOP-3759
[15] https://issues.apache.org/jira/browse/HADOOP-657