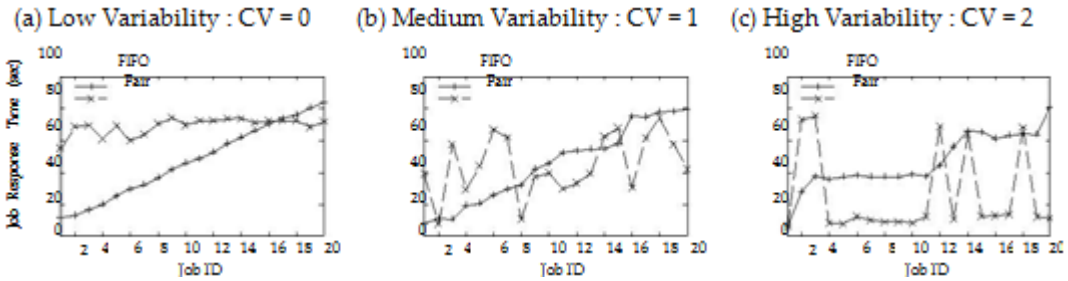

Figure 2: Response times of each WordCount job under FIFO and Fair when the input file sizes have different CV

3.1 Information Collector

To Design an effective scheduling algorithm the job size and patterns of it must considered. So, a light weight information collector is used to get the information of jobs and user. This information is updated when each job is bifurcated as map and reduce tasks.

In Effective scheduler, the important is workload information that needs to be collected for each user u_i includes its average task execution time t_i^m (and t_i^r), average size.

Algorithm 1 Effective scheduler

1. When user i submit a new job
 a. Job size S_i is estimated for user i.
 b. Slot share is modified among users, alg.2
 c. Job Scheduling is tuned for user i, alg. 3
2. When job j is of task is finished Execution time $t_{i,j}$ is updated.
3. When j^{th} job of user i finished average map reduce time is measured $t_{i,j}^m$, $t_{i,j}^r$.
4. After Tuning the available free slots are allotted based of the tuned scheduling order.

Our Statistics are based on below formulas,

$$s_{i,j} = t_{i,j}^m + t_{i,j}^r \quad (1)$$

$$S_i = S_i + (s_{i,j} - S_i)/j \quad (2)$$

$$v_i = v_i(s_{i,j} - S_i)^2.(j-1)/j \quad (3)$$

$$CV_i = \sqrt{v_i/j}/S_i \quad (4)$$

$$SU_i = SU_i^*.(\propto . U.\frac{s^*}{su_{-1}} + 1 - \alpha) \quad (5)$$

Where $s_{i,j}$ denotes size of j^{th} job completed of user u_i, $t_{i,j}^m$ (respected $t_{i,j}^r$) represents the measured average map (respected reduce) task execution time of $j_{i,j}$ means the measured map (resp. reduce) task number of the $j_{i,j}$. A job's size $s_{i,j}$ is defined as the summation of the execution times of all tasks of the job, which is independent on the level of task concurrency during the execution. The estimation of a job's size uses the previous tasks execution times of the job based on a well accepted assumption that the same type of tasks (either map or reduce tasks) of the same job have similar execution times. Additionally, $v_i = j$ denotes the variance of job sizes and are both initialized as 0 and updated each time when a new job is finished and its information is collected.

The data structure used to collect user's information that includes user ID, number of submitted by the user, map/reduce task execution times, average and variance of job sizes, and the last update time for detecting inactive users. The memory space used for each user is 26 bytes and our proposed system requires a total memory space of 130 bytes when 5 users are taken in our experiments. This is a overhead for regular MapReduce clusters. To further reduce the space overhead, proposed system timely checks the inactive user records if a user has not submitted any jobs in 5 minutes. The average task execution time of each active jobis recorded in another data structure, e.g., JobInfo used by Fair. JobInfo is used to store information such as number of running tasks for each active job, which is created when a job is submitted and after execution job is deleted.

3.2 Tier 1

This section tells about algorithm used for scheduling between number of users. The main goal is decide the number of slots allocate the slots to active user. MapReduce consists of two kinds of slots map and reduce slots. We have designed two algorithm, one for allocating map slots and another one to allocate reduce slots.

Assigning slots equally cannot give better result. So, we proposed a new scheduler which adaptively allocates slots shares among all users. Consider an example where two users, if their job ratio is equal to 1:2, then number of allotted to user1 will be twice that of user2. Consequently our scheduler give higher priority to smaller jobs, results in shorter response time.

One serious problem that has to be addressed is how to exactly measure the execution time of map or reduce task that are waiting or running currently. In hadoop it is not possible to get the exact processing time of job before the job is completely executed. We can predict the execution time of job in hadoop as discussed in earlier section. The Resulting share slot need not to be equal to the actual share slot assignment between users. After redistribution which user can get the available slots as shown step 4 in the below algorithm.

Algorithm 2 Tier 1: Slot Share Allocation to users

for each user u_i **do**
Update the slot share details of users SU_i using Eq.6;
for user i's j^{th} job **do**
 if $j_{i,j}$ is submitted first **then**
 $SJ_{i,j}$=SU_i;
 else
 $SJ_{i,j}$=0;

As shown in above algorithm in first step after arriving a new job, Effective scheduler updates the job size of that user

and adaptively adjusts slot share(SU_i) among all users using Eq.5 where SU_i represents the estimated slot share that will be assigned to the users.

The Effective scheduler sorts the user in descending order with redistributed slots. The scheduler will dispatch the job towards the slots after assigning slots. When the free slots are available with new order then that free slots can be allotted to users waiting.

3.3 Tier 2

The Design principal in our proposed system after adjusting share slots among multiple users dynamically tunes among each individual user jobs by observing at each individual user job details. This tier look into the available resources and equally distributes it among shared resources and it avoids small jobs waiting behind large ones i.e., it gives priority to shortest job first and then looks for the longest job.

Algorithm 3 Tier 2: Dynamically tuning the scheduling for each user

for each user u_i **do**
if user u_i is active **then**
find CV_i^* current jobs
if CV_i^*<1 and CV_i<1 **then**
Scheduling is based on job submission;
if CV_i^*>1 and CV_i>1 **then**
slots are equally allotted among jobs;
clear the previous information and start execution from beginning.

The above considers the CV of job sizes, i.e., map plus reduce size, of each user to find out which scheme should be used to allocate the free slots to jobs from that user. To improve the accuracy, we combine the job size information and the estimated size of running and waiting jobs in system. CV_i of currently finished jobs sizes of user i is provided directly by the history information collector, and CV_i of waiting and running jobs' sizes is calculated based on the estimated job sizes. When the two values of a user are both smaller than 1, the proposed scheme schedules the current jobs in that user's in the order of their submission times, otherwise the user level scheduler will equally assign slots among jobs. The previous information will be cleared and a new collection window will start at this time.

4. Model Description

In this section, we introduce a queuing model Which is designed to embed the Hadoop system. The main purpose of this model is to compare with numerous Hadoop scheduling schemes, and give best proof result to our new approach. This model does not include all the details of the complex Hadoop system, but provide a generalized guideline to users .

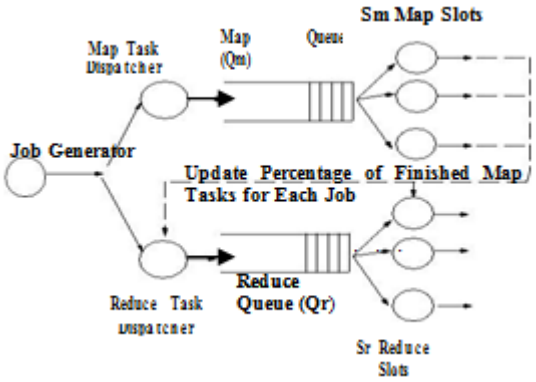

Figure 3: Design of the Hadoop MapReduce cluster simulator.

The model shown in Fig. 3 consists of two queues for map tasks (Q_m) and reduce tasks (Q_r), respectively. Once a job is submitted, tasks will be inserted into Q_m through the map task dispatcher and then it is reduced and inserted to reduce queue(Q_r) through reduce dispatcher.

An very important feature of MapReduce is jobs need to be considered in the model is the dependency between map and reduce tasks. In a Hadoop cluster, there is a Key which decides when a job could start its reduce tasks. By default, this parameter is set as 5%, which shows that the first reduce task can be started when 5% of the map tasks are committed. However, Studies [14] found that this setting would lead to poor performance under the Fair scheduling scheme and proposed to launch reduce tasks gradually according to the progress of map phase. We further found that delaying the launch time of reduce tasks, can improve the performance of the Fair and the other slots sharing based schedulers. However, this is not a necessary assume in the model.

When compared to the complex design of a MapReduce system, the simulator built based on this model is a much-simplified tool. Our aim is design a queue that can quikly adopt any scheduling. Point of the simulation model is to record the reactions of different scheduling policies on the job response time under different conditions. Therefore, the model we designed mainly simulates this key feature, i.e., how to share slots, without capturing the low-level details of a MapReduce system, such as communication costs, locality of data, and fault-tolerant mechanism.

5. Evaluation

Here we present the performance evolution of our proposed scheduler which mainly targets on how to improve efficiency of Hadoop system under different workloads.

5.1 Simulation Results

Initially we test Effective scheduler with the simulation model shown in previous section which is emulated with existing Hadoop system. We use a trace driven simulation model to evaluate the performance of proposed scheduler to improve it in terms of average response time, we use this on the top of the model. Later the performance of the scheduler can be improved by implementing it on a CloudEra Hadoop cluster.

We have configured number of map reduce slots in the clusters in the simulation model. We have *U* users i.e.{ u_1, u_2,.. u_i, ..., u_U } this users get the slots after submitting their jobs to the system. Each user specification like jobId, inter arrival time, job size etc are recorded. Each hadoop job scheduled is determined with number of map tasks and respective reduce tasks during the execution.

We consider different workloads to calculate average job response time and execution time. Here we consider busy workload, intermediate workload and normal workload. We consider three simple cases where the available cluster is shared with two users and with multiple clusters. We consider different job size in first case, different job arrival pattern in case two.

5.1.1 Case 1:Different Job Size with Two Users

Here we consider two users *u1* and *u2* simultaneously submit their Hadoop jobs to the system. We evaluate this Hadoop jobs with existing FIFO and Fair scheduler with *job size patterns* we consider different job size patterns with user *u1* and *u2*. We consider scheduler performance with different jobs and same job size with two users. number of scheduling policies. Job response is measured from when job a particular job is submitted and arrived to the job tracker till it is assigned to map reduce. We noted Figure 4 shows the job response time of both users under that high variance in job size decreases the performance under FIFO because a huge number of small jobs will stuck behind the large ones, it is plotted in Figure 4(a) in contrast with fair and proposed scheduler where performance is improved in other two schemes by scheduling the available slots between users. Our proposed scheduler further improves by scheduling FIFO with user 1 and other 2 is scheduled with FIFO and user1 with other policies as shown in Figure 4(c). Same experiment is done with fair that shows the performance improvement with two users with our proposed scheduler.

5.1.2 Case 2: Different Job Size Arrival Pattern with Two Users

In this Section we consider changes in job arrival pattern. We conduct some experiments with two users with their differing arrival pattern the job arrival ratio between user is considered here so we test with a job arrival ratio say 1:5 with respect user 1 and user 2, we consider the job arrival with different workload patterns, the average response time of two users shown in Figure 5.

Our proposed scheduler performs well in terms job response times as shown in Figure 5(a) we noted that the outcome benefit come with improvement in response time of user, our scheduler assigns more slots to user 1 with FIFO scheduling to smaller jobs based on FIFO because FIFO has low variability job sizes as shown in Figure 5(c).

5.1.3 Case 3: Different job size/arrival pattern with multiple users

To further verify our scheduler with multiple

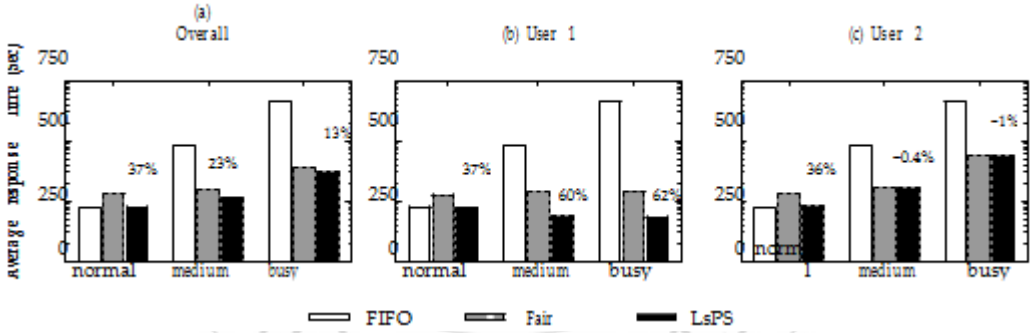

Figure 4: Average job response times of (a) two users, (b) user 1, and (c) user 2 under three different scheduling policies and different job size distribution settings. The relative improvement with respect to Fair is also plotted on each bar of LsPS.

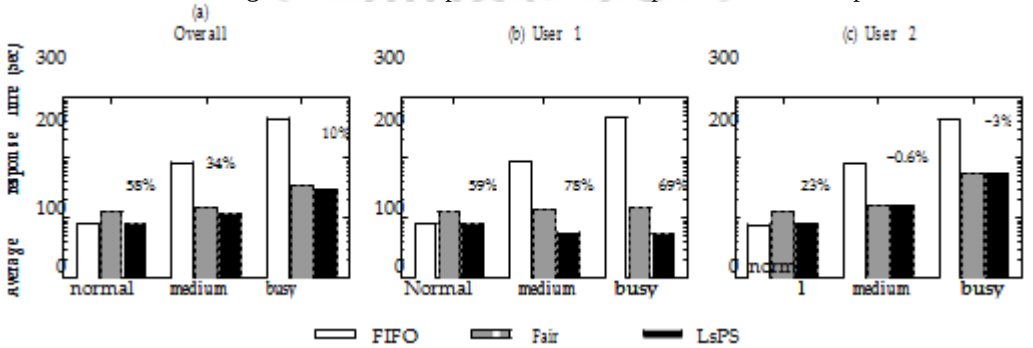

Figure 5: Average job response times of (a) two users, (b) user 1, and (c) user 2 under different scheduling policies and different job inter arrival time distributions. The relative job size ratio of two users is 1:5.

number of users we conduct experiments with complex case of 4 users which have mixed workload of changing job size arrival and job size patterns. Here user with larger job id will have larger job size in average. We also adjusted the average arrival rate such that all the users submit the same load to the system. Below table 2 shows the average job response times of jobs of users under different scheduling schemes. We compare our scheduler with other available schedulers.

Table 2: Average response times (in seconds) of all users and each user under different scheduling policies.

User	FIFO	Fair	Proposed 0:2	Proposed 0:4	Proposed 1:0
1	6398.10	171.83	63.00	50.05	42.41
2	8729.09	233.26	134.64	142.00	142.10
3	8345.62	235.09	176.12	188.88	152.44
4	8509.23	672.11	442.42	534.20	457.81
All	6834.12	745.65	365.23	431.27	342.52

Table 2 shows the average job size in terms time for each individual users and average of all six users as well. The above table also gives the simulation results of effective scheduler with ratio of job size with number of users equal to 0:2,0:4,1:0.

Table 3: Notations used in the algorithm

U/u_i	number of users/ *i*–th user
J_i	set of all user *i* jobs
$t_{i,i}^m / t_{i,i}^r$	average map/reduce task execution time of job
$n_{i,i}^m / n_{i,i}^r$	number of map/reduce task in job
$S_{i,i}$	size of job
S_i / S_i^*	Average size of completed/current jobs from user u_i
CV_i^* / CV_i	CV of completed/current jobs from user u_i
$SU_i / SJ_{i,i}$	Slot share of u_i /slot share of $job_{i,i}$

6. Conclusion

The use of FIFO and Fair scheduler will seriously degrade the performance of the overall Hadoop system. So, the proposed Effective scheduler is an adaptive scheduling technique which can improve the performance of the Hadoop system that process large number of MapReduce jobs. In enterprise the workload will drastically increase with different workload patterns this may happen from seconds to hours that will put workload on MapReduce cluster as well. Adopting our policy can record job size patterns based on the job size pattern knowledge can schedule among all users and further it dynamically tunes the scheduling among individual user jobs and assigns the available slots efficiently. Exeriments done in CloudEra had shown that our Effective scheduler will dramatically improves the performance in terms of job response time under varying workloads.

References

[1] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, 2008, 51(1):1-13.
[2] J. Dean, S. Ghemawat, and G. Inc, "Mapreduce: simplified data processing on large clusters," in OSDI'04, 2004.
[3] Apache Hadoop Users. [Online]. Available: http://wiki.apache. org/hadoop/PoweredBy D. Borthakur, The Hadoop Distributed File System: Architecture and Design. The Apache Software Foundation, 2007.
[4] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/
[5] Yahoo! Launches World's Largest Hadoop Production Application, Yahoo! Developer Network, http://developer.yahoo.com/blogs/hadoop/2008/02/yahoo -worldslargest-production-hadoop.html.
[6] M. Zaharia, D. Borthakur, J. Sarma, et al., "Job scheduling for multiusermapreduce clusters," EECS Department University of California Berkeley Tech Rep UCBEECS200955 Apr, 2009–55. EECSDepartment, University of California, Berkeley. Retrieved from http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-55.pdf.
[7] Chan Tian, Haojie Zhou, Yougqiang He, Li Zha "A Dyanmic MapReduce Scheduler for Heterogeneous Workloads" Institute of computing technology ,Chinese academy sciences,china.
[8] Yongni tao, Lei Shi, Pinhua Chen "Job Scheduling Optimization for Multi-user MapReduce Cluser" Zengzhou University,china