Int. J. of Advanced Networking and Applications
Volume: 01, Issue: 05, Pages: 295-300 (2010)

Design and Implementation of a Two Level Scheduler for HADOOP Data Grids

Dr G. Sudha sadhasivam
Professor, Department of Computer Science & Engineering, PSG College of Technology,
Peelamedu, Coimbatore, TamilNadu 641004, India
Email:sudhasadhasivam@yahoo.com
Anjali M
ME Software Engineering, Department of Computer Science & Engineering,
PSG College of Technology, Peelamedu, Coimbatore, TamilNadu 641004, India
Email:anjalinairm@gmail.com

---ABSTRACT---

Hadoop is a large scale distributed processing infrastructure designed to handle data intensive applications. In a commercial large scale cluster framework, a scheduler distributes user jobs evenly among the cluster resources. The proposed work enhances Hadoop's fair scheduler that queues the jobs for execution in a fine grained manner using task scheduling. In contrast, the proposed approach allows backfilling of jobs submitted to the scheduler. Thus job level and task level scheduling is enabled by this approach. The jobs are fairly scheduled with fairness among users, pools and priority. The outcome of the proposed work is that short narrow jobs will be executed in the slot if sufficient resource is not available for larger jobs. Thus shorter jobs get executed faster by the scheduler when compared to the existing fair scheduling policy that schedules tasks based on their fairness of remaining execution time. This approach prevents the starvation of smaller jobs if sufficient resources are available.

Keywords: hadoop, scheduling, fair share scheduler, backfilling

Date of Submission: February 11, 2010 Date of Acceptance: April 08, 2010

1. Introduction

Hadoop [6] is a large-scale distributed processing infrastructure, designed to efficiently distribute large amounts of work across a set of machines. It is an open source project contributed by Yahoo and is licensed under Apache Software Foundation (ASF). Hadoop is based on the concept of moving computation to place of large data sets. A scheduler allocates multiple tasks or jobs submitted by multiple users to a set of resources. Some of the characteristics of a good scheduler include optimal processor utilization, good throughput, quick response time, better turnaround time, minimized job waiting time and fairness both users and resources.

The default job scheduler in Hadoop [6] has a first-in-first-out queue of jobs for each priority level. Other schedulers include the Facebook's Fair Scheduler [7], and Yahoo's Capacity Scheduler [6]. Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. In capacity scheduling, queues are guaranteed a fraction of the guaranteed capacity. The free resources allocated to any queue beyond its guaranteed capacity are reclaimed within N minutes of need.

Backfilling is the process of allowing small jobs from back of the queue to execute before large jobs that arrived earlier, due to lack of sufficient processors for the large jobs. Based on the number of maps and reduces in a job, backfilling can be applied to bring shorter jobs to front of queue such that they do not cause any delay to reserved jobs. There are two types of backfilling, namely, EASY and conservative. In EASY/aggressive backfilling [1], only job at head of the queue has reservation. Conservative backfilling [1] has reservation for every job.

In most of the commercially available systems, the default scheduling policy is FCFS, and in those management suites that also support backfilling, the governing scheme used is EASY. Jobs can be sorted according to priority, length of the jobs (short/long) and with backfilling [8]. Maui batch scheduler [9], is a simple FCFS batch scheduler, with a backfilling policy that maintains a time reservation for the first job in the queue using EASY backfilling. As the subfactor weights are set to zero, the job's queue time is the only factor that is not zero. So the prioritizing function is the queue time. The default scheduling of IBM's LoadLeveler [12] is FCFS and the default priority function is job's queue time. Backfilling is not set by default. If it is enabled, EASY backfilling policy is used. In platform's LSF[10] FCFS is default scheduling policy. Backfilling is not enabled by default, but when enabled, it's default behavior is similar to EASY. Default scheduler in PBS[d,e] is SJF. To prevent starvation of a job a time out is set (default is 24 hours). No other job can run until the starving job has completed. The system enters in draining mode under starvation. Backfilling is supported only in draining mode for specific queues. SGE [13] uses FCFS, and Equal-Share scheduling policies (a fair share scheduler). Currently, the system does not support backfilling. The workloadmanagement of OSCAR [2] is done using Maui or OpenPBS workload management systems.

compute resources and reports the status of previously submitted jobs to the user. The resource manager helps organize submitted jobs based on priority, resources requested, and availability. As shown in Fig 2.1, the scheduler receives periodic input from the resource manager regarding job queues and available resources, and makes a schedule that determines the order in which jobs will be executed.

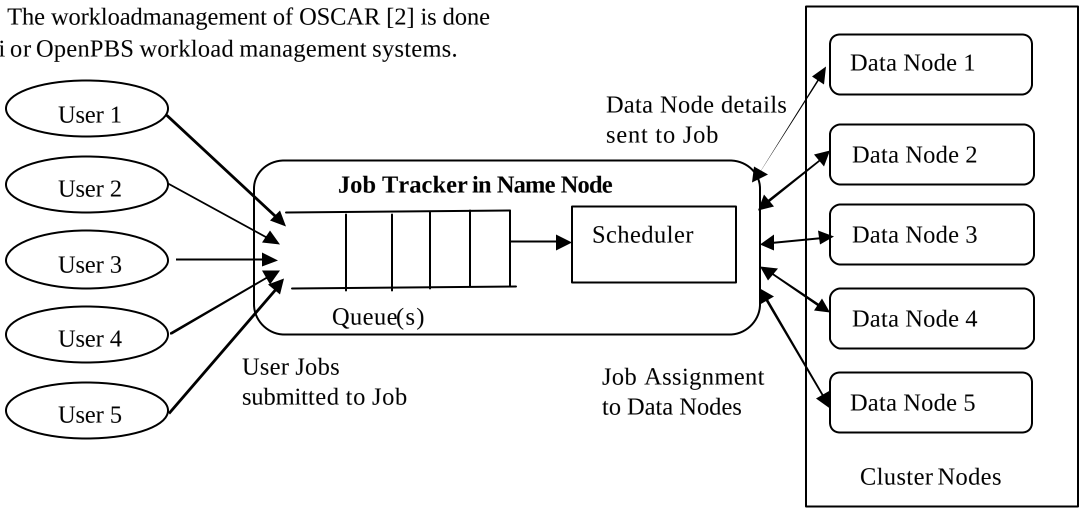


Fig 2.1: System Architecture

In Hadoop, the jobs are submitted by the end user to the jobtracker in the namenode. The scheduler in job tracker ensures that it should be fair to both the processes and the users. The fair share scheduler partitions the jobs fairly into tasks depending on the current machine load. Further, it does not utilize all slots if the task trackers are configured heterogeneously. Even though a fair amount of scheduling has been accommodated in the scheduler, it still is restricted to the task level. One of the best methods of scheduling that is currently available is the backfilling strategy. This is added to the scheduler to enhance the fairness of scheduling jobs in the job queue. Hence the proposed approach aims at designing a scheduler that ensures both job level and task level scheduling.

Section 2 and 3 provide the existing and proposed system architectures. Experimental analysis is presented in section 4.

2. Existing System

2.1 System Architecture

Typically, a resource management system in the name node of hadoop comprises a resource manager and a job scheduler (Fig 2.1). The scheduler communicates with the resource manager to obtain information about queues, loads on compute nodes, and resource availability to make scheduling decisions. The resource manager also sets up a queuing system for users to submit jobs. Users can query the resource manager to determine the status of their jobs. In addition, a resource manager maintains a list of available

Currently hadoop has the following schedulers:

1)Default Scheduler: It is the scheduler used in hadoop without any extra configuration. It schedules jobs in first in first out fashion irrespective of job size. The main drawback of this scheduler is starvation of small jobs in the event of resources being utilized by large jobs.

2) Capacity Scheduler: It schedules based on capacity of the resources. In capacity scheduling, queues are guaranteed a fraction of the guaranteed capacity. The free resources allocated to any queue beyond its guaranteed capacity are reclaimed within 'N' minutes of need.

3) Fair Scheduler: It is used to share MapReduce clusters among multiple users. Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. Fair share scheduling technique applied to clusters can also be applied on a Grid-wide scale [3],[4].

Fair share scheduler [5] ensures fairness to users and jobs. Fairness ensures that no later arriving job should delay an earlier arriving job. The Fair Scheduler arose out of Facebook's need to share its data warehouse between multiple users. Fair share scheduling is a way to guarantee application performance by explicitly allocating shares of system resources among competing workloads to balance between machine's actual share and entitlement. It ensures that over time, each job receives roughly the same amount of resources. Hence, shorter jobs will finish quickly, while longer jobs are guaranteed not to get starved. In fair sharing, the scheduler keeps track of a "deficit" for each job. Deficit is the difference between the amount of compute time it should have gotten on an ideal scheduler, and the amount of compute time it actually got. Every hundred milliseconds, the scheduler updates the deficit of each job. Whenever a task slot becomes available, it is assigned to the job with the highest deficit among the tasks meeting their pool capacity guarantees.

Let us consider 3 groups of users having three, two, and four jobs respectively. 33.33% of the available CPU cycles is distributed to each user. A three users are in group 1, each user gets 11.11% CPU cycles. Similarly group 2 users get 16.67% CPU cycles and group 3 users get 8.33% CPU cycles. 100% / 3 groups = 33.3% per group

2.2 Backfilling

Jobs submitted for scheduling can be broadly classified as short, long and very long jobs [1]. To handle these jobs, fair share approach can be enhanced by adding the backfilling strategy. Backfilling [8],[12] resolves the fragmentation problem caused by resource reservation and produces significant benefits in scheduling. Backfilling allows resource reservation for jobs which cannot be executed due to lack of processors. The algorithm scans the queue, and selects short jobs which can utilize the available resources as backfill jobs. Backfill jobs are scheduled to run before the resource-reserve job. In this way, backfilling dramatically improves system utilization and decreases the response time of short jobs. When using backfilling, users should provide an estimate of job's run time which is used by the scheduler to determine job's termination time and start time. If the requirements of the current job are not satisfied, it is queued. Whenever a job finishes using less than its allotted time, the algorithm tries to promote the existing jobs.

There are two types of backfilling, EASY and conservative. In EASY/ aggressive only job at head of the queue has reservation. Conservative backfilling has reservation for every job. With EASY Backfilling, short jobs can run in advance provided they do not delay the job at the head of the queue. The effects on other jobs will be ignored and the execution of other jobs may be delayed. Conservative Backfilling is not as aggressive as EASY because it only picks out jobs which make no delay of any previous job. It allows scheduling decisions to be made according to the submission time of job. The proposed work enhances the Fair Scheduler with EASY backfilling strategy of jobs based on their estimated time.

3. Proposed System Architecture

The proposed system brings about a job level scheduling in Fair Scheduler. This is done by using the backfilling scheduling. Based on the number of maps and reduces in a job, backfilling can be applied to bring shorter jobs to front of queue such that they do not cause any delay to reserved jobs. An EASY backfilling strategy is used as an initial phase. The jobs of various sizes are submitted to the scheduler.

The proposed approach is a two level scheduler which first schedules each pool jobs into a queue and then schedules the tasks for each of the queue. The Fair Scheduler groups jobs into "pools" and performs fair sharing between these pools. Each pool can use either FIFO or fair sharing to schedule jobs internal to the pool. In FIFO pools, jobs are ordered first by priority and then by submit time, as in Hadoop's default scheduler. In fair sharing pools, job priorities are used as weights to control how much share a job gets. A high-priority job gets more weight than a normal-priority job. Pools can be given weights to achieve unequal sharing of the cluster. Normally, active pools (those that contain jobs) will get equal shares of the map and reduce task slots in the cluster. The fair scheduler can limit the number of concurrently running jobs from each user and from each pool. The jobs that will run are chosen in order of submit time and priority. Jobs submitted beyond the limit, wait for one of the running jobs to finish.

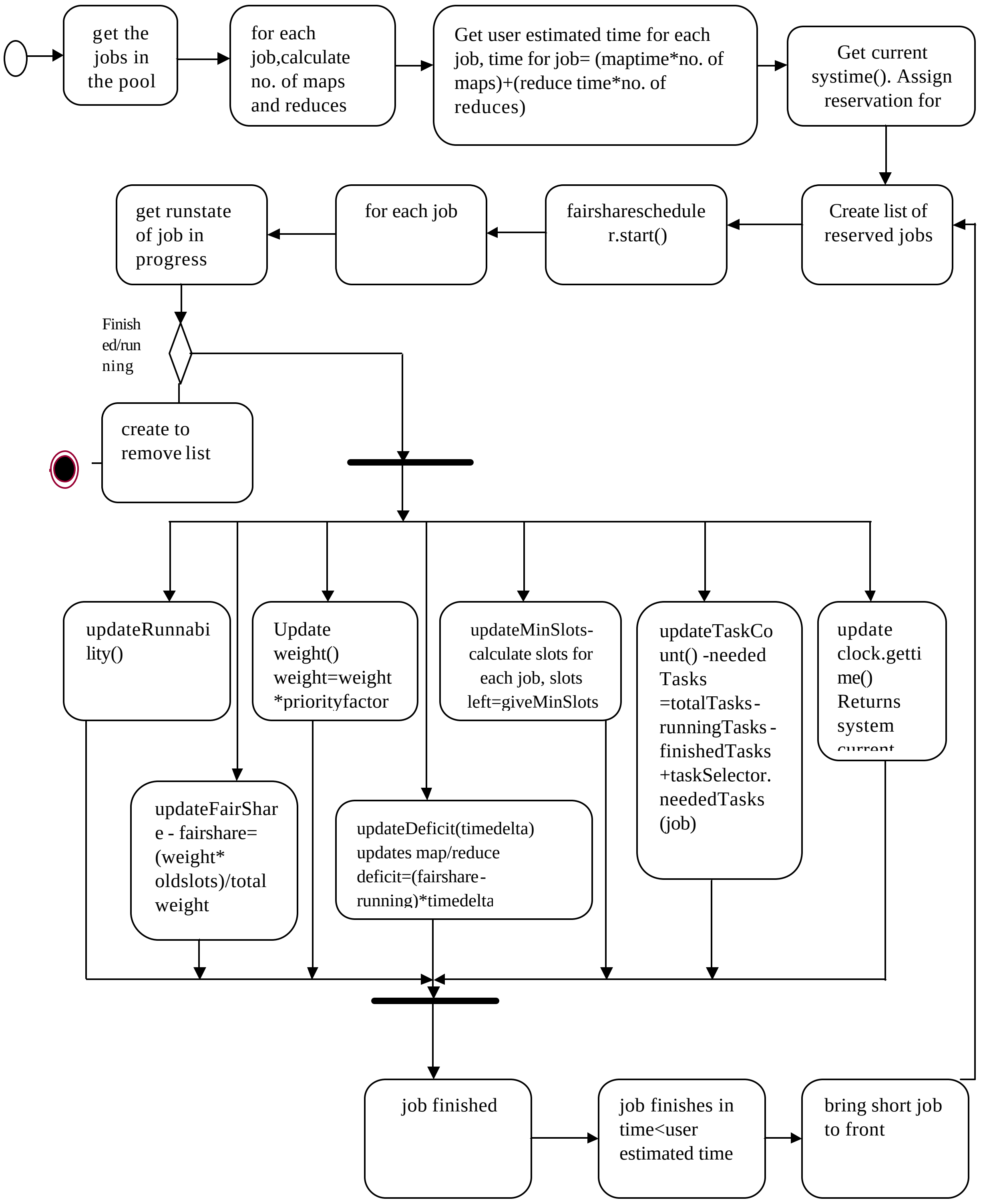
The sequence of operations in Fair Scheduler (Fig.3.1) with backfilling is listed as follows:

1. Group jobs into pools
2. For each job, calculate the number of maps and reduces
3. User estimated time for the job is calculated as

   Estimated time = (maptime*no. of maps)+(reduce time*no. of reduces)
4. Assign reservation for jobs based on priority which inturn decides the job's share.
5. If a job has completed execution, then remove it from the queue.
6. For each runnable job, allocate predefined number of minimum number of slots. If job requires more slots than predefined slots, it will be executed later. If job requires less slots then extra slots are given to other jobs.
7. Update weight as weight=weight * priorityfactor
8. Calculate Minimum Slots for each job.
9. Update TaskCount using num_Tasks = total_Tasks – running_Tasks –finished_Tasks + .needed_Tasks_for_job.
10. Update Fair share parameter as
    fairshare= (weight * oldslots) / totalweight
11. Update Deficit for every timedelta (500ms) as

    MR_deficit= (fairshare - running) * timedelta
12. If job job finishes earlier (in finish_time < user-estimated_time) then rearrange the queue and bring short job to beginning of the queue.
13. Goto step 6

4. Experimental Results


get the jobs in the pool
for each job,calculate no. of maps and reduces
Get user estimated time for each job, time for job= (maptime*no. of maps)+(reduce time*no. of reduces)
Get current systime(). Assign reservation for
Create list of reserved jobs
fairshareschedule r.start()
for each job
get runstate of job in progress
Finish ed/run ning
create to remove list
updateRunnability()
Update weight() weight=weight *priorityfactor
updateMinSlots- calculate slots for each job, slots left=giveMinSlots
updateTaskCount() -needed Tasks =totalTasks-runningTasks-finishedTasks +taskSelector. neededTasks (job)
update clock.gettime() Returns system current
updateFairShare - fairshare= (weight* oldslots)/total weight
updateDeficit(timedelta) updates map/reduce deficit=(fairshare-running)*timedelta
job finished
job finishes in time<user estimated time
bring short job to front

When the job is submitted by the user, it goes to the jobclient. The jobclient in turn passes the job to the jobtracker. The client can now choose the pool in which the job should be run, provided the pools are configured. The jobs are now sent to the scheduler. The scheduler resides within the jobtracker. Hence no RPC calls are required. The jobs are scheduled based on the backfilling strategy first, and then sent to the Fair Scheduler for the task level scheduling. The results of execution can be viewed from the user interface

The results were taken with pools having fair scheduler with a backfilling startefy and pools having fair scheduling with no backfilling. Equal number of map and reduce slots(5 each) were allotted to both the pools to ensure uniformity.

a) Case 1: When all the jobs submitted are of the same size, it is observed that the schedulers take almost same amount of time to complete as there is no variation in the size of jobs submitted.

Job	**Size (in Mb)**	**Scheduling without backfilling**		**Scheduler with backfilling**	
		Map time (sec)	Finish Time (sec)	Map time (sec)	Reduce time (sec)
1	2.5	4.23	5.07	4.33	5.11
2	2.5	4.48	5.01	4.42	5.09
3	2.5	4.41	5.07	4.45	5.09
4	2.5	6.34	6.50	6.50	6.61

Table 1 : Case 1 – Equal size jobs

b) Case 2: The proposed scheduler executes 2MB job prior to 2.5 MB job due to backfilling (TABLE 2).

Job	**Size (MB)**	**Scheduler without backfilling**		**Scheduler with backfilling**	
		Map time (sec)	Finish Time (sec)	Map time (sec)	Reduce time (sec)
1	1	.52	2.50	1.02	2.52
2	2	1.39	2.57	1.38	3.01
3	**2**	**3.32**	**3.34**	**2.39**	**3.08**
4	2.5	2.35	2.57	3.36	3.51

Table 2 : Case 2 – Medium sized jobs

c) Case 3: Initially two small sized jobs (1 MB), 1 job of size 2.5 MB and 1 job of size 3.5 MB size were submitted to the scheduler. Later during execution, a new job of size 1 MB was given, as shown in Figure 5, first 2 small sized jobs are completed. Then the third small job (1MB) are considered after the 2.5 MB job (when free slots were available).

Thus the short jobs get completed first and the overall completion time is decreased because the number of short and narrow jobs is higher. The backfilled scheduler can thus be applied where there are a large number of short jobs than less preferred long jobs.

Job	**Size (in Mb)**	**Scheduling without backfilling**		**Scheduler with backfilling**	
		Map time (sec)	Finish Time (sec)	Map time (sec)	Reduce time (sec)
1	3.5	5.58	6.15	5.18	5.37
2	1	1.34	2.49	.22	2.15
3	2.5	2.26	2.49	1.57	2.26
4	1	1.24	2.57	.55	2.26
5	1	5.49	6.23	4.45	5.36

Table 3 : Case 3 - More small jobs

5. Conclusion

Scheduling is the process of deciding how to commit resources between a variety of tasks. A scheduler is necessary when multiple tasks or jobs are submitted by multiple users to a set of resources. It is a very important component in a large scale distributed system. One of the critical requirements of a scheduler is that it should be fair to both the processes and the users. The proposed work enhances the existing schedule in hadoop with backfilling. This enhancement ensures both job level and task level scheduling. The outcome of the proposed work is t hat short narrow jobs will be executed faster by the scheduler when compared to the current scheduling policy. Hence the overall throughput of the system is improved.

6. Acknowledgements

The authors would like to thank Dr.Rudramoorthy, Principal, PSG College of Technology and Mr. Chidambaran Kollengode, Director, Cloud Computing Group, Yahoo Software Development (India) Ltd, Bangalore for providing the required facilities to complete the project successfully. This project is carried out as a consequence of the Yahoo's University Relation Programme with PSG College of Technology.

References

[1]. Adam K.L Wong, M.Goscinki, Evaluating the EASY backfill job scheduling of Static Workloads on clusters, *IEEE 2007 International conference on Cluster Computing*

[2]. B. des Ligneris, S. Scott, T. Naughton and N. Gorsuch, Open Source Cluster Application Resources (OSCAR): design, implementation and interest for the [computer] scientific community, *First OSCAR Symp.*, May 2003.

[3]. Bo Li, Dongfeng Zhao, Performance Impact of Advance Reservations from the Grid on Backfill algorithms, *IEEE Sixth International Conference on GCC,* 2007

[4]. Erik Elmorth, Peter Gardfjall, Design & Evaluation of a Decentralized System for Grid-wide Fair share Scheduling, *IEEE First International Conference on e-Science*,2005

[5]. Gerald Sabin, Garima Kochhar,Job Fairness in Non-Preemptive Job Scheduling, *IEEE ICPP*'04

[6]. Hadoop Fair Scheduler Design Document from jira Hadoop documentation

[7]. J. Kay and P. Lauder. A fair share scheduler. *Commun. ACM*,31(1):44–55, 1988.

[8]. Juan Wang, Wenming Guo, The Application of Backfilling in Cluster Systems, *IEEE International Conference on Communication and Mobile Computing,* 2009

[9]. MOAB workload manager (Maui scheduler) source code. http://www.supercluster. org/moab/. Version 3.2.6.

[10].Platform Computing Inc. Platform LSF. http://www.platform.com/products/LSFfamily/

[11].R. L. Henderson, D. G. Feitelson and L. Rudolph, Job scheduling under the Portable Batch System, Job Scheduling Strategies for Parallel Processing, *Lect. Notes Comput. Sci. vol. 949*, (Springer-Verlag, 1995), 279-294

[12]. S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. F. Skovira, *Workload Management with LoadLeveler*, *IBM,* first edition, Nov 2001. ibm.com/redbooks

[13].Sun Microsystems, Inc. Sun grid engine. http://gridengine. sunsource.net/, 2004.

[14].V. Systems, *Portable Batch System, Administrator Guide*, 2000 (OpenPBS Release 2.3)

Authors Biography

Dr *G Sudha Sadasivam* is working as a Professor in Department of Computer Science and Engineering in PSG College of Technology, India. Her areas of interest include, Distributed Systems, Distributed Object Technology, Grid and Cloud Computing. She has published 20 papers in referred journals and 32 papers in National and International Conferences. She has authored 3 books. She has coordinated two AICTE – RPS projects in Distributed and Grid Computing areas. She is also the coordinator for PSG-Yahoo Research on Grid and Cloud computing. You may contact her at sudhasadhasivam@yahoo.com

Anjali M is pursuing M.E in Software Engineering in PSG College of Technology, India. Her areas of interest include Grid and Cloud Computing. You may contact her at anjalinairm@gmail.com