SCH_ACR and SCH_LD based Job Scheduling Algorithm in Grid Environment

Deepti Malhotra
Assistant Professor
Department of Computer Science
Central University of Jammu, Jammu

ABSTRACT

To achieve the promising potentials by using tremendous distributed resources, effective and efficient scheduling algorithms are fundamentally important. Scheduling such applications is challenging because target resources are heterogeneous, their load and availability varies dynamically. Previous parallel system was assumed to be organized with homogeneous platform and connected via memory, bus, or LAN. But today its platform is heterogeneous and connected via Internet so each platform has different ability of computation performance and different network bandwidth. So, traditional list scheduling algorithms are inefficient to current parallel system. This research paper proposes and discusses in detail, the two new algorithms for Job scheduling on computational Grids so that the jobs are executed in minimum time and also all nodes of Grid execute equal load relative to their executing power. The main objective of this research paper is to allocate all the incoming jobs to the available computing power.

Keywords

Grid Computing, Job Scheduling, Scheduler, ACR, SCH_ACR, SCH_LD.

1. INTRODUCTION

Grid is a large scale distributed system, concerned with coordinated resource sharing and problem solving. The grid infrastructure provides a mechanism to execute applications over autonomous and geographically distributed nodes by sharing resources which may belong to different individuals and institutions [1]. Computational grid is a kind of grid environments, targeted at solving computationally intensive problems.

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [2]. It is a shared environment implemented via the deployment of a persistent, standards-based service infrastructure that supports the creation of, and resource sharing within, distributed communities. Resources can be computers, storage space, instruments, software applications, and data, all connected through the Internet. Since multiple applications may require numerous resources which often are not available for them so that in order to allocate resources to input jobs, having a scheduling system is essential. Because of the vastness and separation of resources in the computational grid, scheduling is one of the most important issues in grid environment [3]. Vast investigations have been done in this scope, which have led to theories and practical results [4, 5, and 6]. However new scheduling algorithms have been offered with emergence of grid computing. Objectives of scheduling algorithm are increasing system throughput [6], efficiency, and decreasing job completion time.

There are relatively a large number of task scheduling algorithms to minimize the total completion time of the tasks in distributed systems [7, 8, 9, 10, 11, and 12]. These algorithms try to minimize the overall completion time of the tasks by finding the most suitable resources to be allocated to the tasks. It should be noticed that minimizing the overall completion time of the tasks does not necessarily result in the minimization of execution time of each individual task.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Problem formulation for the proposed algorithms is given in Section 3. Section 4 discusses the details of the proposed scheduling algorithm i.e SCH_ACR. (which used maximum ACR (Available Computing Resource) as a selection mechanism).Section 5 discusses the details of the proposed SCH_LD based scheduling algorithm.Section6 gives the comparison results of SCH_ACR based scheduling with the SCH_LD based scheduling algorithm. Finally Section 7 concludes the paper by summarizing our contributions and future works.

2. RELATED WORK

Due to relatively high communication costs in grid environments most of the well known scheduling algorithms are not applicable in large scale distributed systems such as grid environments [7, 13, and 14]. There has been an ongoing attempt to build scheduling algorithms specifically within grid environments. Various algorithms have been proposed which in recent years each one has particular features and capabilities. In this section we review several scheduling algorithms which have been proposed in grid environment. In [15] a scheduling algorithm which is based on HQ-GTSM is presented. This algorithm not only takes into account the input jobs but also considers the resource migration time in deciding on the scheduling. One of the most important features of this algorithm is that it guarantees the grid quality of service. At the present time, job scheduling on grid computing is not only aims to find an optimal resource to improve the overall system performance but also to utilize the existing resources more efficiently.

X. He et al. have presented a new algorithm based on the conventional Min-min algorithm [7].The proposed algorithm which is called QoS guided Min-min, schedules tasks requiring high bandwidth before the others. Therefore,

if the bandwidth required by different tasks varies highly, the QoS guided Min-min algorithm provides better results than the Min-min algorithm. Whenever the bandwidth requirement of all of the tasks is almost the same, the QoS guided Min-min algorithm acts similar to the Min-min algorithm.

E. Elmroth et al. have proposed a user oriented algorithm for task scheduling in grid environments, using advanced reservation and resource selection [16]. The algorithm minimizes the total execution time of the individual tasks without considering the total execution time of all of the submitted tasks. Therefore, the overall makespan of the system does not necessarily get small.

F. Dong et al. have proposed a similar algorithm called QoS priority grouping scheduling [17]. This algorithm, considers deadline and acceptation rate of the tasks and the makespan of the wholes system as major factors for task scheduling. In comparison with Min-min and QoS guided Min-min, the QoS priority grouping scheduling algorithm achieves better acceptance rate and completion time for the submitted tasks.

K. Etminani et al. have proposed a new algorithm which uses Max-min and Min-min algorithms [12]. The algorithm determines to select one of these two algorithms, dependent on the standard deviation of the expected completion times of the tasks on each of the resources

B. Yagoubi et al. have offered a model to demonstrate grid architecture and an algorithm to schedule tasks within grid resources [18]. The algorithm tries to distribute the workload of the grid environment amongst the grid resources, fairly. Although, the mechanism used in [18] and other similar strategies which try to create load balancing within grid resources can improve the throughput of the whole grid environment, the total makespan of the system does not decrease, necessarily.

3. PROBLEM FORMULATION

Grid Scheduler is the important part of Grid Resource Management System (GRMS), which gathers the information about the resources and chooses the best resource as per the job requirements. This is followed by the actual execution of the jobs. The problem of finding the best "*job-resource*" pair is a compute-intensive problem and need to be formulated mathematically to find the optimal solution.

In the present setup, each Grid node consists of number of computational resources (processors, denoted by $P_{ni}\{\sum_{n=1}^{x} n = node\ no. | \sum_{i=1}^{4} i = processors\ no.\ in\ node\}$).The resources on the Grid are usually accessed via an executing "job". Each resource has a limited capacity (e.g., number of CPUs, amount of memory). It is assumed that each node consist of 4 processors. The specification of each processor is Intel(R) Core(TM)2 Duo CPU with different computing power.

To formulate the problem, we consider mapping a set of independent user jobs $\{J_1 J_2 J_3 \ldots\ldots, J_k\}$ to a set of heterogeneous nodes$\{n_1, n_2, n_3 \ldots . n_x\}$ and processors / resources $\{P_{n1}, P_{n2}, P_{n3}, P_{n4}\}$(Though there can be any type of resource that can be shared on a computational Grid, but for the simulations only nodes and processors of a node will be taken into consideration as a resource).This mapping is done with an objective of minimizing the execution time and utilizing the resources effectively, and also minimizing the delay in meeting user specified deadline.

The load of each node(L_n) is expressed as the summation of load of all the processors of a respective node.

$$\boldsymbol{L_n = L_{p_{n1}} + L_{p_{n2}} + L_{p_{n3}} + L_{p_{n4}}}$$

Any job *J* has to be processed by $P_{ni}\{\sum_{n=1}^{x} n = node\ no. | \sum_{i=1}^{4} i = processors\ no.\ in\ node\}$, until completion.

4. SCH_ACR BASED SCHEDULING ALGORITHM

In this section proposed Grid scheduling algorithm based on ACR is discussed in detail. It uses the ACR (Available Computing Resource) as a selection mechanism. In SCH_ACR based schedule all the processors of the nodes are accessed and the processor P_{ni} that has the maximum value of ACR is determined and then assigned to the job. The proposed scheduling algorithm allows single job request to be processed on multiple processors irrespective of the nodes. Assume the total number of nodes in a Grid as 3.Then all the 12 processors are accessed to determine the P_{ni} that has the maximum value of ACR.

4.1 ACR Representation

The GRD keeps track of the ACR (Available Computing Resources) of all the computers on the Grid. Here is the representation for ACR given in figure1:

$$ACR = CPU\_Frequency\ (MHz) * CPU\_Idle\ (\%)\ /100$$

Fig 1: ACR Representation

4.2 Selection

SCH_ACR uses the ACR as a selection mechanism. The main idea of this scheme is to predict the performance of each resource by estimating the maximum ACR of each processor and then maps the job with the resource/processor having maximum value of ACR. For example, in Figure2, job is allocated to the processor 1 of node 2. Consider P_{21} , which means processor 1 of node 2.

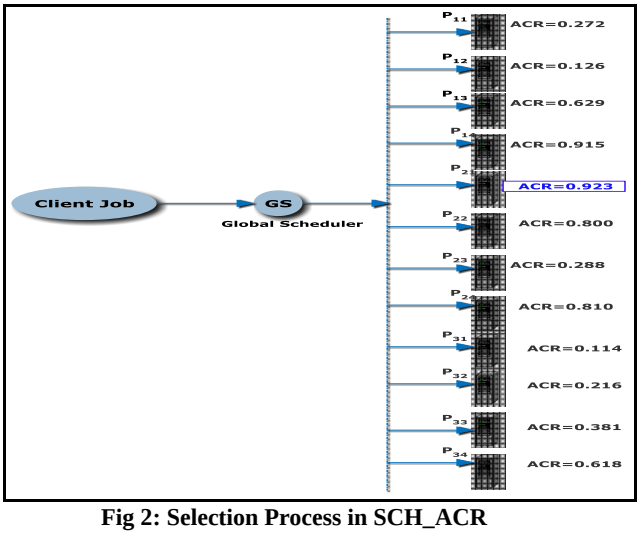

Fig 2: Selection Process in SCH_ACR

4.3 Flowchart

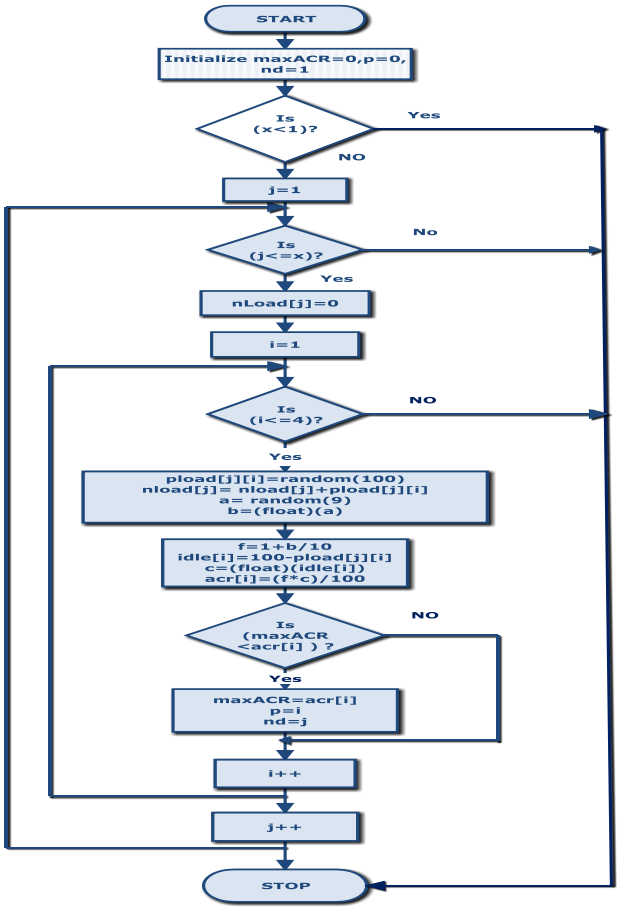

Fig 3: Flowchart of ACR based Algorithm

4.4 Algorithm Description

In step 1 of Algorithm given in section 4.4.1, the user's request is processed and split into individual job requests. Step 2, consists of list of all nodes available in the Grid. The actual scheduling process starts from step 3 that is repeated for each job request. In step 4, the scheduler discovers the available resources by contacting GRD (GridResource Database). Here resources are evaluated according to the requirements in the job request and only the appropriate resources are kept for further processing. Step 5 gives the total number of nodes. Variables are initialized in step 6. Step 7 checks if number of nodes is less than 1 then go to Step 11. Step 8 & 9 initializes the load of a node as zero. Step10 calculate the ACR of the processor and predict the performance of each resource by estimating the maximum ACR of each processor and then maps the job with the resource/processor having maximum value of ACR. Step11 is the end of the program.

4.4.1 Algorithm (PSEUDO CODE)

Step 1: cList= list of all individual requests by validating the client specification(s);

Step 2: nodeList =list of all nodes. Each Grid node comprises a number of computational resources (processors, denoted by P_{ni} $(\{\sum_{n=1}^{x}; n = \text{node no.} \mid \sum_{i=1}^{4} i = \text{processors no. in node}\})$;

Step 3: For each job do the following steps;

Step 4: Filter out the resources that do not fulfill the job requirements. Contact GRD (GridResource Database) to obtain a list of available resources;

Step 5: $x = total\ no\ of\ nodes;$

Step 6: $maxACR = 0, p = 0$;

Step 7: $If\ x < 1$

{

$go\ to\ step11;$

}

*/*Checking Nodes*/*

Step8: $for(j = 1; j <= x; j++)$

{

*/*initialize the load of a node as zero*/*

Step9: $nLoad[j] = 0;$

Step10: $for(i = 1; i <= 4; i++)$

{

*/*Calculate the load of a node */*

$pload[j][i] = random(100);$

$nload[j] = nload[j] + pload[j][i];$

$a = random(9);$

$b = (float)(a);$

/ Frequency for Processor*/*

$f = 1 + \frac{b}{10};$

/ Idle time for Processor*/*

$idle[i] = 100 - pload[j][i];$

$c = (float)(idle[i]);$

/ Available Computing Resource for Processor*/*

$acr[i] = (f * c)/100;$

$If\ (maxACR < acr[i])$

{

$maxACR = acr[i];$

/ Processor selected*/* $p = i;$

/ Node no of which processor is selected*/*

$nd = j;$

} */* End If*/*

*}/*End For */*

*} /*End For */*

Step11: exit.

5. SCH_LD BASED SCHEDULING ALGORITHM

SCH_LD is a scheduling algorithm which is based on the rule that first the selection of a light-loaded processing node takes place then the job is allocated to the processor of a selected node having maximum available CPU resource (ACR). The proposed scheduling algorithm allows single job request to be processed on multiple nodes. The SCH_LD schedule introduces NS (node-selection) at the global scheduler and the PS (processor selection) at the local scheduler. The NS is based on the rule that the light-loaded processing node is selected for the job allocation. This technique fetches the jobs from the Global job queue that is ready to execute and assign these jobs to the best nodes (i.e. node having minimum value of a load) of the Grid. The PS (processor selection) schedule the job to the processor of a selected node having maximum available CPU resource (ACR). So this selection mechanism has the advantage of lesser number of computations as compared to the SCH_ACR based scheduling.

5.1 Selection

SCH_LD is a scheduling algorithm which uses the load of a node and ACR (Available Computing Resource) of a processor as a selection mechanism. In SCH_LD based schedule assigning job to the resource is a two step process-First the lightly loaded node among all the nodes in a Grid is selected; then in the second step only the processors of the selected node are estimated for the maximum value of ACR. Thus, SCH_LD based schedule help in mapping jobs to resources allows good solutions to be found quickly. Table1 and Figure 4 show the selection process by considering the number of nodes is equal to 3.

Step1: Estimation of lightly loaded node

Table 1: Load Estimation

$L_{p_{11}}$ = 34	$L_{p_{21}}$ = 46	$L_{p_{31}}$ = 91
$L_{1_{22}}$ = 59	$L_{p_{22}}$ = 51	$L_{p_{32}}$ = 86
$L_{p_{13}}$ = 33	$L_{p_{23}}$ = 31	$L_{p_{33}}$ = 58
$L_{p_{14}}$ = 91	$L_{p_{24}}$ = 28	$L_{p_{34}}$ = 77
L_1 = 217	L_2 = 156	L_3 = 312

Node Selected = 2

***Step2*:** Processor Selection for Node 2

Available Computing Resource for P_{21}= 0.923 (maximum value of ACR)

Available Computing Resource for P_{22} =0.800

Available Computing Resource for P_{23}=0.288

Available Computing Resource for P_{24} =0.810

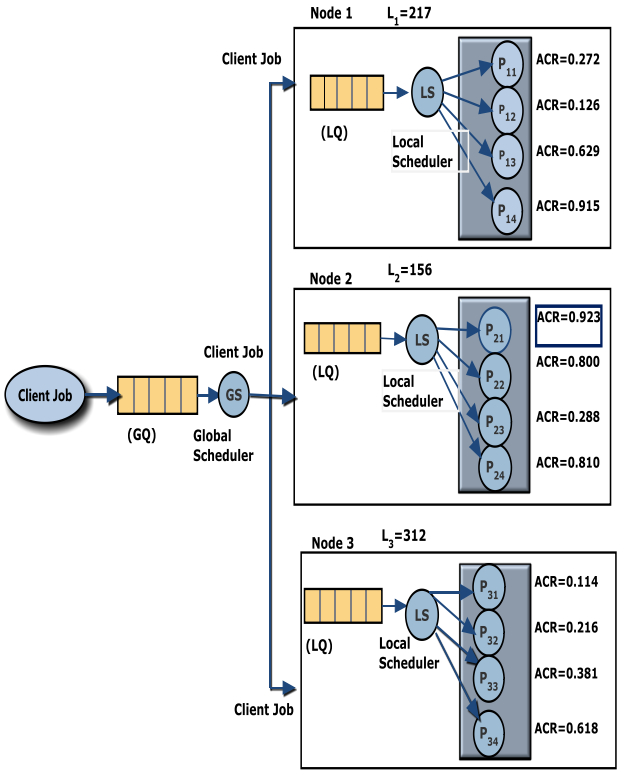

Fig 4: Selection Criterion for SCH_LD based Scheduling Algorithm

5.2 Algorithm Description

In this section, we describe the routing strategies involved in the Grid model, which is integrated by the node-selection (NS) at the global scheduler (GS) and the processor selection (PS) at the local scheduler (LS).

In step 1 of Algorithm given in section 5.2.1, the user's request is processed and split into individual job requests. Step 2, consists of list of all nodes available in the Grid. The actual scheduling process starts from Step 3 that is repeated for each job request. In step 4, the scheduler discovers the available resources by contacting GRD (GridResource Database). Here resources are evaluated according to the requirements in the job request and only the appropriate resources are kept for further processing. Step 5 gives the total number of nodes. Variables are initialized in Step 6. Step 7 checks if number of nodes is less than 1 then go to Step14. Step 8 calculate the load for each combination of processor i and node j. In Step 9 $nLoad[1]$ is assigned as minimum loaded node.Step10 predicts the performance of each resource by estimating the minimum load of each node and then maps the job with the resource/node having minimum value of load. Step11 calculate the ACR of each processor of selected node by using the frequency and idle time of each processor. Initialization of variables $maxACR$ and p takes place in Step 12.In Step 13 the job is assigned to the processor having maximum value of ACR. Step 14 is the end of the program.

5.2.1 Algorithm (PSEUDO CODE)

Step 1: cList= list of all individual requests by validating the client *specification(s);*

Step 2: nodeList =list of all nodes. Each Grid node comprises a number of computational resources (processors, denoted by P_{ni} $\{\sum_{n=1}^{x} n = \text{node no.} \mid \sum_{i=1}^{4} i = \text{processors no. in node}\}$);

Step 3: For each job do the following steps;

Step 4:Filter out the resources that do not fulfill the job requirements. Contact GRD (GridResource Database) to obtain a list of available resources;

Step 5: $x = total\ no\ of\ nodes;$

Step 6: $int\ \ nLoad[200]\ , p = 0, nd = 1;$

Step 7: $If\ x < 1$

{

$go\ to\ step14;$

}

*/*Checking Nodes for selection)*/*

Step 8: $for(j = 1; j <= x; j++)$

{

*/*initialize the load of a node as zero*/*

$nLoad[j] = 0;$
$for(i = 1; i <= 4; i++)$

{

*/*Calculate the load of a node */*

$pload[j][i] = random(100);$

$nload[j] = \ nload[j] + pload[j][i];$

} */*End For */*

}*/*End For */*

Step 9: $minLOAD = nLoad[1];$

/ Selection for lightly loaded node*/*

Step10: $for\ (k = 1; k <= x; k++)$

{

$If(minLOAD > nLoad[k])$

{

$minLOAD = nLoad[k];$

$nd = k;$

} */*End If */*

} */*End For*/*

Step 11: $for(i = 1; i <= 4; i++)$

{

$a = random(9);$

$b = (float)(a);$

$f = 1 + \frac{b}{10};$

*/*Calculate the idle time of each processor of selected Node n */*

idle[i]=100-pload[nd][i];

$c = (float)(idle[i]);$

/ Available Computing Resource for Processor*/*

$acr[i] = (f * c)/100;$

} */*End For*/*

Step 12: $maxACR = ACR[1], p = 1;$

/ Processor selection for maximum ACR*/*

Step13: $for\ (k = 1; k <= 4; k++)$

{

$If\ (maxACR < \ ACR[k])$

{

$maxACR = \ ACR[k];$

p $= k;$

} */*End If */*

} */*End For*/*

Step14: exit

5.3 Flowchart

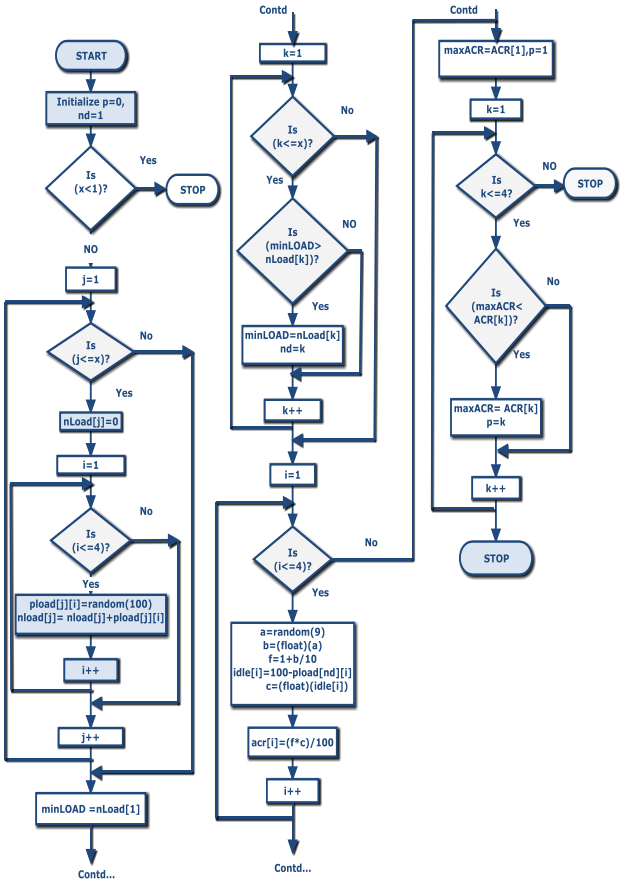

Fig5: Flowchart of SCH_LD based Algorithm

6. COMPARISON BETWEEN SCH_ACR AND SCH_LD

Next experiments were carried on to compare the ACR based scheduling algorithm (SCH_ACR) and the light-loaded processing node based scheduling algorithm (SCH_LD).Table 2 shows the comparison result obtained between SCH_ACR and SCH_LD based schedule in terms of number of computations. Thus, SCH_LD based scheduling algorithm works better than SCH_ACR based scheduling algorithm.

Table 2: No of Computations in SCH_ACR and SCH_LD based Scheduling

No of nodes	SCH_ACR	SCH_LD
2	32	24
4	64	32
6	96	40
8	128	48
10	160	56

Table3: Analysis of number of computations

Number of computations required to calculate	SCH_ACR	SCH_LD
No of Nodes	2	2
No of Processors	8	8
Load of all processors	8	8
Load of all processors of selected node	NA	4
Frequency of all Processors	8	4
Idle Time of all Processor	8	4
ACR of all Processors	8	4
Total number of computations	8+8+8+8=32	8+4+4+4+4=24

Table3 gives the calculation of total number of computations required in SCH_ACR and SCH_LD based schedule when the number of nodes is equal to 2. It is clear from the table that the number of computations required in computing the *load of all processors* in SCH_ACR and SCH_LD is 8 and 8 respectively. Similarly, computing *load of all processors of selected node* in SCH_ACR and SCH_LD is zero and 4 respectively. To calculate the *frequency of all processors* in SCH_ACR and SCH_LD based schedule, no. of computations required are 8 and 4 respectively.

Idle Time of all Processor is computed by taking 8 computations in case of SCH_ACR and 4 computations in case of SCH_LD. To compute the ACR of all processors the no. of computations required in SCH_ACR and SCH_LD based schedule is 8 and 4 respectively. Last row of table3 gives the total number of computations i.e. 32 and 24 respectively in case of SCH_ACR and SCH_LD based schedule when the number of nodes is equal to 2.

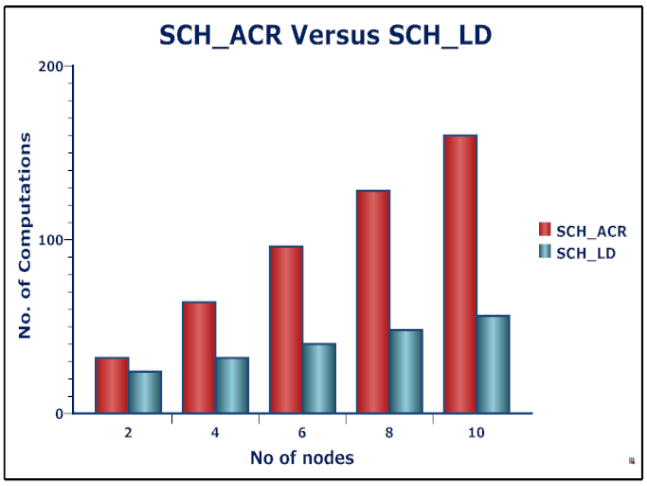

Fig 6: Comparison results for SCH_ACR and SCH_LD

Graph in Figure 6 show the comparison of SCH_ACR based schedule and the SCH_LD based schedule. X-axis of the graph consists of number of nodes and Y-axis consists of number of computations. From the graph, it is clear that the number of computations in case of SCH_ACR is always greater than the number of computations in SCH_LD. Thus, SCH_ACR works better than SCH_LD.

7. CONCLUSION AND FUTURE WORK

The success of grid computing will depend on the effective utilization of the system for various computationally intensive jobs. Given a vast number of resources that are available on a Grid, an important problem is the scheduling of jobs on the grid with various objectives. There are relatively a large number of task scheduling algorithms to minimize the total completion time of the tasks in distributed systems. These algorithms try to minimize the overall completion time of the tasks by finding the most suitable resources to be allocated to the tasks. It should be noticed that minimizing the overall completion time of the tasks does not necessarily result in the minimization of execution time of each individual task. The above mentioned algorithms SCH_ACR and SCH_LD have adopted an application-centric scheduling objective function taking the execution time parameters into considerations, which tries to take care of user specified deadlines, as well as tries to ensure the maximum utilization of resources. The main objective of this research paper is to allow all the incoming applications to be allocated to the available computing power.

Now we discuss some of the limitations of this work and present some possible directions for future research. In this work, we assume that there is no precedence constraint among different jobs or different tasks of a job. Usually, the jobs are independent of each other in the grid, but different tasks of a job may have some precedence constraints. Hence, it is an interesting direction for future research. Such dependencies will not only make the problem extremely difficult to solve, but would also require estimating a very large number of parameters. In the future we should also consider some fault tolerant measures to increase the reliability of our algorithm.

8. REFERENCES

[1] I. Foster, and C. Kesselman.2004.The Grid 2: Blueprint for a New Computing Infrastructure, Second Edition, Elsevier and Morgan Kaufmann Press.

[2] I. Foster and C. Kesselman (editors).1999. The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, USA.

[3] Rajkummar Buyya.2002.Economic-based Distributed Resource Management and Scheduling for grid computing.PhD thesis, Monash university, Melborn, Australia.

[4] K. Al-Saqabi, S. Sarwar, and K. Saleh.1997. Distributed gang scheduling in networks of heterogeneous workstations, Computer Communications Journal, pp.338-348.

[5] Maheswaran M, Ali S, Siegel H J, et al.1999.Dynamic mapping of a class of independent tasks on to heterogeneous computing systems. In the 8th IEEE Heterogeneous Computing Workshop (HCW '99),San Juan, Puerto Rico,(Apr. 1999), pp.30-44.

[6] XiaoShan He, XianHe Sun, and Gregor von Laszewski.2003.QoS Guided Min-Min Heuristic for Grid Task Scheduling, Computer Science and Technology, 18(4):442-451.

[7] X. He, X-He Sun, and G. V. Laszewski.2003.QoS Guided Min-min Heuristic for Grid Task Scheduling, Journal of Computer Science and Technology, Vol. 18, pp. 442-451.

[8] M. Maheswaran, Sh. Ali, H. Jay Siegel, D. Hensgen, and R. F. Freund.1999. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems, Journal of Parallel and Distributed Computing, Vol. 59, pp. 107-13.

[9] T. D. Braun, H. Jay Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao.2001.A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems,Journal of Parallel and Distributed Computing, Vol. 61, pp. 810-837.

[10] F. Dong, J. Luo, L. Gao, and L. Ge.2006.A Grid Task Scheduling Algorithm Based on QoS Priority Grouping," In the Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC'06), IEEE.

[11] E. Ullah Munir, J. Li, and Sh. Shi.2007. QoS Sufferage Heuristic for Independent Task Scheduling in Grid. Information Technology Journal, 6 (8): 1166-1170.

[12] K. Etminani, and M. Naghibzadeh.2007.A Min-min Max-min Selective Algorithm for Grid Task Scheduling,The Third IEEE/IFIP International Conference on Internet, Uzbekistan.

[13] B.T. Benjamin Khoo, B. Veeravalli, T. Hung, and C.W. Simon See.2007.A multi-dimensional scheduling scheme in a Grid computing environment," Journal of Parallel and Distributed Computing, Vol. 67, pp. 659-673.

[14] B. Yagoubi, and Y. Slimani.2007.Task Load Balancing Strategy for Grid Computing, Journal of Computer Science, Vol. 3, No. 3, pp. 186-194.

[15] Huyn zhang, chanle wu, Q.xiong, and L.Wu,G. Ye. 2006. Research on an Effective Mechanism of Task Scheduling in Grid Environment. In IEEE, Fifth International Conference on Grid and Cooperative Computing (GCC'06).

[16] E. Elmroth, and J. Tordsson.2008.Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions," Journal of Future Generation Computer Systems, Vol. 24, pp. 585-593.

[17] F. Dong, J. Luo, L. Gao, and L. Ge.2006.A Grid Task Scheduling Algorithm Based on QoS Priority Grouping," In the Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC'06), IEEE.

[18] B. Yagoubi, and Y. Slimani.2007.Task Load Balancing Strategy for Grid Computing," Journal of Computer Science, Vol. 3, No. 3, pp. 186-194.