Proposing an Advanced Dynamic Resource Reservation (ADRR) Framework for Grid Computing

Dr. Shobha Bhatia

Assistant Professor, Department of Computer Science, DAV Institute of Management, Faridabad

Abstract

Grid resource management plays an important role while enabling the sharing and coordinating of resources in Grid computing environments. Resource reservation is an important part of the Grid resource management. Currently existing resource management systems provide only means to reserve resources starting with the reservation attempt and lasting for an unspecified duration. However, most of the approaches rely on administrators' intervention for proper allocation. In this paper, we propose dynamic resource allocation mechanisms of computations using a combination of best fit algorithm and process migration.

Keywords: *Grid Computing, Resource Allocation, Dynamic Resource Reservation, Process Migration.*

1. INTRODUCTION

The last decade has witnessed tremendous progress of various distributed computing infrastructures aiming to support Internet-wide resources collaboration and sharing. Grid technology flourished worldwide in the early 2000's, and grid community requires participants to exhibit some degree of trust, accountability, and opportunities for sanctions in response to inappropriate behaviour. This relatively close community facilitates Quality of Service (QoS) guarantee mechanism. Virtualization, a software-based technology for building shared hardware infrastructures in Grid computing, helps to achieve greater system utilization while lowering total cost of ownership and responding more effectively to changing business conditions. Cloud computing also employs virtualization technology to provide dynamic resource pool. Therefore, virtualization will be a widely used technology for next generation computing platforms. Various resources are virtualized as a resource pool, and are managed as a whole.

The work of this paper is to develop an Advanced Dynamic Resource Reservation (ADRR) technique which is empowered with all the desired capabilities which can provide solutions for the identified problem areas in a grid scenario. To demonstrate the usability of proposed techniques, a simulation test bench was implemented using the GridSim and successful simulation was achieved. GridSim toolkit supports simulation of various types of grids and application models scheduling. GridSim is an opensource software platform, written in Java, which provides features for application composition, information services for resource discovery, and interfaces for assigning applications to resources. The GridSim toolkit also provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers and schedulers. It can be used to simulate application schedulers for single or multiple administrative domain(s) distributed computing systems such as clusters and grids. The results show that the ADRR performs better than the other algorithms by reducing both the reservation duration time and the total completion time, increase resource utilization and reduce no of rejections, etc

2. PROPOSED METHODOLOGY

The Advanced Dynamic Resource Reservation Model (ADRR) is shown in Figure 1. When the user submits the job to the job handler, job handler goes through five states. In the first state, users are sorted according to their priorities. In the second state, the job handler identifies the job as new or mature and forwards the job to the Dynamic Priority Resolution (DPR). When the job handler identifies the job as new, it forwards the new job with user requirements to the DPR. When the job handler identifies the job as mature, it forwards the mature job and the matched job Id to the DPR. The DPR queues the new job in the new job queue and the mature job in the mature job queue. In the third state, the jobs of each user are sorted based on the lengths of jobs. In the fourth state, the jobs of various users are sorted in accordance with the job's priority based on job length and on priority of the user to which the jobs belong. This is done by Dynamic Priority Resolution (DPR) technique.

Gridlet Sorting Policy (GSP) is based on a predefined value decides that how many jobs of a given user are to be scheduled and send to reservation. Finally in the last state, jobs are placed in a queue called Gridlet queue from where they are sent to resource analyst. The resource analyst comprises of two components viz. User Resource Information (URI) and Rank List Unit (RLU). The User Resource Information (URI) receives the new job with the user requirements from the job handler and the Rank List Unit (RLU) receives the mature job with the matched Job Id from the job handler.

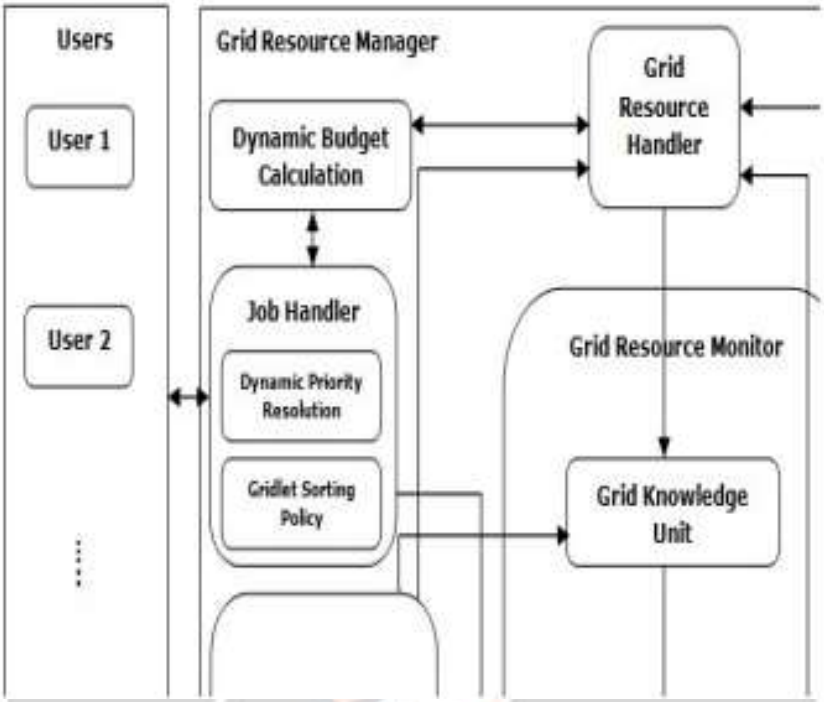

Figure 1: ADRR Framework

The User Resource Information (URI) maintains Resource Information Table and the Rank List Unit maintains Rank List Table. For mature jobs, the Rank List Unit (RLU) fetches the rank list of the matched job Id from the rank List table and forwards the mature job with the rank list to the resource handler for allocation. For new jobs, the User Resource Information (URI) performs the preliminary selection of resources by considering the static parameter (Resource name, Resource location, Resource CPU type, Resource OS, and Memory capacity) resource type of resources. The resource handler proposes an initial budget of resources to be given to each new process. This budget is initialized with no prior knowledge of the process resource requirement rather on the current request made by it. However, this budget is updated to accommodate the resource requests made by a process. Thus, every time the process makes a new resource request within the budget allotment is done using available resources and reservation pool. However, if the requested resources are beyond the budget granted to a process, then the budget is modified to accommodate the new request. The resources capabilities selected in the preliminary selection and the job with user requirements is forwarded to the Resource Search Engine (RSE). The Resource Search Engine (RSE) prepares a new rank list for the new job and forwards the job and the new rank list to the resource handler for allocation.

A. Grid Resource Manager

The Grid manager is responsible for managing all grid activities. It is an interface between the user and the service provider. It has job handler, grid resource handler, resource search engine, dynamic budget calculation and grid resource monitor.

1. Job handler

The functions of the job handler are, when the user wants to register itself, the user sends the user_registration message to the job handler. The information field contains the metadata given by the user at the time of registration. The job handler receives the user_registration message. It assigns the user a unique user_id and sends the user_ack message to the user as an acknowledgement. The user receives the user_id and utilizes the user_id to submit the job to the job handler. The user submits the job by sending the job_submission message to the job handler. The job_submission message includes the requirements of the user and the executable data.

After the user submits the job, the job handler assigns the submitted job a unique job_id. The job handler sends the user the assigned job_id as an acknowledgement for the received job. The job handler maintains the job table. The job handler stores the user requirements of the job along with the user_name, user_id, job_name, job_id, arrival date of the job, and etc., in the job table.

Whenever a user enters grid, it is enlisted by the proposed algorithm in a queue called User List referred by job table. The users from the User List are sorted based on their priority and are placed in a queue called User queue. Thereafter jobs that are generated by various users are ordered on the basis of their lengths and are placed in another queue named Gridlet List referred by job table. The jobs are prioritized again. The proposed algorithm prioritizes them by using a Gridlet Sorting policy and Dynamic Priority Resolution (DPR) technique. The sorting policy decides on the number of jobs of each user to be prioritized and finally to be sent for reservation. Thus, the Sorting policy ensures that jobs of various users are entertained in a way so as to avoid starvation among the jobs. On the other hand, the DPR technique is used to resolve the priority of jobs by considering the parameters associated with jobs priority. The DPR therefore is consequential in determining the priority of incoming jobs having higher user priority and higher job priority based on the job length. The proposed algorithm finally stores prioritized jobs in a queue called Gridlet queue. The jobs thus scheduled by proposed algorithm are submitted to resource allocation.

The priority work done by proposed algorithm can be broadly divided into two phases namely User Priority Phase and Gridlet Priority Phase. The first phase deals with the user priority whereas the second phase takes care of job priority of the jobs.

(i) User Priority Phase

These users can have one or multiple jobs. In this phase, first the users are sorted and then the jobs of each user are sorted. The users are sorted in decreasing order of their priorities using User Sorting Policy. If the users have equal priorities, they are arranged on first come first basis else insertion sort technique is used. Once the users have been sorted, jobs of each user are sorted on the basis of their lengths. The jobs are sorted using insertion sort technique. The jobs thus sorted are placed in the Gridlet List. This technique is used as it saves time and therefore is more efficient. This is due to the fact that insertion sort has complexity of O(n) which is comparatively less than the time complexities O(nlog(n)) and O(n2) of other sorting methods in the best or worst case respectively.

(ii) Gridlet Priority Phase

In this phase, jobs already sorted in User priority phase are further prioritized based on two criteria. This criterion can either be User Priority or job length or both referred to as User High Priority (UHP) phase and User Job Priority (UJP) phase respectively. DPR technique decides whether jobs will be sorted in UHP or UJP phase depending on the difference in priority of the jobs (that is User Priority). DPR technique takes two users at a time that are competing for reservation and compares their User Priority. If the difference between User Priority is larger than significant difference SD, only the jobs belonging to the higher priority user are sorted. The jobs belonging to lower priority user are queued up. These jobs are treated as pending jobs which will further participate in the next sorting process in the User priority phase. Out of these sorted jobs of a higher priority user, only a predefined number of jobs are considered for reservation (as decided by pre-defined function) and rest are discarded. This is referred to as UHP phase. Whereas if the difference between the User Priority of the two users competing for reservation is less than a significant difference SD, the priority of jobs is evaluated on the basis of their job length and User Priority. SD ensures that competing jobs are well differentiated among themselves. As the jobs keep arriving, the sorting process is repeated and the pending jobs of lower priority users are allowed to participate again and hence they are prevented from getting starved. This phase is known User Job Priority phase.

(iii) Gridlet Sorting Policy

The sorting technique used to sort the jobs in the above phases is decided by GSP. The GSP uses shortest first job technique for sorting the jobs of various users. This policy also uses a pre-defined function in UHP or UJP phase to decide as to how many jobs of high priority user are to be considered for reservation. The policy prevents all the jobs of high priority user from getting entertained and therefore avoids starvation of jobs of low priority users. The policy takes this decision on the basis of a pre-defined value.

- Start.
- Create a User List and add users to this list as users arrive and join the grid.
- Sort the users in the User List based on assigned User Priority using insertion sort following User Sorting Policy.

- Create a Gridletlist and insert Gridlets in this list according to their User Priority such that all Gridlets of each user are placed in a sequence.
- Sort the Gridlets of various users joined.
- Dynamic Priority Resolution (DPR) is calculated at runtime by using DPR = User[i].priority -User [i+1].priority
- Compare DPR with SD which is significant difference between User Priority. If (DPR>SD)

User High Priority (UHP) phase: Sort the Gridlets of each user according to shortest order and place into a user's Gridlet queue. Consider only N number of shortest jobs of higher priority user to be sent for reservation schedule as decided by GSP. Else

User Job Priority (UJP) phase: Sort the Gridlets of users according to GSP, taking into account the User Priority of both the users which being compared, are placed into a queue. This leads to Gridlets from both the users in the Gridlet list.

- If (new user)

  Go to step 2

- If (new Gridlet)

  Go to step 4

- Stop

After the dynamic priority resolution done the result submits to the job handler, the job handler verifies the Gridlet queue to know whether the submitted job has been previously processed, by calculating the similarity Index SI of the submitted job with all jobs in the job table whose time of arrival is less than Δt (where Δt is the time interval at which the resource's dynamic and behavioral parameters are updated in the resource analyst by the grid knowledge unit). The similarity index SI for each submitted job is calculated using the Equation.

$$SI = \frac{2n_{xy}}{(n_x + n_y)}$$

Were,

n_{xy} is the number of parameters matched with the job in the job table.

n_x is the number of parameters of the job in the job table

n_y is the number of parameters of the submitted job

If the submitted job's similarity index SI is equal to 1 with respect to a particular job in the job table, the job handler identifies the submitted job as 'mature'. The job handler then sends the job to the resource analyst by sending the job_description message. If the submitted job's similarity index SI is not equal to 1 with respect to a particular job in the job table, the job handler identifies the job as 'new'. The job handler then sends the job to the resource analyst by sending the job_description message for analyze the initial budget values.

When the user wants to know the status of the submitted job, the user sends a status_request message to the job handler to retrieve the status of the submitted job. The job handler receives the status_request message from the user. The job handler forwards the status_request message to the resource handler. The resource handler fetches the status of the submitted job from the resource table in the resource handler and sends the status_ response message to the job handler. The job handler forwards the status response message to the user.

2. Resource handler

The functions of the resource handler are, when the service provider wants to register itself with the grid, the service provider sends a service provider_registration message to the resource handler. The resource handler receives the service provider_registration message. It assigns the service provider a unique service provider _id and sends the service provider_ack message to the service provider as an acknowledgement. The service provider receives the service provider_id and uses the service provider_id to register the resource with the resource analyst through the resource handler. The service provider registers the resource capabilities (static and dynamic parameter values) with the resource analyst through resource handler by sending the

resource_registration message to the resource handler. The resource handler receives the resource capabilities from the service provider. The resource handler assigns a unique resource_id to the registered resource and sends the resource_registration_ack message to the service provider as an acknowledgement. The resource handler forwards the registered resource's capabilities to the resource analyst by sending the resource_submission message to the resource analyst. The resource handler maintains a resource table. For each newly submitted job, the resource search engine calculates a rank_list. The resource search engine then sends the rank_list to the resource handler. The resource handler allocates the highly ranked resource in the rank_list to the job. The resource handler stores the rank_list in the resource table for rescheduling of resources using initial budget in case of resource/job failures. The resource handler stores the rank_list, job status and the resource status in the resource table.

3. Dynamic budget calculation using threshold

The assumption that the resource requirement is known in advance is serious limitation of most of the deadlock avoidance techniques as this estimation has considerable overhead. The present work used a dynamic budget technique using threshold to overcome the limitation of existing technique of assuming that the maximum resources required by each process is known in advance.

The resource handler proposes an initial budget of resources to be given to each new process. This budget is initialized with no prior knowledge of the process resource requirement rather on the current request made by it. However, this budget is updated to accommodate the resource requests made by a process. The safety sequence test ensures that the system is deadlock free with the proposed budget. Thus, every time the process makes a new resource request within the budget allotment is done using available resources and reservation pool. However, if the requested resources are beyond the budget granted to a process, then the budget is modified to accommodate the new request. To ensure the deadlock Free State, the safety sequence is estimated. If the system is safe the requested resources are granted. However, if the safety sequence does not exist the process state is stored (check pointed) and queued in the pending state with its budget updated to the actual resource request. This process may be aborted to prevent the hold and wait condition which may lead to a deadlock. However, this can be postponed till some process requests for the resources held by this pending process. This will reduce the repercussions of over budgeting.

Begin

```
1. Initialize Threshold_Value[y] =Reserve[y] =[[Request[x][y]
∀x=1,2…n], 0]∀ y=1, 2, …m
2. Initialize Budget_Value[x][y]= Allocation [x][y] = Need[x][y]=
{0}∀x=1,2…n and ∀y=1, 2, …m
3. Available[y]= Available[y]-Reserve[y] ∀ y=1, 2, …m
4. For (Request[x][y] by a process Pi)
Do
        A. If Request[x][y]>Need[x][y] for any y = 1, 2, … m
        I.  Budget_Value[x][y]  =  Allocation[x][y]  +
Request[x][y]+ Threshold_Value[x][y]
        II. If Safety Sequence exists
                i. Accept process Px
                ii. If (Request[x][y]<=Available[x][y])
                        a. Allocate (Allocation, Request,
                Available, 0)
                        Else
                        b. Allocate (Allocation, Need,
                Available, Reserve)
                iii. Request[x][y] ={0}
                Else
                iv. Save the status and insert the process Pi in
        pending queue
                v. Budget_Value[x][y] =Allocation[x][y] +
        Request[x][y]
Else
B.          If          (Request[x][y]<=Need[x][y]          &&
Request[x][y]<=Available[x][y])
        I. Allocate (Allocation, Request, Available, 0)
        II. Request[x][y] ={0}

Else
        I. Allocate (Allocation, Need, Available, Reserve)
        II. Request[x][y] ={0}
5. When a Process completes and releases resources.
End
Allocate (Allocation, To_Allocate, Available, Reserve)
1. For (y=1 to m)
        a. Available[y] =Available[y] + Reserve[y]
        b. Available[y] =Available[y] – To_Allocate[x][y]
        c.  Allocation[x][y]  =  Allocation[x][y]  +
To_Allocate[x][y]
        d. Need[x][y] = Budget[x][y] – Allocation[x][y]
        e. Reserve[y] = min (Available[y], Threshold[y])
        f. Available[y] =Available[y]-Reserve[y]
End
```

4. Resource Search Engine (RSE)

This is the core part of the proposed algorithm. The primary function of the resource search engine is to match the user requirements of the submitted job with the resource capabilities. The resource analyst performs the preliminary selection of resources using the static parameter viz. resource type and forwards the selected resource's capabilities and the submitted job with user requirements to the resource search engine. The functions of the resource search engine are,

- Filter resources using Bloom filer method
- Static parameters matching using assignment method

- Dynamic resource matching using correlation method
- Rank_list calculation
- Behavioral parameters (Hit rate and failure rate) updation.

1. The resources capabilities selected in the preliminary selection and the job with user requirements is forwarded to the Resource Search Engine (RSE) from resource analyzer, local grid information service extracts positions from them. On this basis, positions in counting Bloom filter themselves; perform the trend of resources inquiry as follows: if counter value is 0 at least in one position in Bloom filter, requested resources do not exist in local grid. If counter values of none of these positions in Bloom filter is 0, then string codes pertaining to positions 1st to K-1th are created as stated previously; and the list connected to position Kth in Bloom filter is searched according to string codes. If targeted member is found on the list, then the list of identifiers of similar resources registered in this member is sent along with task's identifier to user. If the list is empty and/or is not but does not contain targeted member, then false positive are detected, indicating that no requested resource exists in local grid.

```
Receive (index , id) from user ;
bloom_index= extract(index); j=0; str=null;
(user_id , job_id) = extract(id);
for i=0 to k-1 do
        j=bloom_index[i];
        if bloomfilter[j].counter==0 then
                send (index , id, RegionalGIS_id) to GIS;
                return;
        else if i<k-1 then str=str+'j';
        end if
end for
LocalResourcelist=search( bloomfilter[j].list , str);
If LocalResourcelist ==null then
        Preliminary selection of resources using the static
parameter by assignment method;
return;
else
send (LocalResourcelist, job_id) to user;
```

2. The resource search engine uses the assignment method to match the job's static parameter values with the resource's static parameter values. The static parameters are Resource name, Resource location, Resource CPU type, and Resource OS and Memory capacity. Assignment method is used for the optimal assignment of a static parameter to a resource to ensure maximum efficiency. For each submitted job, the resource search engine computes a relative parameter match value (C_{isk}) by comparing the job's static parameters and the preliminarily selected resource's static parameters

$$C_{isk} = \begin{cases} 1 & \text{if } X_{ik} - R_{sk} \\ X_{ik} - R_{sk} & \text{otherwise} \end{cases}$$

X_{ik} Represents job parameters

R_{sk} Represents resource parameters

3. The resource search engine uses the correlation method to match the job's dynamic and behavioral parameter values with the resource's dynamic and behavioral parameter values. Correlation method is used to find the correlation between the dynamic and behavioral parameters of the submitted job and the resource. The dynamic parameters are Hard disk availability, Economic state, Dedication rate, Resource budget and Deadline. The behavioral parameters are Success rate, Hit rate, Failure rate and Recovery time.

The correlation of any resource R_s with the job X_i of the i-th user isgiven in below equation.

$$R_s = \frac{k.\sum_{k=1}^{n} X_{ik}R_{sk} - \sum_{k=1}^{n} X_{ik} . \sum_{k=1}^{n} X_{ik}R_{sk}}{\sqrt{(k.\sum_{k=1}^{n}(X_{ik})^2 - (\sum_{k=1}^{n} X_{ik})^2) * (k.\sum_{k=1}^{n}(R_{sk})^2 - (\sum_{k=1}^{n} R_{sk})^2)}}$$

Where n is number of dynamic and behavioral parameters. The resources are ranked according to the correlated value of the resource parameter with the job parameter. The resource whose correlated value is nearly equal to one is given the highest rank.

4. The rank values are obtained for each dimension viz. static, dynamic and behavioral using assignment method and correlation method. The resource search engine finally calculates the rank_list for the submitted job using measures of dispersion. In case of unnormalized ranks, priority is given to the highly ranked resource in the first dimension since the first-dimension matching considers the static weighted parameters. The three-dimensional Matchmaking engine forwards the rank_list and the submitted job to the resource handler for allocation. The rank_list is also stored in the resource repository table of the resource analyst.

5. The resource search engine finds for each resource, the number of occurrences of the resource in the rank_list and the number of occurrences of a resource in the three-dimensional matchmaking engine and calculates the hit rate for every $\Delta t1$ time and updates the calculated hit rate at every Δt time with the grid knowledge unit by sending the hit_rate_update message. The failure rate of a resource is updated in the grid knowledge unit by the resource search engine. The failure rate of a resource is updated whenever the resource is rejected in the rank_list the resource occurs in the resource search engine and the resource search engine updates the failure rate of a resource in the grid knowledge unit by sending the failure_rate_update message to the grid knowledge unit.

5. Resource Monitor

The resource monitor monitors the resources in the grid site through the resource handler. It contains two components. They are Resource analyst and Grid knowledge unit.

a). Resource analyst

Resource analyst is the central part of the matchmaking architecture. The functions of the resource analyst are Resource repository table maintenance and Preliminary resource selection.

Resource repository table maintenance: Every service provider registers the static and dynamic parameter values of its resources in the resource repository table through resource handler. The resource analyst stores the resources' static, dynamic and behavioral parameter values along with the resource_id and the serviceprovider_id in the resource repository table. Initially the resource search engine sets the values of the behavioral parameters like success rate, Hit rate, Failure rate of each resource to 1.0 , and the Recovery time to 0 in the resource repository table. The service provider updates the dynamic parameters of the registered resources at every Δt time in the resource repository table. The knowledge unit updates the behavioral parameters of the registered resources at every Δt time in the resource repository table.

Preliminary resource selection: The resource analyst performs the preliminary resource selection by considering the static parameter. When the user submits new job, the resource analyst performs the preliminary selection of resources by considering the first static parameter. The resource analyst selects all the resources of that particular resource type. The resource analyst then forwards the three-dimensional parameter values of the selected resources and the submitted job along with the user requirements to the resource search engine for rank_list calculation.

b). Grid Knowledge Unit

The grid knowledge unit receives the information about the history of behavior of resources in the resource pool from the resource search engine and the resource handler. It receives the values of resources for success rate calculation and recovery time profiles of resources for recovery time calculation from the resource handler. It receives the hit rate of a resource at every $\Delta t1$ time and values of resources for failure rate calculation from the resource search engine. The grid knowledge unit predicts the success rate, hit rate, failure rate and recovery time for each resource for the next matchmaking cycle based upon its previous behavior with the jobs. It also updates the predicted values of success rate, hit rate, failure rate and recovery time for each resource in the resource repository table of resource analyst at every Δt time interval. It maintains the behavior parameter table and stores the predicted success rate, Hit rate, Failure rate and Recovery time of the resources in it.

3. EXPERIMENTAL RESULTS AND DISCUSSIONS

The proposed algorithm is simulated in GridSim 5.2 and following simulation parameters (given by Table 1) are used:

Table 1: Simulation Parameters

Simulation runs	50
No. of Resources	10
No. of Users	10 – 50
No. of Jobs	1000– 5000
Processing power of Resources (In MIPS)	200

Execution Time

The effect on execution time of jobs by varying the number of users from 10 to 50 and fixing the number of resources at 10 for the proposed algorithm (ADRR) and other algorithms is represented in Figure 2. The graph shows that the resource reservation time and execution time of jobs under ADRR is less as the incoming job is assigned to the resource with minimal queue length. So, ADRR reveals better performance than RSPB and TARR.

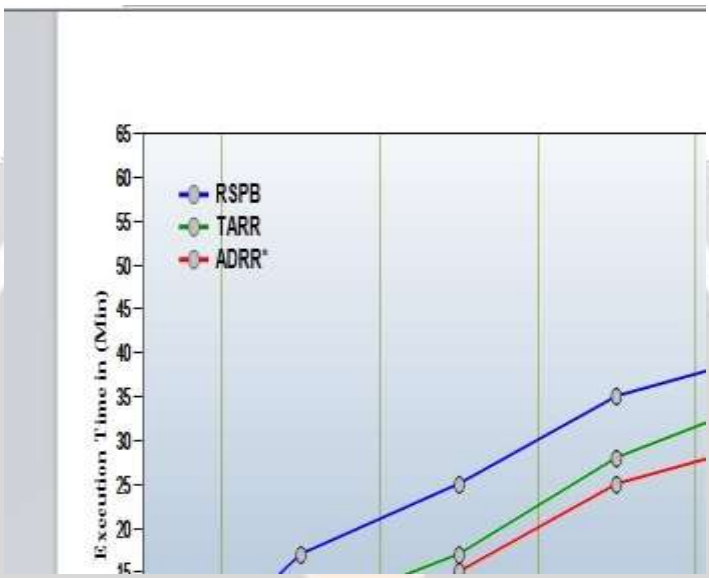

Figure 2: Comparison using execution time

Makespan analysis: The makespan values of the best solutions throughout the optimization run were recorded and the averages were calculated from the different trials. In a grid environment, the main emphasis was to allocate the job and resource as fast as possible by using appropriate resource reservation algorithm. The following Figure 3 shows the makespan time of jobs run in each node of grid environment. These developments indicate the fastest of each node by using proposed resource reservation algorithm ADRR. This shows that the proposed algorithm makespan is minimized when compared to other algorithms.

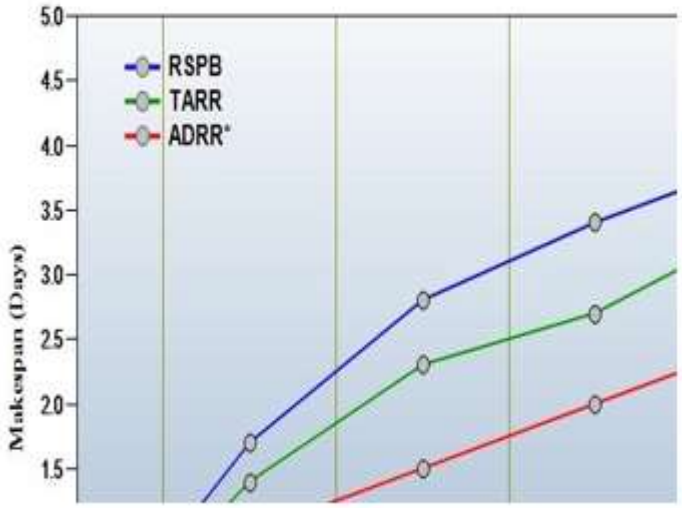

Figure 3: Makespan analysis

Resource utilization: The resource utilization is a Quality of Service (QOS) criterion. Below Equation gives the average utilization of resources for a schedule S. The value of average_utilization has to be maximized.

$$\text{average}_{\text{utilization}} = \frac{\sum_{\{i \in \text{Machines}\}} \text{completion}[i]}{\text{makespan} \times \text{nb\_machines}}$$

Where nb_machines denote the number of resources, this Equation gives the completion time of a machine m. It is evident that the average utilization of resources for all algorithms is ranked as: 1) ADRR 2) TARR and 3) RSPB. The overall performance of ADRR is good while considering the average resource utilization.

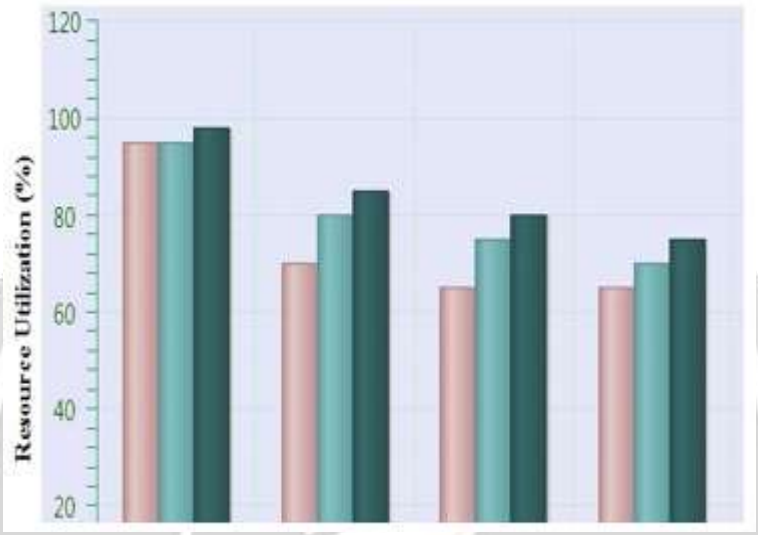

Figure 4: Graph between percentage of utilization and percentage of jobs reserved

No of Rejections: Figure 5 depicts an analysis between number of rejections and the percentage of jobs seeks reservation, the number of rejections goes up. As seen from the figure, the ADRR gives improved results over TARR and RSPB. This is due to the reason that the shortest jobs of higher priority users are entertained dynamically at the run time.

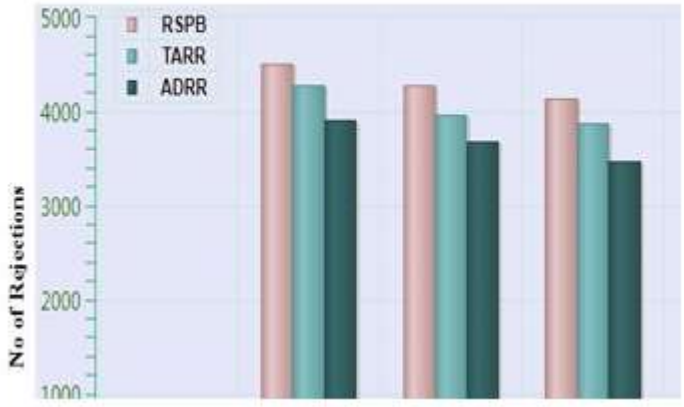

Figure 5: Graph between total no of rejections and percentage of jobs reserved

4. CONCLUSION

Resource reservations mechanisms and scheduling efficiency are keys to the success of computational Grids, and especially to Grid capability to deliver commercial applications a guaranteed and personalized Quality of

Service (QoS). This paper introduces a novel way of incorporating QoS constraints and priority into an advance reservation scheduling algorithm for in Grid computing systems. The paper compares the performance of the proposed algorithm (ADRR) with an existing advanced reservation algorithm, namely the TARR, RSPB and analyzes the simulation results. The evaluation results indicated an increase in the efficiency of resource utilization, a decrease in the process execution delays and also a decrease in the request rejection rate for the proposed method compared with other two methods.

We are currently investigating in the design and implementation of a real grid application which can take benefit from our dynamic resource allocation mechanisms. The project consists of delivering learning objects from a grid-computing environment for the purpose of building e-learning applications.

5. REFERENCES

1. U. Rusydi, A. Arun and C. R. Rao, "Data Structure for Advance Planning and Reservation in a Grid System," International Journal of Computer Applications (IJCA), vol. NCRTC, no. 1, pp. 33-37, 2012.
2. Wu, Zhiang & Cao, Jie & Wang, Youquan. (2011). Dynamic Advance Reservation for Grid System Using Resource Pools. 6985. 123-134. 10.1007/978-3-642-24403-2_10.
3. Mai, Yi-Ting & Hu, Chih-Chung. (2021). Design of Dynamic Resource Allocation Scheme for Real-Time and Non-Real-Time Traffics in The Advanced Mobile Communications Network. 10.21203/rs.3.rs-699648/v1.
4. Deadlock-Free Scheduler, Hindawi Mathematical Problems in Engineering, Volume 2018, Article ID 1438792, 18 pages.
5. W. Smith, I. Foster, and V. Taylor, “Scheduling with Advanced Reservations,” International Parallel and Distributed Processing Symposium (IPDPS ’00), May 2000.
6. S. Nirmala Devi, A. Pethalakshmi,” TARR: Time Slice based Advance Resource Reservation in Grid Computing Environments”, International Journal of Computational Intelligence and Informatics, Vol. 6: No. 1, June 2016
7. Sulistio and R. Buyya , "A Grid simulation infrastructure supporting advance reservation," in 16th International Conference on Parallel and Distributed Computing and Systems, pp. 1–7. ACTA Press, Calgary, 2004.
8. Ismail, Leila. (2007). Dynamic Resource Allocation Mechanisms for Grid Computing Environment. 1-5. 10.1109/TRIDENTCOM.2007.4444737
9. Goutam, Shubhakankshi & Yadav, Arun. (2015). Preemptable Priority Based Dynamic Resource Allocation in Cloud Computing with Fault Tolerance. 10.1109/ICCN.2015.54.