International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 4 | ISSN : 2456-3307

Comparative Analysis of Various Advance Resource Reservation Algorithms In Grid Computing Environments

[1]S. Sivakumar, [2]Dr. D. Maruthanayagam
[1]Research Scholar, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu. India
[2] Head/Professor, PG and Research Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India

ABSTRACT

Grid computing is an emerging paradigm for next generation distributed computing. The Grid is a highly dynamic environment with servers coming on-line, going off-line, and with continuously varying demand from the clients. Advance Reservation is a process of requesting resources for use at a specific time in the future. Common resources whose usage can be reserved or requested are CPUs, memory, disk space and network bandwidth. Reservation for a grid resource solves the above problem by allowing users to gain concurrent access to adequate resources for applications to be executed. Reservation also guarantees the availability of resources to users and applications at the required times. In this paper experiments analyzed and compared Average Waiting Time and Resource Idle Time of four different Reservation techniques such as RSPB, ORR, TARR and DRR.

Keywords: Grid Computing, Resource, Reservation, RSPB, ORR, TARR and DRR

I. INTRODUCTION

Grid computing uses middleware to coordinate dissimilar IT resources across a network, allowing them to function as a virtual whole. The goal of a computing grid, like that of the electrical grid, is to provide users with access to the resources they need, when they need them. Grids address two distinct but related goals: providing remote access to IT assets, and aggregating processing power. The most obvious resource included in a grid is a processor, but grids also include sensors, data-storage systems, applications, and other resources [1].

Grid resource management (administration) plays a substantial role while enabling the sharing and coordinating of resources in grid computing environments. Resource reservation is an important element of the grid resource management. An advance resource reservation is a scheduling object which reserves a group of resources for a particular timeframe for access only by a specified entity or group of entities. Requests of advance reservation with fixed parameters (Start time, End time and resource capability) may be rejected due to instantaneous peaks of resource utilization. Gaps between these peaks are too narrow for additional requests to fit in. As a result, the call acceptance rate of reservation would decrease dramatically, and the performance of resource may be reduced. In fact, many resource reservations for grid applications do not need fixed parameters. In flexible advance reservation, parameters can be modified according to resource status in order to fill the gaps of resources. Particular admission control algorithm for this new type of reservation is also provided. Simulation shows that it can improve performance of resource reservation in terms of both call acceptance rate and resource utilization [2].

The advance resource reservation technique makes it possible to obtain a guaranteed start time for a job, giving several advantages. It makes it possible to meet deadlines for time-critical jobs and to coordinate the job with other activities. The reservation protocol supports two operations: requesting a reservation and releasing a reservation. The reservation request contains the start time and requested duration of the reservation and the required number of CPUs. Upon receiving a reservation request from the broker, the server on the resource

authorizes the requestor. After authorizing the user, the job management plug-in of the server invokes a script to request a reservation from the local scheduler. If the scheduler accepts the request and creates the reservation, the server returns a unique identifier and the start time of the reservation to the broker. If no reservation can be created, a message indicating failure is returned [3]. The server saves the reservation identifier and a copy of the user's proxy for every successful reservation, enabling subsequent authorization of the user who made the reservation. To release a reservation, the broker uploads a release message containing the reservation identifier and the server confirms that the reservation is released. Within larger combined distributed systems, the efficient description of resources plays a significant role in the performance of the computing system. However, distributed resource allocations can also result in lower resource utilization owing to the delay involved in arbitration, the delay in taking up the agreed resources, and the tentative allocation of resources during the arbitration process. Resource oversubscription allows for better utilization of resources in distributed systems, however, this must be done in a controlled way to ensure that the resulting allocations can be fulfilled.

II. ADVANCE RESERVATION ALGORITHMS

A. Reservation Scheduler With Priorities And Benefit Functions (RSPB)

The [4] proposes and evaluates several algorithms for supporting advance reservations in supercomputer scheduling systems. These algorithms improve traditional scheduling algorithms by unifying scheduling traditional tasks from job queues with the reservation requests. These advance reservations allow users to request multiple resources simultaneously from scheduling systems at specific times. However, [4] allocates the "time slots" exclusively, i.e., the resources are not reserved in a shared fashion by multiple clients for the same duration. The applications are assumed to operate on a "best effort" basis and the reservation requests are assumed to have different priority than the applications. These differences in priorities are considered while the reservations and applications are scheduled by the system. A Reservation Scheduler with Priorities and Benefit Functions (RSPB) algorithm schedules reservations while considering the relative priorities of the various reservation requests. In RSPB, each reservation request has an associated benefit function that quantifies the "profit" for the client accrued by the client by securing the resource at the requested level. When the client is willing to negotiate for lower service levels, it could indicate this by providing a benefit function that shows a reduced but positive benefit for lower resource levels. This facility provided by the benefit functions removes the need for negotiations when there is a resource scarcity. The RSPB can be implemented on top of a CPU scheduler such as the DSRT [5] or a QoS enhanced operating system kernel such as QLinux [5].

Reservation Scheduling with Priorities and Benefit Functions: The algorithm is based on the following underlying assumptions. Once a request is granted reservation, a contract for the reservation is signed between the application and the system. The reservation scheduler won't examine the same request more than once except when a QoS violation occurred. This situation should be handled by a higher level QoS broker that engages in renegotiation to establish another reservation or a continuation of the current reservation. Based on the operating policies, the reservation scheduler may find another reservation or the application may operate under best-effort conditions. In this algorithm, each reservation request involves a single resource, i.e., no co-reservation of resources is considered here [6]. Figure 1 shows the outline of the dynamic reservation scheduler. In this scheduler, dynamically arriving requests are collected for a predefined time interval to form a meta-request.

```
t=t0 ;; scheduler start time
Δt ;; inter-schedule time
while (true)
t = t + Δt;
while (current time < t)
get current request R;
add R to Rmeta
if (requested start time of R < t)
t = current time;
endif
endwhile
schedule Rmeta(Rmeta)
endwhile
```

Figure 1: Outline of dynamic reservation scheduler.

B. Optimal Resource Reservation

In the ORR (Optimal Resource Reservation) approach the best fetch strategy is considered. During reservation, if the slots requested are empty then they are reserved. The conflict occurs only when the slots are not available. Normally, the reservation denial is done in FCFS approach. In TARR, the free slots are considered and the reservation slots are provided rather than as a single slot. For this purpose, in addition to the proposed start time and finish time, the defer time (DT) is also considered. The defer time is time until which the job can be completed or considered for reservation. In TARR approach, whenever a free slot is available it is allotted as such. TARR necessitates more context switching i.e., whenever a small chunk of time slice is available, then that is reserved which requires more process suspension and resumption. The entity resource can be reserved for a period of time. After the elapse of time the current process in execution need to undergo process switching. The current status of the PCB (Process Control Block) is stored. And the detail on process to be resumed is retrieved. The new process possesses the resource. The state transition or process switching causes additional overhead. Hence in this ORR the slicequeue and select slicequeue are maintained [6].

Slice Queue: The reservation list of a resource maintains the details on resource and also the job which has reserved the respective resource. Figure 2 depicts the resource list with reservation and empty slots.

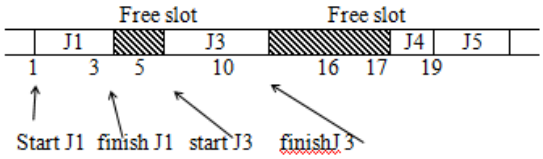

Figure 2: Reservation List maintained by resources.

To find the empty slice initially the reservation list is taken as input. The reservation list consists of the job id which has reserved the resource with its period of reservation i.e., its start time and finish time. The free slots are identified. The difference between the finish time of the previous reservation and the start time of the current reservation is computed. If there exist any time then that free slice is inserted into the slice queue.

Thus the slice queue maintains the free time slices. The Slice queue consists of slice id, the slice start time (SST) and the slice finish time (SFT) of a free slice. The following algorithm inserts the free time slices into the queue.

```
Algorithm
SliceQueue_Insert(reserve_list)
Begin
      front = 0, rear = 0
      for i = 0 to list_size - 1
                  timeslice =
            starti+1 - finishi
                  if timeslice > 0
            then
                  sst =finishi
                  sft = starti+1
                  slicequeue[rear]=
            {sid,SST,SFT}
                  end if
            end for
end
```

While considering the Figure 2, the free slots from 3 to 5 and from 10 to 16 are placed in the slice queue. The Slice queue maintains all the free time slots. The time slots are taken in first free slot available approach as in TARR. The slice queue for Figure 2 is shown in Figure 3. This leads to more context switching which leads to increase in overhead.

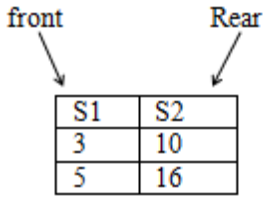

Figure 3: Slice Queue.

Select Slice Queue: After generating the slice queue, the slice is searched for reservation. The slice queue consists of all the free slices. But not all the free slices can be used. Hence, the select slice queue algorithm finds the slice which is more relevant to the required slot. The algorithm considers the free slot and the Defer Time (DT) to fetch the slots [7].

```
Algorithm
selectslicequeue(slicequeue)
begin
      if slicequeue is empty then
            return no_slice
      else
            for    i    =    0    to
      size(slicequeue)
```

```
            if  (  slicequeue.sft  <
        DT) then
            insert
        selectslicequeue(sid,sliceque
        ue.sst,
            slicequeue.sft)
            end if
            end for
        end if
    end
```

The selectslicequeue algorithm selects the slots only when the time slice is within the defer time.

ORR Algorithm: The ORR algorithm finds the optimal time slice from the selectslicequeue. For this the algorithm takes in the job id of the process and the selectslicequeue as input. And it returns the optimal reservation of resource which is explained in following algorithm. The selectslicequeue maintains the time slices that can be reserved. The size function in ORR returns the number of elements in the particular queue. Initially the time required for the job is estimated. It is computed by calculating the difference between the finish time of that job and the start time of the job. Then they found value is set to zero. It is used as a flag variable. If it is left as zero then it means that the reservation is not made possible. Only when it is turned to one the slot is available. The size of the free slice is computed from the selectslicequeue. Since the selectslicequeue maintains the slice start time and slice finish time the difference between the two is computed and maintained in slot variable. In this algorithm if the first slice itself is in required size then it is allocated and it is the optimized allotment as it doesn't require the context switching. Otherwise the slot is allotted and the required time is reduced by the slot value. So that the required time value contains the required time slot after partial reservation. The found value indicates the number of time slices considered for reservation for the particular job id. The found value is compared with the threshold value. If they found is greater than the threshold value then the randomize function is called on the selectslicequeue. This function shuffles the time slices and the time slices are selected. When the numbers of time slices are below the threshold value then the selected slices are assigned for the job [8].

```
Algorithm ORR(JID, SelectSliceQueue)
      begin
            requiredtime = FT – ST
            found = 0
      for i = 1 to size(selectslicequeue)
            slot = selectslicequeue.sft -
      selectslicequeue.sst
      if (slot >= requiredtime && found =
      0) then
            allot(sid,
      selectslicequeue.sst,
      selectslicequeue.sft)
      return
      else
            requiredtime = requiredtime –
      slot
            allot(sid,selectslicequeue.ss
      t,selectslicequeue.sft)
            found = found + 1
      end if
      end for
      if found > T then
            randomize(selectslicequeue)
            ORR(JID, selectslicequeue)
      else
            assignslice(found)
      return
      end if
      end
```

C. Time-Slice based Advance Resource Reservation (TARR)

A new reservation scheme called Time-Slice based Advance Resource Reservation (TARR) is proposed. In this scheme, the reservation is done when the resource is free. If the resource is already reserved during that timeslot then the free the time slices can be used for the reservation. This splits the resource utilization period, i.e., whenever a free time-slice is available the resource is reserved for that duration and the remaining is deferred over a period of time where the free time slice is available. In all the existing approaches only when the resource is available in the specified start time and finish time the reservation is done. Because of this even if the resource is available for short duration than expected then the resource is kept unused. The TARR approach tries to remove this drawback by allowing the usage of time-slices between the existing reservations The Start, finish and defer time for using the resources are given by the user while submitting a job for execution. The start time and finish time are the expected start and finish time for using the resources. Sometimes it is not possible to reserve the resource at the stipulated start and finish time. In such case the reservation can be done until the defer time. Defer time is the time until which the reservation can be postponed [9].

TARR Algorithm

```
Algorithm TARR
      if List is empty then
            no conflict found. Hence
      accept the new reservation.
      else
      for i = 0 to List-Size -1 do
            Put i into templist if one of
      the properties are true
            Startnew ≤ starti&&finishnew≥
      finishi || Startnew ≤ finishi||
      finishnew≤ finishi
      end for

      if templist is empty then
            no conflict found. Hence
      accept this reservation.
      else
            requiredtime = finishnew-
      startnew
            balancereserve = requiredtime
      for i = 1 to n
            timeslice = starti+1 -finishi
      if timeslice> 0 then
            Put into
      slicequeue_insert(id,sst,sft)
      end if
      end for
      while (slicequeue&&balancereserve)
      if sst ≥ Startnew then
            assign new reserve
            currentreserve=sft – sst
            Balancereserve = requiredtime
      –currentreserve
      end if
      end while
      end if
      end if
```

Dynamic Resource Reservation (DRR)

There were various resource reservation schemes available as FCFS (First Come First Served), reservation based on negotiation, TARR (Time Slice based Advanced Resource Reservation), ORR (Optimized Resource Reservation). In all these methods, the reservations done are not checked for utilization at any period of time. But there are chances of unutilized reservation due to network failure, termination of parent process, termination of current process etc., In the DRR (Dynamic Resource Reservation) scheme, the resource is checked for availability. If the resources are available then it is allotted. If the resources are already reserved then it is checked for the reservation requirement at that particular point of time. If the reservation requirement is not required then the current reserved slot is provided to the current process [10].

Slice Queue: The free time slots are maintained in the slicequeue for every resource upto a specified time. Normally, the slicequeue used to have the start time and finish time. Whenever the resource request is made then the availability is checked in this slicequeue. If the time slot is available then it is provided. If the time slot is not available then the DRR algorithm can be used to get the required slice from the slice queue depending upon the possibility. The algorithm DRR shows the proposed reservation scheme. The algorithm takes in the process id, start time (ST), finish time (FT) and defer time (DT) as input.

```
Algorithm DRR (Process_id, ST, FT, DT)
      begin
      If the list is empty then
            No conflict found. Hence
      accept the reservation.
      else
            get_jobid(timeslot)
      if isalive(jobid)
      if isrequire(timeslot)
            return()
      else
            allot(current_jobid,
      timeslot)
      end if
      else
            allot(current_jobid,
      timeslot)
      end if
      end if
      end
```

If the reservation list is empty then it means that the resource is free during the required time slot. Hence the reservation can be done. When reservation is already available during the time slot, then the job can be checked for the availability. The isalive function returns whether the reserved job id is still alive or not. If the reserved job is not alive then the slot is available for the current reservation hence that can be allotted for the current reservation. If the required time slot is reserved then the isrequire function is called which determines whether the reservation made is further required or not. If it is not required then the time slot can be used for the current reservation.

III. PERFORMANCE METRICS

There were various performance metrics that are considered to evaluate the performance of the Reservation algorithms. Here, The average waiting time and Resource Idle Time taken as metrics for evaluation [10][11][12],

A) Average Waiting Time (AWT): The waiting time (WT) of the reservations are computed. Sometimes the resources are not available at the time of reservation requirement. But the resources can be reserved within the deferred time. In that case, the difference between the expected start time and the actual reserved start time is the waiting time.

$$\text{Waiting Time (WT)} = \text{Start}_{\text{reserve}} - \text{Start}_{\text{new}}$$

The Total Waiting Time (TWT) is computed as the sum of all the waiting time at a specific point of time.

$$\text{Total Waiting Time (TWT)} = \text{WT} \sum_{i=1}^{size}$$

Where size refers to the length of the reservation list at a specific point of time. Then ,

$$\text{AWT} = \text{TWT} / \text{No of Reservations}$$

B) Resource Idle Time (RIT): The resources may be idle even when the reservation request available. This happens when the idle time does not fit into the allocation policy. Thus TARR provides a better allocation policy, as even the time-slices are used for reservation rather than allocating the entire request as a single unit. The RIT is computed by applying the below formula

$$\text{RIT} = \text{Finish}_{\text{previous}} - \text{Start}_{\text{current}}$$

When there exists a reservation request with a conflict. The total resource idle time is computed by the following equation

$$\text{Total RIT} = \sum_{i=1}^{size} \text{RIT}$$

IV. COMPARATIVE ANALYSIS

To compare the Existing algorithms as RSPB, 0RR, TARR and DRR in Two Grid Environments are considered. The Expected start time (EST),Expected finish time (FT) and Rescheduling time (RT) for 5 jobs fewer than 2 Grid Environment are listed in table 1 and table 2. The table 1 depicts the reservation of Grid Environment 1 based on the four different models with its Average waiting time and Grid Resource idle time. The table 2 depicts the reservation of Grid Environment 2 based on the four different models with its Average waiting time and Grid Resource idle time.

Table 1: Reservation of Grid Environment 1

GRID ENVIRONMENT 1											
Job Allotment	EXPECTED STARTING TIME	EXPECTED FINISHING TIME	Average Waiting Time(AWT)				Grid Resource Idle Time(RIT)				RESCHEDULE TIME
			RSPB	ORR	TARR	DRR	RSPB	ORR	TARR	DRR	
Job1	0.25	7.56	5.41	5.24	4.8	3.69	5.16	4.99	4.55	3.44	7.81
Job2	0.59	8.52	6.71	6.54	6.1	4.99	6.12	5.95	5.51	4.4	9.11
Job3	1.25	9.57	8.42	8.25	7.81	6.7	7.17	7.0	6.56	5.45	10.82
Job4	1.55	10.24	9.39	9.22	8.78	7.67	7.84	7.67	7.23	6.12	11.79
Job5	2.08	10.59	10.27	10.1	9.66	8.55	8.19	8.02	7.58	6.47	12.67

Table 2: Reservation of Grid Environment 2

GRID ENVIRONMENT 2											
Job Allotment	EXPECTED STARTING TIME	EXPECTED FINISHING TIME	Average Waiting Time(AWT)				Grid Resource Idle Time(RIT)				RESCHEDULE TIME
			RSPB	ORR	TARR	DRR	RSPB	ORR	TARR	DRR	
Job1	0.13	6.24	3.97	3.8	3.36	2.25	3.84	3.67	3.23	2.12	6.37
Job2	0.19	7.03	4.82	4.65	4.21	3.1	4.63	4.46	4.02	2.91	7.22

Job3	0.59	7.56	5.75	5.58	5.14	4.03	5.16	4.99	4.55	3.44	8.15
Job4	1.26	8.14	7	6.83	6.39	5.28	5.74	5.57	5.13	4.02	9.4
Job5	1.58	9.07	8.25	8.08	7.64	6.53	6.67	6.5	6.06	4.95	10.65

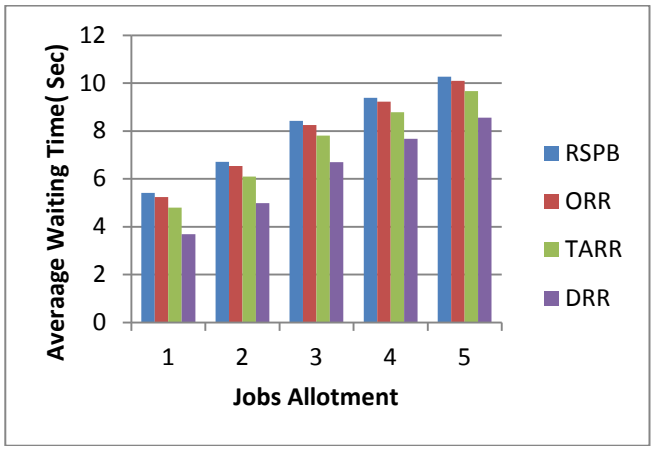

Figure 4: Comparison of Reservation process Average Waiting Time (AWT) in Grid Environment1

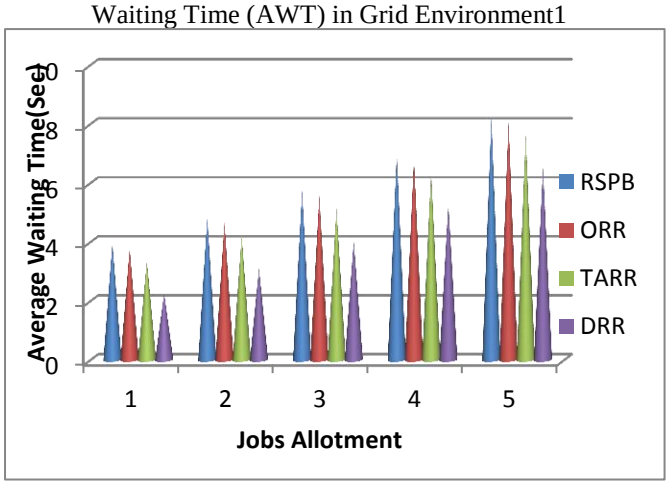

Figure 5: Comparison of Reservation process Average Waiting Time (AWT) in Grid Environment2

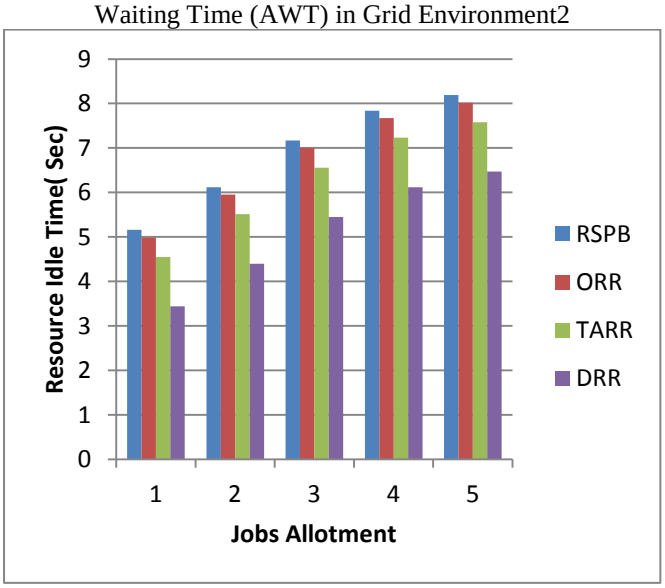

Figure 6: Comparison of Reservation process Resource Idle Time (RIT) in Grid Environment1

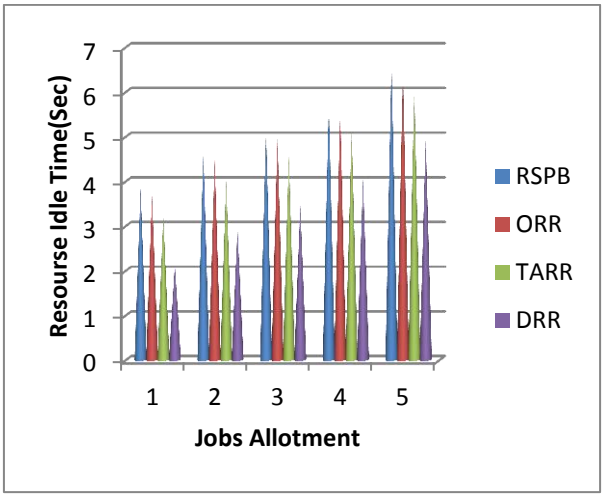

Figure 7: Comparison of Reservation process Resource Idle Time (RIT) in Grid Environment2

V. CONCLUSION

Resource reservation provides a way through which the resources are made readily available for the entire execution of the process. This reduces the waiting time which in turn reduces the execution time also. Two parameters such as average waiting time and resource idle time are considered for evaluation in all the existing reservation techniques. Finally, the existing reservation algorithm DRR provides better result when compared with RSPB, ORR, and TARR reservation algorithms.

VI. REFERENCES

[1]. A. W. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. IEEE transactions on Parallel and Distributed Systems, 12(6):529{543, 2001}.

[2]. J. MacLaren, editor. Advance Reservations: State of the Art (draft). GWD-I, Global Grid Forum (GGF), June 2003. http://www.ggf.org. 69

[3]. W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, May 1{5, 2000}.

[4]. A. Schill, F. Breiter, and S. Kuhn. Design and evaluation of an advance reservation protocol on top of rsvp. In Proceedings of the IFIP TC6/WG6.2 4th International Conference on roadband Communications (BC'98), Stuttgart, Germany, April 1998.

[5]. J. MacLaren (Ed.), Advance Reservations: State of the Art (draft), GWD-I, Global Grid Forum (GGF), June 2003 http://www.ggf.org

[6]. A. Roy and V. Sander, Advance Reservation API, GFD-E.5, Scheduling Working Group, Global Grid Forum (GGF), May 2002.

[7]. H. Chu and K. Nahrstedt, "A Soft Real Time Scheduling Server in UNIX Operating System,"*European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS '97),* Sep. 1997.

[8]. P. Goyal, X. Guo, and H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," *2nd Symposium on Operating System Design and Implementation (OSDI '96),* Oct. 1996, pp. 107-122.

[9]. W. Smith, I. Foster, and V. Taylor, "Scheduling with Advanced Reservations," *International Parallel and Distributed Processing Symposium (IPDPS '00),*May 2000.

[10]. **S.Nirmala Devi and A.Pethalakshmi** ORR: Optimal Resource Reservation In Grid Computing Environments, Indian Journal of Science and Technology, *Vol 9(48), DOI: 10.17485/ijst/2016/v9i48/102300, December 2016*

[11]. **S.Nirmala Devi and A.Pethalakshmi, "**DRR: Dynamic Resource Reservation in Grid Computing", *International Journal of Computer Applications (0975 – 8887) Volume 159 – No 6, February 2017* 27

[12]. **S.Nirmala Devi and 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

VII. AUTHORS PROFILE

S. Sivakumar received his M.Phil Degree from Alagappa University, Karaikudi in the year 2005. He has received his M.C.A Degree from Periyar University, Salem in the year 2001. He is working as Assistant Professor, Department of Computer Science, PGP College of Arts & Science, Namakkal, Tamilnadu, India. He is pursuing his Ph.D Degree at at Sri Vijay Vidyalaya College of Arts & Science (Affiliated Periyar University),Dharmapuri, Tamilnadu, India. His areas of interest include Grid Computing and Data Mining.

Dr.D.Maruthanayagam received his Ph.D Degree from Manonmanium Sundaranar University, Tirunelveli in the year 2014. He has received his M.Phil, Degree from Bharathidasan University, Trichy in the year 2005. He has received his M.C.A Degree from Madras University, Chennai in the year 2000. He is working as HOD Cum Professor, PG & Research Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India. He has 15 years of experience in academic field. He has published 4 books, 23 International Journal papers and 28 papers in National and International Conferences so far. His areas of interest include Computer Networks, Grid Computing, Cloud Computing and Mobile Computing.