Data Structure for Advance Planning and Reservation in Grid System

Rusydi Umar
Dept. of Information and Computer Sciences, Univ. of Hyderabad, Hyderabad, India.
Dept. of Informatics Engineering, Ahmad Dahlan Univ., Yogyakarta, Indonesia

Arun Agarwal
Department of Information and Computer Sciences, University of Hyderabad.
Prof CR Rao Road, Gachibowli, Hyderabad, 500046 India

CR Rao
Department of Information and Computer Sciences, University of Hyderabad.
Prof CR Rao Road, Gachibowli, Hyderabad, 500046 India

ABSTRACT
In Grid system, we need an advance reservation to ensure that specified resources are available for applications in a particular time in the future. The impact of advance reservations is decreasing resource utilization due to fragmentations. To mitigate this problem in our previous work we have proposed a novel advance reservation scheduling namely First Come First Serve – Ejecting base Dynamic Scheduling (FCFS-EDS) with advance planning. In order to implement reliable FCFS – EDS scheduling, it is important to store information in data structures about future allocations and to provide fast access to the available information. This paper proposes a novel data structure used by FCFS – EDS scheduling strategy to increase the throughput in a grid environment.

General Terms
Data Structure, Algorithm.

Keywords
Data Structure, FCFS-EDS, Advance Reservation.

1. INTRODUCTION
Ian Foster et al. [1] defined a grid computing as "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities". Grid computing system has the ability to provide the quality of service (QoS) requirements given by their users. Example of QoS is the availability of resources needed by a user [2]. Grid computing technology is well-suited for Bag-of-Tasks (BoT) applications. In BoT applications each application consists of independent tasks or jobs [3, 4]. In most grid systems, submitted jobs are initially placed into a queue if there are no available resources. Therefore, there is no guarantee as to when these jobs will be executed. This causes problems in time-critical or parallel applications, such as task graph, where jobs may have interdependencies [5].

Advance reservation is a mechanism for requesting a resource for use at a specific time in the future from multiple scheduling systems. Currently, several grid systems are able to provide advance reservation functionalities, such as GARA [6], ICENI [7], Maui Scheduler [8], Portable Batch System PBS Pro [9], DSRT [10, 11], Sun Grid Engine [12], and GridSim [13]. The impact of advance reservations is decreasing resource utilization due to fragmentations [14]. Several strategies have been proposed to mitigate the impact of advance reservation, i.e. to increase resource utilization. We have proposed advance reservation strategy to increase resource utilization namely First Come First Serve – Ejecting base Dynamic Scheduling (FCFS – EDS) with advance planning[15]. In order to implement reliable FCFS – EDS scheduling, it is important to store information in data structure about future allocations and to provide fast access to the available information. This paper proposes a novel data structure used by FCFS – EDS scheduling system.

2. RELATED WORKS
There are several data structures for administering advance reservation. In general there are two types of data structure for administering advance reservation i.e. time slotted data structure and continuous data structure. In time slotted data structure each request is stored in a certain number of consecutive time slots, where in continuous data structure each request defines its own time scale. Time slotted data structure approach has the advantage of restricting the amount of data that must stored, i.e., the memory consumption is bounded and, furthermore, it can be easily implemented [16]. The majority of current implementations in the field of advance reservations support time slotted data structure [17, 8, 18, 19, 20, 21 and 25]. Consequently, the data structure presented here is designed to support slotted time.

A tree-based data structure is commonly used for admission control in network bandwidth reservation [20, 22, and 23]. Brown et al. [21] have proposed Calendar Queue (CalendarQ), as a priority queue for future event set problems in discrete event simulation. Qing Xiong et al. [24] have proposed a linked-list data structure for advance reservation admission control. Sulistio et al. [25] have proposed GarQ (Grid Advance Reservation Queue) for administering advance reservation in grid system. GarQ was partly influenced by Calendar Queue data structure. GarQ has buckets with a fixed δ, which represents the smallest slot duration, as with the Calendar Queue. Sulistio et al. [25] also implemented all data structures aforementioned above in grid system and compare the performance of GarQ and the other data structure. They had a result that GarQ performed better among the other data structure.

3. PROPOSED DATA STRUCTURE
All existing data structure can not be used for our scheduling strategy (FCFS-EDS [15]) for administering advance reservation. Our proposed data structure for administering advance reservation using FCFS-EDS scheduling strategy is influenced by GarQ data structure, because it has better performance among other data structure reported in the literature. The proposed data structure has buckets with a fixed time (our assumption in 5 minutes), which represents the smallest slot duration, as with the Calendar Queue.

Our proposed data structure can be seen in Fig. 1. It is an array of a record that contains of a variable named free that holds the amount of free compute node in the time slot and pointer p to a linked list of nodes. Name of the array is tslot. The index of the array is the number of time slot. Each time slot contains a list of nodes of reservation that starts in that time slot. The node contains this information:

userID : this information is useful for identification the user
jobID : user can submit more than one independent job, jobID can identify them
t_{es} : the earliest start time that the job can be started
t_{ls} : the last start time that the job can be started
t_e : the execution time of the job
next : pointer to a node of reservation

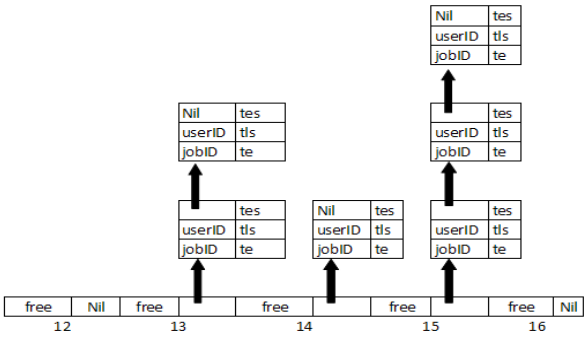

Fig 1: Proposed Data Structure for FCFS-EDS Scheduling Strategy

We define k as the number of nodes (reservation) in each timeslot and t_r as the relax time (the earliest start time – the latest start time) or $t_r = t_{ls} - t_{es}$.

Let us see the following illustration. Wa have a total number of compute node maxCN = 5, and a sequence of incoming reservation is depicted in Table 1. For example the given parameters for userId = 3 in the Table 1 implies the following: "User 3 reserved 3 time slots at time slot 12 up to 14, and cannot be delayed/shifted. (t_{es}=t_{ls}=12, t_e=3)". Data structure for storing all reservation in Table 1. Can be seen in Fig. 2.

FCFS-EDS (First Come First Serve - Ejecting base Dynamic Scheduling) strategy takes an advantage of shifting earlier reservations made (subject to given flexible constraints) to make room for new incoming reservation request.

Definition of variables is explained in lines 4-9. Lines 10–14 initialize the variables.

Table 1. Parameters of reservation request.

userID	jobID	t_{es}	t_{ls}	t_e
1	1	11	11	1
1	2	11	11	1
2	1	11	11	3
3	1	12	12	3
4	1	15	16	1
5	1	15	15	1
5	2	15	15	1
6	1	11	11	2
7	1	13	13	1
8	1	16	16	2
9	1	11	11	2
10	1	15	15	3

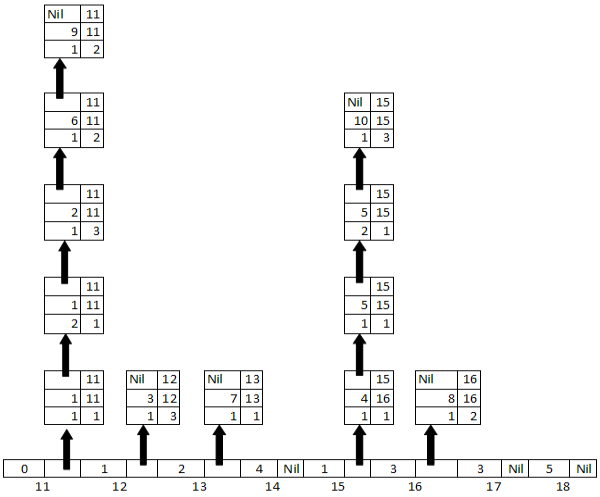

Fig 2: Data Structure for storing reservation request from Table 1.

Let us assume that earlier "n-1" reservation requests made to FCFS-EDS, specified by the following parameters: t_{es}, t_{ls}, t_e, userId, jobId, have been successfully scheduled.

Incoming "n^{th}" reservation request is scheduled based on first fit strategy (iteration of lines 16-25). It tries to search for resources within the given constraints (t_{es} and t_{ls}) and without disturbing plan for previous "n-1" reservation requests that were made.

Let us call this as plan P_{old}. If within the given flexible constraints the search fails to allocate resources the algorithm tries to move around previous "n-1" reservations to accommodate "n^{th}" reservation request (lines 31-44). If the resources are found then the search is declared successful and the algorithm outputs a new plan P_{new} that depicts "n" reservation requests as a logical view. If the search is a failure then the "n-1" reservation request plan has to be restored to its previous state i.e. P_{old}.

The time complexity of FCFS-EDS algorithm is O(n.m) where n is t_r (where t_r is time of relax is equal to $t_r = t_{ls}$-t_{es}) and m is t_e.

```
Algorithm 1: FCFS-EDS
1 Function searchAndAlloc(userId, jobId, tes, tls, te : integer) → boolean
2 //search and allocate job with given tes, tls, te

3  Dictionary :
4  start : integer /*start time of the job*/
5  finish : integer /*finish time of the job*/
6  min : integer  /*min free within interval  start - finish*/
7  t   : integer  /*timeslot of minimum available node between start to finish*/
8  tr  : integer  /*relax time, length between of tes and tls*/
9  relax : integer /*different between start and tes time (start - tes + 1)*/

Algorithm :
```

```
10 tr ← tls – tes
11 succeed ← false
12 start ← tes
13 finish ← tes + te – 1
14 relax ←  start - tes
15
16 while (!succeed and (relax≤tr)) do  /*searching by first fit strategy*/
17    /*searching minimum free node between start to finish*/
18    min,t ← minFreeNode(start, finish)
19      if(min > 0) then
20        allocate(userId, jobId, tes, start, tls, te)
21        succeed ← true
22      else
23        start ← t + 1
24        finish ← start + te - 1
25        relax ← start - tes
26 /*end while, the state is succeed = true or succeed = false (relax > tr)
27
28 start ← tes
29 finish ← tes + te – 1
30 relax ←  start - tes
31 while (!succeed and (relax≤tr)) do
32    /*searching minimum free node between start to finish*/
33    min,t ← minFreeNode(start, finish);
34    if(min > 0) then
35        /*push or schedule the job to data structure using our lemma below and
36          update free node between start to finish*/
37      allocate(userId, jobId, tes, start, tls, te)
38      succeed ← true
39    else
40      /*try to shift a job that start at t time slot*/
41      if(!shiftNode(t)) then //can't be shifted, move start to index+1
42        start ← t + 1
43        finish ← start + te - 1
44        relax ← start - tes
45 /*end while, the state is succeed = true or succeed = false (relax > tr)*/
46 if (!succeed)
47   putBackAllShiftedJob()
48 return succeed
```

Algorithm 2. shows an algorithm for allocating/adding the reservation in the data structure. Here new reservation is put in an ascending order by userId and jobId. Lines 12-19 is adding reservation in the head of the list (insert fist). Lines 22-29 are finding the right place in ascending order by userId, if the list is already having the userId then lines 31-40 are finding the right place for jobId (ascending order by jobId). Updating the free field of tslot array is done by line 42-43. Deleting an existing reservation is shown in Algorithm 3. If the reservation will be deleted, is in head node of the list then it is done by lines 10-12. If the reservation will be deleted is not in head node of the list then it is done by lines 13-18. Updating the free field of tslot array is done by line 20-21.

Algorithm 2. Allocating/Adding a reservation

```
1  Procedure allocate(int userId, int jobId, int
tes, int start, int tls, int te)
2  //allocate a reservation with userID, jobID,
eStartTime=tes, lStartTime=tls,
3  //execTime=te at timeslot start

4  Dictionary :
5  finish : integer /*finish time of the job*/
6  n : node /*record of reserevation*/
7  pn : pointer to node

8  Algorithm :
9  finish ← start + te – 1
10 n = new node(userId, jobId, tes, tls, te)
11
12 if (tslot[start].p = nil) then
13    tslot[start].p ← n     //insert first
14 else if (tslot[start].p.userId > n.userId) then
15    n.next ← tslot[start].p
16    tslot[start].p ← n     //insert first
17 else if (tslot[start].p.userId = n.userId and
tslot[start].p.jobId > n.jobId) then
18    n.next ← tslot[start].p
19    tslot[start].p ← n     //insert first
20 else
21    pn ← tslot[start].p
22    while (pn.next!=nil AND pn.next.userId <
n.userId) do
23       pn ← pn.next
24    //here pn.userId < n.userId or pn.userId =
n.user.Id or pn.next = nill
25    if (pn.next = nil) then
26       pn.next ← n        //insert last
27    else if (pn.next.userId > n.userId)
28       n.next ← pn.next   //insert new userId
29       pn.next ← n
30    else //the same userId is found or
pn.next.userId = n.user.id
31       while(pn.next!=nil and
pn.next.userId=n.userId and pn.next.jobId<n.jobId)
do
32          pn ← pn.next
33       if(pn.next = nil) then
34          pn.next ← n    //insert last
35       else if(pn.next.userId != n.userId) then
36          n.next ← pn.next   //insert last for old
userId
37          pn.next ← n
38       else if (pn.next.jobId > n.jobId) then
39          n.next ← pn.next
40          pn.next ← n    //insert old userId
41
42 for(int i=start; i<=finish; i++)
43    tslot[i].free ← tslot[i].free -1  //update a
free node
```

Algorithm 3. Deleting a reservation

```
1  Procedure delete(int userId, int jobId)
```

```
2  //delete a reservation with userId, jobId
3  Dictionary :
4  finish : integer /*finish time of the job*/
5  pn : pointer to node
6  pdel : pointer to node
7  Algorithm :
8  finish ← start + te - 1

9  pn ← tslot[start].p
10 if(pn.userId = userId and pn.jobId = job.Id)
//delete first
11    tslot[start].p = pn.next
12    delete pn //delete pn from memory
13 while (pn.next.userId != userId or pn.next.jobId
!= jobId) do
14     pn ← pn.next
15
16 pdel ← pn.next
17 pn.next ← pdel.next
18 delete pdel
19
20 for(int i=start; i<=finish; i++)
21    tslot[i].free ← tslot[i].free -1  //update a
free node
```

Suppose User 11 wishing to reserve 3 time slots from 12 up to 14, for his/her 3 independent jobs as depicted in Table 2.

Table 2. User 11's Parameters of reservation request.

userID	jobID	t_{es}	t_{ls}	t_e
11	1	12	14	3
11	2	12	14	3
11	3	12	14	3

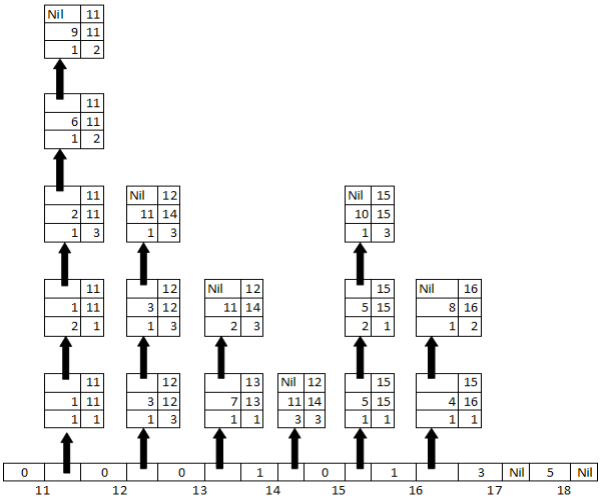

Fig 3: Data Structure for storing reservation request from User 11.

The result of FCFS-EDS for scheduling reservations from User 11 is a data structure as depicted in Fig 3.

Our proposed data structure has search time complexity $O(t_e.t_r)$ where t_e is execution time and t_r is the relax time (the earliest start time – the latest start time). Allocating/Adding reservation has time complexity $O(k+t_e)$, where k is the number of reservation in the list in for each timeslot. For deleting reservation our proposed data structure has time complexity $O(k+t_e)$.

4. RESULTS

User requests that are input for our FCFS-EDS, are generated randomly. The input specifications are:

a. The rate of incoming reservation requests are assumed to follow poison distribution with mean 2.0,
b. Execution time (t_e) for reservation requests are between 5 to 48 timeslots distributed uniformly,
c. Earlier starting time (t_{es}) is between 0 to 48 timeslots distributed uniformly,
d. Percentage of user request that are for flexible advance reservation is assumed to at most 50% (selected randomly),
e. Relax time (t_r) is between 1 to 24 timeslots distributed uniformly and $t_{ls} = t_{es} + t_r$
f. In the experiment it is assumed that a time slot is equal to 5 minutes (clock time).

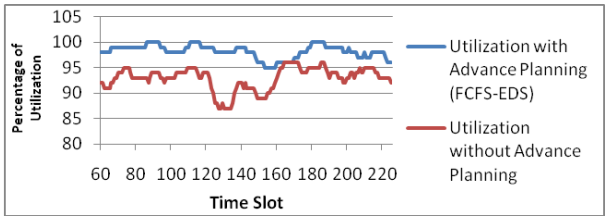

Fig 4: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning

We compare the performance of the proposed method (FCFE-EDS with advance planning) and an existing approach (flexible advance reservation strategy without advance planning). With above inputs and total number of compute node is 30 (maxCN=30), the utilization factors of both strategies are measured. The comparison of resource utilization of both strategies is shown in Fig 4. Percentage of utilization factor is calculated within sliding window of size 12 time slots (1 hour). Fig 4. shows that FCFS-EDS yields better utilization than the traditional strategy (without advance planning).

5. CONCLUSION

We have discussed the impact of advance reservation in Grid Computing. Advance reservation will decrease the resource utilization due to a lot of fragmentations. We have proposed a novel data structure for our strategy to increase resource utilization FCFS-EDS [15]. Our results show that this data structure for FCFS-EDS can increase resource utilization due to fragmentations caused by advance reservation.

6. REFERENCES

[1] Foster, I., and Kesselman, C. Computational Grids. Morgan Kaufmann, 1998.

[2] Joshy Joseph and Craig Fellenstein, Grid Computing, Prentice Hall PTR, New Jersey, December 30, 2003

[3] W. Cirne, F. Brasileiro, J. Sauve, N. Andrade, D. Paranhos, E. Santos-Neto, and R. Medeiros. Grid computing for bag of tasks applications. In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government, Sep 2003

[4] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: Killer application for the global grid? In Proc. of the 14th

International Symposium on Parallel and Distributed Processing (IPDPS), Cancun, Mexico, May 1–5 2000.

[5] Sulistio A., Buyya R., A Grid simulation infrastructure supporting advance reservation. In Proceedings 16th International Conference on Parallel and Distributed Computing and Systems, Cambridge, USA, November 9–11 2004.

[6] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In Proc. of the 7th International Workshop on Quality of Service, London, UK, 1999.

[7] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Workflow enactment in ICENI. UK e-Science All Hands Meeting, pages 894–900, September 2004.

[8] Maui Cluster Scheduler. http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php, Last Accessed June 2009

[9] B. Nitzberg, J. M. Schopf, and J. P. Jones. PBS Pro: Grid Computing and Scheduling Attributes. In Grid Resource Management: State of the Art and Future Trends, pages 183– 190. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[10] K. Kim. Extended DSRT Scheduling System. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (USA), Aug. 2000

[11] G Garimella, Advance CPU Reservations With The DSRT Scheduler. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (USA), 1999.

[12] Sun Grid Engine. http://gridengine.sunsource.net, 2008

[13] R. Buyya and A. Sulistio, Service and Utility Oriented, Data Centers and Grid Computing Environments: Challenges and Opportunities for Modeling and Simulation Communities, Keynote Paper, In Proceedings of the 41st Annual Simulation Symposium (ANSS'08), April 13–16, 2008, Ottawa, Canada.

[14] W. Smith, I. Foster and V. Taylor, "Scheduling with Advanced Reservations", In Proc. of the 14th IEEE International Symposium on Parallel and Distributed Processing (IPDPS'00), 2000, pp. 127-132.

[15] Rusydi Umar, Arun Agarwal, CR Rao, Advance Planning and Reservation in a Grid System, The Fourth International Conference on Networked Digital Technologies, NDT 2012, April 24 – 26, 2012, Dubai. To Appear in CCIS/LNCS Vol 7899

[16] L.-O. Burchard. Analysis of data structures for admission control of advance reservation requests. IEEE Transactions on Knowledge and Data Engineering, 17(3), 2005.

[17] L.-O. Burchard and H.-U. Heiss, "Performance Evaluation of Data Structures for Admission Control in Bandwidth Brokers," Proc. Int'l Symp. Performance Evaluation of Computer and Telecommunication Systems (SPECTS '02), Soc. for Modeling and Simulation Int'l, pp. 652-659, 2002.

[18] R. Guerin and A. Orda, "Networks with Advance Reservations: The Routing Perspective," Proc. IEEE INFOCOM '99, pp. 118-127, 2000.

[19] O. Schelen, A. Nilsson, J. Norrgard, and S. Pink, "Performance of QoS Agents for Provisioning Network Resources," Proc. Seventh Int'l Workshop on Quality of Service (IWQoS '99), pp. 17-26, 1999.

[20] A. Brodnik and A. Nilsson. A static data structure for discrete advance bandwidth reservations on the internet. In Proc. of Swedish National Computer Networking Workshop (SNCNW), Stockholm, Sweden, September 2003.

[21] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. Communications of the ACM, 31(10):1220{1227, 1988.

[22] T. Wang and J. Chen. Bandwidth tree – a data structure for routing in networks with advanced reservations. In Proc. of the 21st Intl. Performance, Computing, and Communications Conference (IPCCC), pages 37–44, Phoenix, USA, 2002.

[23] L. Yuan, C.-K. Tham, and A. L. Ananda. A probing approach for effective distributed resource reservation. In Proc. of the 2nd International Workshop on Quality of Service in Multiservice IP Networks, pages 672–688, Milan, Italy, February 2003. Springer-Verlag.

[24] Q. Xiong, C. Wu, J. Xing, L. Wu, and H. Zhang. A linked-list data structure for advance reservation admission control. In Proc. of the 3rd International Conference on Networking and Mobile Computing (ICCNMC), Zhangjiajie, China, August 2-4 2005.

[25] A. Sulistio, U. Cibej, S. Prasad, and R. Buyya, GarQ: An Efficient Scheduling Data Structure for Advance Reservations of Grid Resources, International Journal of Parallel, Emergent and Distributed Systems (IJPEDS), DOI: 10.1080/17445760801988979, April 4, 2008, Taylor & Francis Publication, UK.