The Report is Generated by DrillBit Plagiarism Detection Software

Submission Information

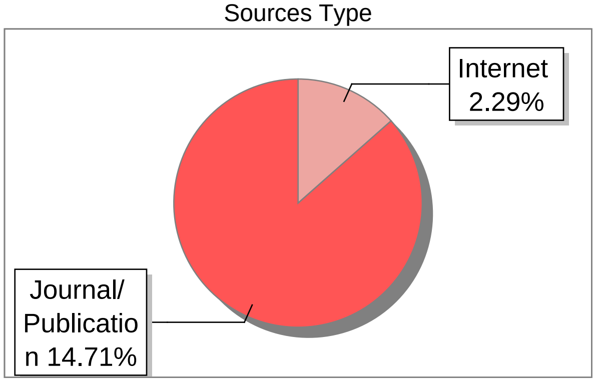
Author Name	Ardi Pujiyanta, Fiftin Noviyanto
Title	Design Of A Job Scheduling Data Structure For Grid Resources
Paper/Submission ID	2323206
Submitted by	perpustakaan.similarity@uad.ac.id
Submission Date	2024-09-18 13:27:39
Total Pages, Total Words	8, 3753
Document type	Article

Result Information

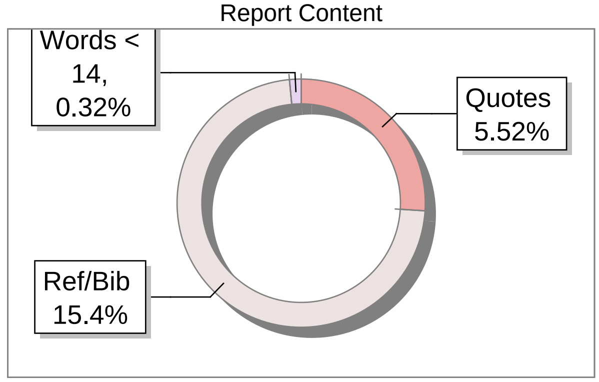
Similarity **17 %**

1 10 20 30 40 50 60 70 80 90

Sources Type



Report Content



Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Source: Excluded < 14 Words	Not Excluded
Excluded Source	**74 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

17	8	B	A-Satisfactory (0-10%) B-Upgrade (11-40%) C-Poor (41-60%) D-Unacceptable (61-100%)
SIMILARITY %	MATCHED SOURCES	GRADE	

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
2	Deadline-aware advance reservation scheduling algorithms for media pro by Barshan-215	1	Publication
5	cloudbus.org	<1	Publication
6	Improving naive Bayes classifier by dividing its decision regions by Zhi-yon-2011	1	Publication
8	jurnal.ugm.ac.id	<1	Internet Data
11	Thesis submitted to shodhganga - shodhganga.inflibnet.ac.in	10	Publication
12	cloudbus.org	3	Publication
22	moam.info	2	Internet Data
27	sportpedagogy.org.ua	<1	Publication

EXCLUDED SOURCES

1	jurnal.umj.ac.id	74	Publication

Volume 16 No. 2
July 2024
ISSN : 2085 – 1669
e-ISSN : 2460 – 0288
Website : jurnal.umj.ac.id/index.php/jurtek
Email : jurnalteknologi@umj.ac.id

Jurnal Teknologi
UNIVERSITAS MUHAMMADIYAH JAKARTA

Design Of A Job Scheduling Data Structure For Grid Resources

Ardi Pujiyanta[1,*], Fiftin Noviyanto[2]
[1,2]Informatics Engineering, Faculty of Industrial Technology, Universitas Ahmad Dahlan Yogyakarta, Jl. Ahmad Yani Tamanan Banguntapan Bantul Yogyakarta, 55166, Indonesia

*Corresponding author email: ardipujiyanta@tif.uad.ac.id

Jurnal Teknologi use only:
Received 19 May 2022; Revised 9 September 2023; Accepted 30 July 2024

ABSTRACT

Essentially, Grid computing is an infrastructure that offers high-speed computing capacity in a distributed system by utilizing geographically distributed resources. Grid resources are owned by different organizations and have their own policies and access models. Scheduling future jobs in a grid system requires a data structure capable of handling parallel jobs, known as the Message Passing Interface (MPI). A data structure model needs to be proposed to minimize search time, and efficiently add and remove MPI jobs. Data structures that support future scheduling models will improve resource utilization efficiency. This research proposes a data structure capable of handling future MPI job scheduling to increase resource utilization. Experimental results on the data structure show that the average memory consumption of the FCFS-LRH data structure is lower than that of FCFS and FCFS-EDS. For average empty timeslot searches, FCFS-LRH is faster than FCFS-EDS but slower than FCFS. The average data insertion speed of FCFS-LRH is faster than that of FCFS-EDS.

Keywords: MPI jobs, grid resources, data structures.

Introduction

Basically the Grid is an infrastructure that offers high speed computing capacity on a distributed system by utilizing geographically distributed resources. Grid resources are owned by different organizations and have their own policies and access models [1]. Grid computing has many names such as meta computing, scalable computing, global computing, internet computing and recently referred to as utility computing [2]. Efficient scheduling algorithms can make good use of the processing capacity of the grid system, thereby improving application performance [3].

First Come First Serve Ejecting Based Dynamic Scheduling (FCFS-EDS) reservation strategy is used to improve resource utilization in a grid system by using a local scheduler [4], [5]. The percentage of utilization performance is calculated in a sliding window with a size of 12 timeslots. The experimental results compared with the traditional strategy (flexible advance reservation strategy without planning) resulted in better utilization performance. The FCFS-LRH method utilizes user-submitted parameters to improve resource utilization and reduce job waiting times and can handle future scheduling of MPI jobs, to maximize resource utilization [6], [7].

DOI: https://dx.doi.org/10.24853/jurtek.16.2.283-290

ISSN : 2085 – 1669
e-ISSN : 2460 – 0288

User reservation requests in future scheduling need to be stored in the data structure. The data structure is used to store summary reservation request information and is the basis for direct input control in the resource reservation process. The data structure must be able to provide fast access and handle information efficiently. About 60 percent of the total processing time is needed for data structure management, 8 percent is used for selecting appropriate resources, and the remaining 32 percent is for resource management [8]. If application requests are provided for all potential reservation services in advance, then more time is required. For example, during the scanning and resource detection interval, the data structure processing time reaches 90% of the total time [9].

Scheduling future jobs in a grid system requires a data structure that can handle parallel jobs or is called a Message Passing Interface (MPI). A data structure model needs to be proposed to minimize search time, add and delete MPI jobs. Data structures that support future scheduling models will increase the efficiency of resource use. Advance Reservation (AR) in grid computing is an important research area because it allows users to gain concurrent access to resources and allows applications to execute in parallel. It also provides a guarantee of resource availability at a specified time in the future. Efficient data structures are important in minimizing the time complexity required to perform AR operations [10], [11].

In managing advance reservations (advance receipt control) in a grid system, an efficient data structure plays an important role in order to minimize the time for searching for available computing nodes, adding and deleting reservations. A user who requests a reservation in advance will get a fast response time, in order to provide results whether the reservation request is accepted or not.
The aim of this research is to test the memory consumption of the proposed FCFS-LRH data structure compared to the memory consumption of the FCFS and FCFS-EDS methods.

Methods

There are several data structures for managing reservations in advance which can generally be categorized into two types, namely timeslot data structures and continuous data structures. Meanwhile, the timeslot data structure is divided into two, namely static and dynamic. Static timeslots are divided into a fixed time period, while dynamic timeslots, the duration and number of timeslots are allowed to vary according to the number of reservation requests. The majority of timeslot-based reservation approaches proposed in the literature follow static solutions [12]. A data structure that stores and places each request at a fixed time interval is called a timeslot. In a continuous data structure each request is defined as its own time scale i.e. each advance reservation can start and finish at a flexible time. Examples of continuous data structures are link lists and examples of timeslot data structures are segment trees and calendar queues. The timeslot data structure approach has the advantage of limiting the amount of data stored so that memory consumption can be limited, and is easy to implement [8], [13].

Several studies whose approach is based on static timeslots are used to find optimal bandwidth solutions in media production [14]–[16]. The majority of current implementations in the field of advance reservations are supported by the timeslot data structure [17]–[21].

Proposed FCFS-LRH Data Structure

The data structures reported in the literature cannot be used for FCFS-LRH scheduling strategies, to manage advance planning. The proposed data structure for managing advance reservations using the FCFS-LRH scheduling strategy is influenced by the GarQ data structure because it has better performance among the data structures reported in the literature. GarQ [20] is modified so that it can handle flexible left-shift and right-shift planning, whereas GarQ can only handle rigid reservations. The properties added are t_{esr}, t_{lsr} and removing t_c in the data structure.

Proposed MPI Job Data Structure

The proposed data structure for MPI work can be seen in **Figure 1**, which is depicted as an array-based data structure. The array name is pSlot, the array index represents a specific timeslot. Each timeslot contains a list of reservations starting at that timeslot. The nodes or elements of the pSlot array are records that contain two fields (variables), namely the sv field which stores the number of virtual computing nodes available in the timeslot and the pj field is a link list pointer to other connected nodes. This node contains information about a job:

UserId: User identification
jobId : User can submit more than one independent job, jobid is used to identify.
tesr : Earliest start time to start work
tlsr : Latest start time to start the job
texe : Job execution time
jumCN : Number of resources required
node : Pointer to the reservation.

An example is given to make it easier to explain how the MPI data structure works, if the grid system has computing nodes in the physical view of maxC=5 (C0-C4), then the number of virtual nodes in the logical view is 5 (V0-V4) as well. **Table 1** shows the job arrival order with jumC≤maxC and jumJob is the number of jobs sent by userId. Consider the given parameters userId4 in **Table 1**. The information given is as follows, user 4 has reserved 4 timeslots in the pSlot array, with a timeslot index between 6 to 9, and the given job cannot be postponed or shifted because texe =4 and tesr = tlsr =6.

Suppose userID9 sends a job 3 timeslots from 8 to 13, requires 2 compute nodes for one independent job and can be delayed until timeslot 13 (tesr=8, tlsr=13, texe=3, jumJob=1, jumCN=2), shown in **Figure 2**. **Figure 3** shows the mapping results on actual nodes for MPI jobs.

The data structure resulting from storing all reservation requests in **Table 1** can be seen in **Figure 4**. As shown in **Figure 4** there is one job that starts in timeslot 4 with the remaining timeslot sv=4, two jobs that start in timeslot 5 with the remaining timeslot sv= 2, one job starting in timeslot 6 with remaining timeslot sv=0, one job starting in timeslot 7 with remaining timeslot sv=0, one job starting in timeslot 8 with remaining timeslot sv=0, no jobs starting from timeslot 9 with remaining timeslot sv=1, three jobs start in timeslot 10 with remaining timeslot sv=1. Reservation node next=nil if it does not point to a reservation node. In the pSlot array, the pointer pj = nil if it does not point to a reservation node.

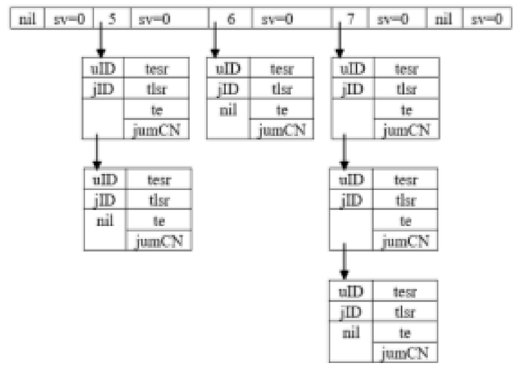


Figure 1. Proposed MPI Job Data Structure.

Table 1. MPI Job Reservation Request Parameters

UserId	t_{esr}	t_{lsr}	te	JumCN	JumJob
[illegible]	4	4	2	1	1
2	5	5	2	1	1
3	5	5	3	1	1
4	6	6	4	3	1
5	7	7	1	1	1
6	8	8	2	2	1
7	8	10	4	1	1
8	9	10	3	2	1

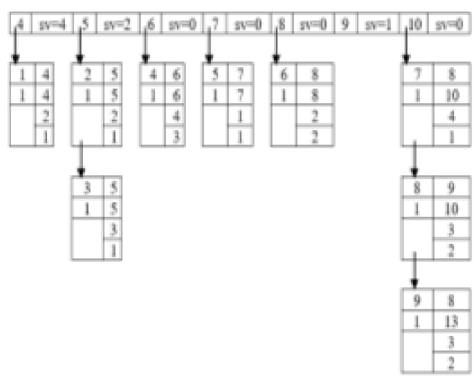


Figure 2. MPI Data Structure for Storing Reservations from **Table 1**.

ISSN : 2085 – 1669
e-ISSN : 2460 – 0288

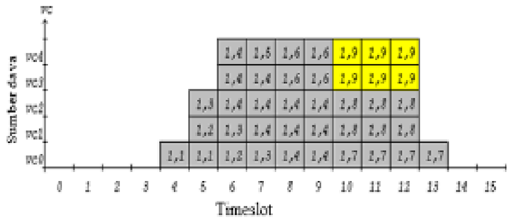


Figure 3. Placement of userID9 in the logical view using the FCFS-LRH method.

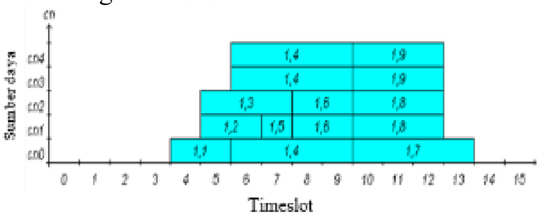


Figure 4. Mapping results on actual nodes for MPI jobs.

Data structure components in timeslots
SlotNo :0 [] Sv: 5
SlotNo :1 [] Sv: 5
SlotNo :2 [] Sv: 5
SlotNo :3 [] Sv: 5
SlotNo :4 [1 1 4 4 2 1] Sv: 4
SlotNo :5 [1 2 5 5 2 1, 1 3 5 5 3 1] Sv: 2
SlotNo :6 [1 4 6 6 4 3] Sv: 0
SlotNo :7 [1 5 7 7 1 1] Sv: 0
SlotNo :8 [1 6 8 8 2 2] Sv: 0
SlotNo :9 [] Sv: 0
SlotNo :10 [1 7 8 10 4 1, 1 8 9 10 3 2, 1 9 8 13 3 2] Sv: 0
SlotNo :11 [] Sv: 0
SlotNo :12 [] Sv: 0
SlotNo :13 [] Sv: 4
SlotNo :14 [] Sv: 5
SlotNo :15 [] Sv: 5
SlotNo :16 [] Sv: 5
SlotNo :17 [] Sv: 5
SlotNo :18 [] Sv: 5
SlotNo :19 [] Sv: 5
Additions to MPI Jobs:
There are 4 possible cases for adding a new reservation in a data structure:
1. The reservation list of tesr elements in the pSlot array is empty, add a new reservation as the first reservation node (insert it first). Lines 3 to 4 in Algorithm 1 are used to add reservations.
2. The first node of the reservation list has a userID that is greater than the incoming userID, so add a new reservation as the first node of the reservation list. This addition is made in Algorithm 1, lines 7 to 10.
3. The first element of the reservation list has the same userID as the incoming userID and the jobID of the first node of the reservation list is greater than the incoming jobID. Add the new reservation as the first component of the reservation list. Algorithm 1 in lines 11 to 16 is used for job addition. Fix the pSlot array shown in lines 19 to 21. Lines 24 to 31 update the timeslots in the logical view.
4. In this case the new reservation will be entered in the middle or last of the reservation list, shown in Algorithm 2.
Step to shift work components starting at timeslot.

1. If pSlot is empty, insert row 5.
2. Code lines 1 to 7 are used to check if there is a job starting in the shiftable timeslot.
3. If yes, check to see if any work can be shifted
4. Shift the job, and update the free nodes accordingly
5. Save the shift on stack line 32.

When inserting in the middle there are 3 possibilities:
• condition 1. still in the time range (line 10), the time range for the incoming job is smaller than the stored job tlsr, then shift the saved job to make space for the incoming job to be inserted.
• condition 2, the next slot is empty (line 18), shift the job in, check whether it can be inserted, if yes it can be inserted, add the job in (line 29), update the pSlot on the right side (line 30) and the left side (line 31).
• condition 3, do not shift jobs from the same userId.

Algorithm 1

```
1 procedure append(timeSlot, Component comp)
2 insert ← false;
3 If (pSlot[timeSlot].listComp.isEmpty()) then
4         pSlot[timeSlot]←listComp(comp);
5         else
6   for (int i=0) to ( i<pSlot[timeSlot].listComp.size()) do
7  If (comp.userID < pSlot[timeSlot].listComp(i , userID)) then
8  pSlot[timeSlot]←listComp(i, comp);
9         insert ← true;
10        break;
11 else if (comp.userID ==
```

Jurnal Teknologi 16 (2) pp 283-290 © 2024

```
pSlot[timeSlot].listComp(i , userID)) then
  if (comp.jobID <
pSlot[timeSlot].listComp(i , jobID)) then
  pSlot[timeSlot]←listComp(i, comp);
                                   insert ← true;
                                   break;
                                Endif
                      Endif
               Endfor
                      If (insert == false) then
  pSlot[timeSlot]←listComp(comp);
                      Endif
  Endif
  //==Update cell
        For (i=0) to (i<comp.execTime) do
    For (j = 0) to (j<cell[timeSlot].length) do
  For (j = 0) to (j<cell[timeSlot].length) do
  If (cell[timeSlot+i][j].userID = = 0) then
cell[timeSlot+i][j]←Cell(comp.userID,comp
.jobID,comp.StartTime, comp.lStartTime);
                                break;
                      Endfor
               Endfor
               Endfor
```

Algorithm 2

```
Function boolean insRes(userID,
jumCNneeded)
  integer i, j
  Component comp, S1
  succ ← false
   If (pSlot[time].listComp.Empty) then
         return succ
   else
         For (i=0) to
(i<pSlot[time].listComp.size) do
              comp ← pSlot[time].listComp(i)
          If (comp.tlsstartTime-time>0  AND
pSlot[comp.endTime+1].getFree>=comp.jumC
N)  then
              insertComp.getinsert← add(comp);
insertComp.jumCN←insertComp.jumCN+
comp.jumCN;
  If (insertComp.getjumCN >=
jumCNneeded) then
                  break
              Endif
          Endif
          Endfor
   If (!insertComp.getinsert.Empty) then
     For(i=0) to (i<insertComp.insert.size) do
        comp = insertComp.insert.get(i);
For (j=0) to (j<pSlot[time].listComp.size)
do
  S1 = pSlot[time].listComp.get(j);
   If (comp.userID == S1.userID AND
comp.jobID == S1.jobID) then
                        break;
                  Endif
             Endfor
        pSlot[time].listComp.remove(j)
removeCell(comp.userID, comp.jobID)
        comp.setstartTime(time+1);
        Append(time+1, comp);
           Endfor
                  succ = true;
        Endif
        insertComp ← new InsertComponent()
        Endif
                  return succ;
  Endfunc
```

Results and Discussions

Experiments have been carried out to measure the memory consumption used by the FCFS-LRH data structure compared to FCFS which uses the LIST data structure and EDS which uses the link list data structure. Testing is carried out by:

1. Generate workload data with a total of 400 to 800 data, which refers to research [6].
2. The results of generating workload data will be used by the FCFS-LRH, FCFS and FCFS-EDS data structures, then the results will be compared.

The results show that the LRH data structure is smaller in memory consumption is shown in **Table 2** and **Figure 5**. The LRH data structure does not perform well when searching for jobs compared to rigid FCFS, because LRH has to shift jobs so that incoming jobs can be accepted, shown in **Table 3** and **Figure 6**. **Table 4** and **Figure 7** shows that the time required to add work to the data structure using the LRH method is faster than the EDS method. **Table 5** and **Figure 8** show that the time required for deleting work on data structures using the LRH method is faster than the EDS method.

Overall the FCFS-LRH data structure is better than FCFS and FCFS-EDS, because the LRH data structure can shift left and right in scheduling. While EDS can only slide right,

ISSN : 2085 – 1669
e-ISSN : 2460 – 0288

FCFS cannot slide left and right because it is rigid.

Table 2. Memory Consumption Used by FCFS, EDS and LRH.

	Number of jobs			
Method	400	600	750	Average
FCFS	4340	9800	12600	8913.3
EDS	1690.1	1727.8	1996.6	1804.8
LRH	1613.2	1658.1	1733.8	1668.4

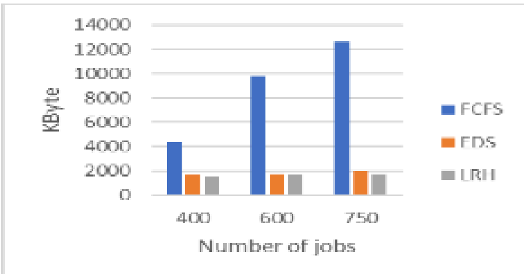


Figure 5. Total Memory Consumption of FCFS, EDS and LRH Data Structures.

Table 3. Searching Data Structure Using FCFS, EDS and LRH Based on Number of Jobs.

	Number of Jobs			
	400	600	750	Average
FCFS	70	56.3	37	54.4
EDS	222	175	190	195.7
LRH	215	180.2	160.4	185.2

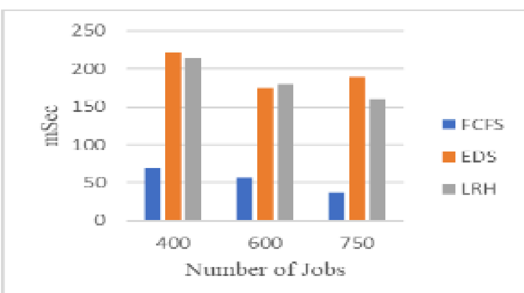


Figure 6. Searching Data Structure Using FCFS, EDS and LRH Based on Number of Jobs

Table 4. Add to Data Structure Using EDS and LRH Based on Number of Jobs.

	400	600	750	Average
LRH	5.85	8.75	7.5	7.37
EDS	6.5	8.4	9.505	8.14

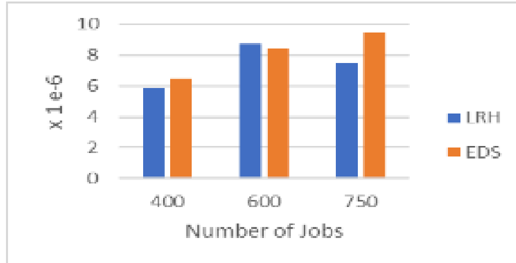


Figure 7. Add to Data Structure Using EDS and LRH Methods Based on Number of Jobs

Table 5. Delete Data Structure Using EDS and LRH Methods Based on Number of Jobs.

	400	600	750	Average
LRH	15.38	29.59	19.12	21.36
EDS	16.65	38.56	19.24	24.82

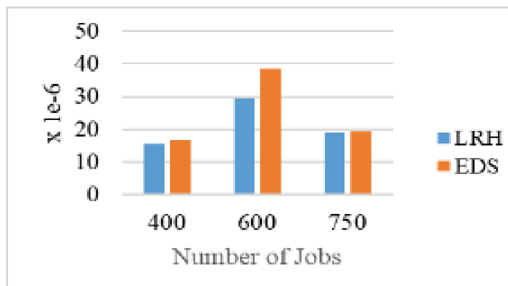


Figure 8. Delete Data Structure Using EDS and LRH Based on Number of Jobs.

Conclusions

Experimental results on data structures show that the average memory consumption of the FCFS-LRH data structure is smaller than FCFS and FCFS-EDS. The average search for empty timeslots of FCFS-LRH is faster than FCFS-EDS and slower than FCFS. FCFS-LRH's average data insert is faster than FCFS-EDS.

Acknowledgment

The researcher would like to express his gratitude to the Informatics Study Program at Ahmad Dahlan University, which has provided the freedom to use the laboratory for research.

Funding
This research was carried out independently, without financial assistance from external parties.

Author Contributions
The first researcher has a role in designing the proposed method, coding. The second author had the role of results analysis.

Conflict of interest
The authors declare no conflict of interest. There were no outside funders for this research.

References
[1] M. Singh, "An Overview of Grid Computing," *Proc. - 2019 Int. Conf. Comput. Commun. Intell. Syst. ICCCIS 2019*, vol. 2019-Janua, pp. 194–198, 2019, doi: 10.1109/ICCCIS48478.2019.8974490.
[2] R. Nawaz, W. Y. Zhou, M. U. Shahid, and O. Khalid, "A qualitative comparison of popular middleware distributions used in grid computing environment," *2nd Int. Conf. Comput. Commun. Syst. ICCCS 2017*, pp. 36–40, 2017, doi: 10.1109/CCOMS.2017.8075262.
[3] L. Feng and G. Wei-Wei, "Research and Design of Task Scheduling Method Based on Grid Computing," *Proc. - 2nd Int. Conf. Smart City Syst. Eng. ICSCSE 2017*, pp. 188–192, 2017, doi: 10.1109/ICSCSE.2017.54.
[4] R. Umar, A. Agarwal, and C. R. Rao, "Advance Planning and Reservation in a Grid System," *Commun. Comput. Inf. Sci.*, vol. 293 PART 1, pp. 161–173, 2012, doi: 10.1007/978-3-642-30507-8_15.
[5] A. Sulistio *et al.*, "An Adaptive Scoring Job Scheduling algorithm for grid computing," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5, no. 1, pp. 68–72, 2015, doi: 10.1177/1094342006068414.
[6] A. Pujiyanta, L. E. Nugroho, and Widyawan, "Resource allocation model for grid computing environment," *Int. J. Adv. Intell. Informatics*, vol. 6, no. 2, pp. 185–196, 2020, doi: https://doi.org/10.26555/ijain.v6i2.496.
[7] A. Shukla, S. Kumar, and H. Singh, "An improved resource allocation model for grid computing environment," *Int. J. Intell. Eng. Syst.*, vol. 12, no. 1, pp. 104–113, 2019, doi: 10.22266/IJIES2019.0228.11.
[8] L. O. Burchard, "Analysis of data structures for admission control of advance reservation requests," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 3, pp. 413–424, 2005, doi: 10.1109/TKDE.2005.40.
[9] L.-O. Burchard and H.-U. Heiss, "Performance Evaluation of Data Structures for Admission Control in Bandwidth Brokers," *Int. Symp. Perform. Eval. Comput. Telecommun. Syst. (SPECTS '02)*, pp. 652–659, 2002, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.1535&rep=rep1&type=pdf.
[10] B. Li, Y. Pei, H. Wu, and B. Shen, "Resource availability-aware advance reservation for parallel jobs with deadlines," *J. Supercomput.*, vol. 68, no. 2, pp. 798–819, 2014, doi: 10.1007/s11227-013-1067-8.
[11] A. Pujiyanta, L. E. Nugroho, and Widyawan, "Job Scheduling Strategies in Grid Computing," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 12, no. 3, pp. 1293–1300, 2022, doi: 10.18517/ijaseit.12.3.10147.
[12] N. Charbonneau and V. M. Vokkarane, "A survey of advance reservation routing and wavelength assignment in wavelength-routed WDM networks," *IEEE Commun. Surv. Tutorials*, vol. 14, no. 4, pp. 1037–1064, 2012, doi: 10.1109/SURV.2011.111411.00054.
[13] M. D. de Assuncao, "Enhanced Red-Black-Tree Data Structure for Facilitating the Scheduling of Reservations," 2015, [Online]. Available: http://arxiv.org/abs/1504.00785.
[14] A. Gadkar, T. Entel, J. M. Plante, and V. M. Vokkarane, "Slotted advance reservation for multicast-incapable optical wavelength division multiplexing networks," *J. Opt. Commun. Netw.*, vol. 6, no. 3, pp. 340–

ISSN : 2085 – 1669
e-ISSN : 2460 – 0288

354, 2014, doi: 10.1364/JOCN.6.000340.

[15] M. Barshan, H. Moens, J. Famaey, and F. De Turck, "Deadline-aware advance reservation scheduling algorithms for media production networks," *Comput. Commun.*, vol. 77, no. 2015, pp. 26–40, 2016, doi: 10.1016/j.comcom.2015.10.016.

[16] M. Barshan, H. Moens, B. Volckaert, and F. De Turck, "A comparative analysis of flexible and fixed size timeslots for advance bandwidth reservations in media production networks," *2016 7th Int. Conf. Netw. Futur. NOF 2016*, 2017, doi: 10.1109/NOF.2016.7810118.

[17] R. Brown, "Calendar Queues: A Fast 0(1) Priority Queue Implementation for the Simulation Event Set Problem," *Commun. ACM*, vol. 31, no. 10, pp. 1220–1227, 1988, doi: 10.1145/63039.63045.

[18] R. A. Guerin and A. Orda, "Networks with advance reservations: The routing perspective," *Proc. - IEEE INFOCOM*, vol. 1, pp. 118–127, 2000, doi: 10.1109/infcom.2000.832180.

[19] A. Brodnik and A. Nilsson, "A Static Data Structure for Discrete Advance Bandwidth Reservations on the Internet," *Swedish Natl. Comput. Netw. Work.*, vol. 41, pp. 1–15, 2003.

[20] A. Sulistio, U. Cibej, S. K. Prasad, and R. Buyya, "GarQ: An efficient scheduling data structure for advance reservations of grid resources," *Int. J. Parallel, Emergent Distrib. Syst.*, vol. 24, no. 1, pp. 1–19, 2009, doi: 10.1080/17445760801988979.

[21] L. Wu, P. Dang, T. Yu, and L. Nie, "Research on efficient non-slotted tree structures for advance reservation," *Commun. Comput. Inf. Sci.*, vol. 401, pp. 50–61, 2013, doi: 10.1007/978-3-642-53959-6_6.