HASIL CEK_Pujiyanta_MPI jobs, grid resources, data structures

by Ardi Pujiyanta Designing A Data Structure

Submission date: 26-Sep-2023 09:09AM (UTC+0700)
Submission ID: 2177046657
File name: Manuscript_Template_JurnalTeknologi-2023-English_Version_1.doc (1.37M)
Word count: 3288
Character count: 17936

Volume 16 No. 1
January 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

Manuscript Template Jurnal Teknologi

DESIGNING A DATA STRUCTURE FOR JOB SCHEDULING ON GRID RESOURCES

Ardi Pujiyanta[1,*]

[1]*Teknik Informatika, Fakultas Teknologi Industri, Universitas Ahmad Dahlan Yogyakarta, Jl. Ahmad Yani Tamanan Banguntapan Bantul Yogyakarta, 55166*

*ardipujiyanta@tif.uad.ac.id

Jurnal Teknologi use only: Received date here; revised date here; accepted date here

ABSTRACT

Grid computing is an infrastructure that offers high-speed computing capacity on a distributed system by utilizing geographically distributed resources. Grid resources owned by different organizations and have their policies and access models. Scheduling future jobs in a grid system requires a data structure that can handle parallel jobs which is called Message Passing Interface (MPI). A data structure model needs to be proposed to minimize the search time and add and delete MPI jobs. Data structures that support future scheduling models will improve resource use efficiency. This research proposes a data structure that can handle the MPI work schedule in the future to increase resource utilization. The experimental results on the data structure show that the average memory consumption of the FCFS-LRH data structure is smaller than that of FCFS and FCFS-EDS. Searching for the average empty timeslots, FCFS-LRH is faster than FCFS-EDS and slower than FCFS. The average data insert of FCFS-LRH is faster than 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(Singh 2019). Grid computing has many names such as metacomputing, scalable computing, global computing, internet computing and recently referred to as utility computing(Nawaz et al. 2017). Efficient scheduling algorithms can make good use of the processing capacity of the grid system, thereby improving application performance(Feng and Wei-Wei 2017).

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(Umar, Agarwal, and Rao 2012)(Sulistio et al. 2015). The percentage of

DOI: https://dx.doi.org/10.24853/jurtek.16.x.xxx-xxx

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

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(Pujiyanta, Nugroho, and Widyawan 2020)(Shukla, Kumar, and Singh 2019).

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(L. O. Burchard 2005). 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(L.-O. Burchard and Heiss 2002).

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(Li et al. 2014)(Pujiyanta, Nugroho, and Widyawan 2022).

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.

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(Charbonneau and Vokkarane 2012). 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(L. O. Burchard 2005)(de Assuncao 2015).

Several studies whose approach is based on static timeslots are used to find optimal bandwidth solutions in media production(Gadkar et al. 2014)(Barshan et al. 2016)(Barshan et al. 2017). The majority of current implementations in the field of advance reservations are supported by the timeslot data structure(Brown 1988)(Guerin and Orda 2000)(Brodnik and Nilsson 2003)(Sulistio et al. 2009)(Wu et al. 2013)
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, *j*obid 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.

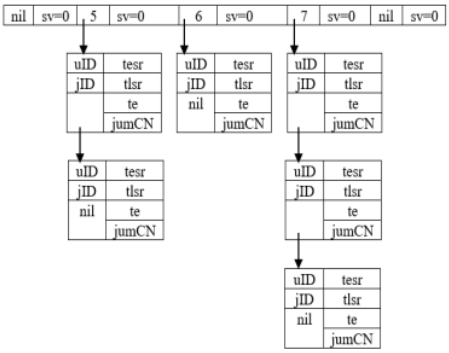


Figure 1 Proposed MPI Job Data Structure.

Table 1 MPI Job Reservation Request Parameters

UserId	t_{esr}	t_{lsr}	te	JumCN	JumJob
1	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

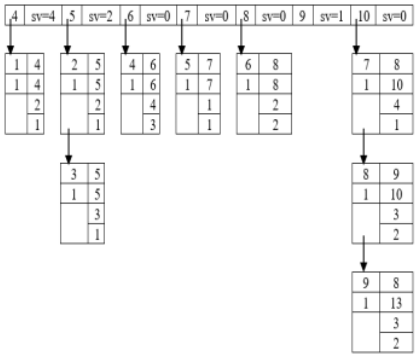
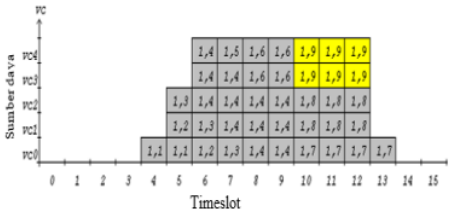


Figure 4 MPI Data Structure for Storing Reservations from Table 1.



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

Figure 2 Placement of userID9 in the logical view using the FCFS-LRH method.

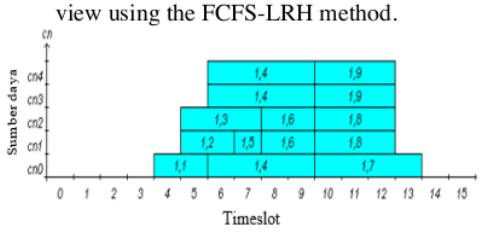


Figure 3 Mapping results on actual nodes for MPI jobs.

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 ==
pSlot[timeSlot].listComp(i , userID)) then
12  if (comp.jobID <
pSlot[timeSlot].listComp(i , jobID)) then
13 pSlot[timeSlot]←listComp(i, comp);
14                          insert ← true;
15                          break;
16                    Endif
17              Endif
18          Endfor
```

```
19                  If (insert == false) then
20  pSlot[timeSlot]←listComp(comp);
21                  Endif
22   Endif
23   //==Update cell
24        For (i=0) to (i<comp.execTime) do
25     For (j = 0) to (j<cell[timeSlot].length) do
26  For (j = 0) to (j<cell[timeSlot].length) do
27  If (cell[timeSlot+i][j].userID = = 0) then
28cell[timeSlot+i][j]←Cell(comp.userID,comp
.jobID,comp.StartTime, comp.lStartTime);
29                          break;
30                  Endfor
31              Endfor
32              Endfor
```

Algorithm 2

```
1 Function boolean insRes(userID,
jumCNneeded)
2  integer i, j
3  Component comp, S1
4   succ ← false
5    If (pSlot[time].listComp.Empty) then
6         return succ
7    else
8         For (i=0) to
(i<pSlot[time].listComp.size) do
9              comp ← pSlot[time].listComp(i)
10         If (comp.tlsstartTime-time>0  AND
pSlot[comp.endTime+1].getFree>=comp.jumC
N)  then
11              insertComp.getinsert← add(comp);
12 insertComp.jumCN←insertComp.jumCN+
comp.jumCN;
13  If (insertComp.getjumCN >=
jumCNneeded) then
14                 break
15              Endif
16         Endif
17         Endfor
18    If (!insertComp.getinsert.Empty) then
19      For(i=0) to (i<insertComp.insert.size) do
20         comp = insertComp.insert.get(i);
21 For (j=0) to (j<pSlot[time].listComp.size)
do
22  S1 = pSlot[time].listComp.get(j);
23   If (comp.userID == S1.userID AND
comp.jobID == S1.jobID) then
24                  break;
25               Endif
26            Endfor
27         pSlot[time].listComp.remove(j)
28 removeCell(comp.userID, comp.jobID)
29         comp.setstartTime(time+1);
30      Append(time+1, comp);
34         Endfor
35              succ = true;
36      Endif
37      insertComp ← new InsertComponent()
38      Endif
39              return succ;
40  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, the number of workloads used is between 400 to 800. 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. Tables 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.

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

Method	Number of jobs			Average
	402	605	787	
FCFS	4347	9856	12700	8967,7
EDS	1690,34	1728,2	1997,6	1805,4
LRH	1613,77	1659,3	1734,8	1669,3

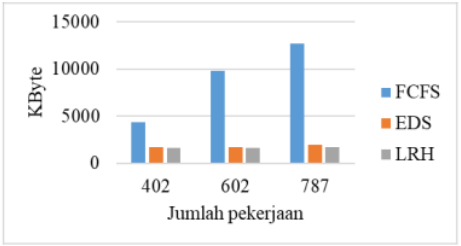


Figure 5. Total Memory Consumption of FCFS, EDS and LRH Data Structures.

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

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

	Number of Jobs			
	402	602	787	Average
FCFS	70,29	56,83	39,57	55,56
EDS	224,76	176,5	192,5	197,92
LRH	218,22	181,62	162,84	187,56

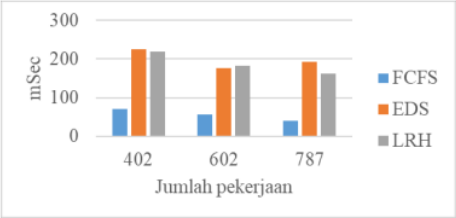


Figure 6 Searching Data Structure Using FCFS, EDS and LRH Based on Number of Jobs

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

	402	602	787	Average
LRH	5,97	8,96	8,05	7,7
EDS	6,61	8,52	10,96	8,7

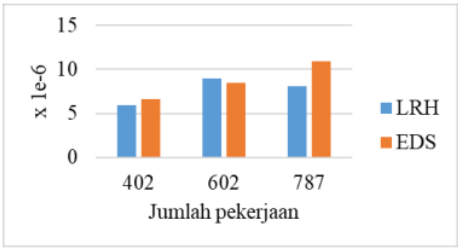


Figure 7 Add to Data Structure Using EDS and LRH Methods Based on Number of Jobs

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

	402	602	787	Average
LRH	16,48	30,49	19,27	22,079
EDS	17,79	39,46	19,27	25,507

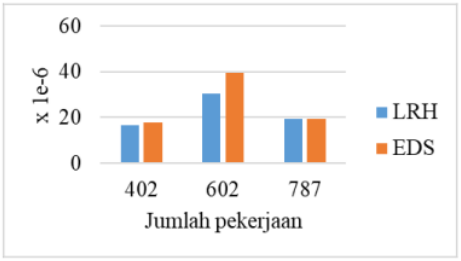


Figure 8 Delete Data Structure Using EDS and LRH Based on Number of Jobs.

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.

References

Assuncao, Marcos Dias de. 2015. "Enhanced Red-Black-Tree Data Structure for Facilitating the Scheduling of Reservations." http://arxiv.org/abs/1504.00785.

Barshan, Maryam, Hendrik Moens, Jeroen Famaey, and Filip De Turck. 2016. "Deadline-Aware Advance Reservation Scheduling Algorithms for Media Production Networks." *Computer Communications* 77 (2015): 26–40. https://doi.org/10.1016/j.comcom.2015.10.016.

Barshan, Maryam, Hendrik Moens, Bruno Volckaert, and Filip De Turck. 2017. "A Comparative Analysis of Flexible and Fixed Size Timeslots for Advance Bandwidth Reservations in Media Production Networks." *2016 7th International Conference on the Network of the Future, NOF 2016*. https://doi.org/10.1109/NOF.2016.7810118.

Brodnik, Andrej, and Andreas Nilsson. 2003. "A Static Data Structure for Discrete Advance Bandwidth Reservations on the Internet." *Swedish National Computer Networking Workshop (SNCNW)* 41: 1–15.

Brown, R. 1988. "Calendar Queues: A Fast 0(1) Priority Queue Implementation for

the Simulation Event Set Problem." *Communications of the ACM* 31 (10): 1220–27. https://doi.org/10.1145/63039.63045.

Burchard, L.-O., and H.-U. Heiss. 2002. "Performance Evaluation of Data Structures for Admission Control in Bandwidth Brokers." *International Symposium of Performance Evaluation of Computer and Telecommunication Systems (SPECTS '02)*, 652–59. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.1535&rep=rep1&type=pdf.

Burchard, Lars Olof. 2005. "Analysis of Data Structures for Admission Control of Advance Reservation Requests." *IEEE Transactions on Knowledge and Data Engineering* 17 (3): 413–24. https://doi.org/10.1109/TKDE.2005.40.

Charbonneau, Neal, and Vinod M. Vokkarane. 2012. "A Survey of Advance Reservation Routing and Wavelength Assignment in Wavelength-Routed WDM Networks." *IEEE Communications Surveys and Tutorials* 14 (4): 1037–64. https://doi.org/10.1109/SURV.2011.111411.00054.

Feng, Liu, and Guo Wei-Wei. 2017. "Research and Design of Task Scheduling Method Based on Grid Computing." *Proceedings - 2nd International Conference on Smart City and Systems Engineering, ICSCSE 2017*, 188–92. https://doi.org/10.1109/ICSCSE.2017.54.

Gadkar, Arush, Tim Entel, Jeremy M. Plante, and Vinod M. Vokkarane. 2014. "Slotted Advance Reservation for Multicast-Incapable Optical Wavelength Division Multiplexing Networks." *Journal of Optical Communications and Networking* 6 (3): 340–54. https://doi.org/10.1364/JOCN.6.000340.

Guerin, Roch A., and Ariel Orda. 2000. "Networks with Advance Reservations: The Routing Perspective." *Proceedings - IEEE INFOCOM* 1: 118–27. https://doi.org/10.1109/infcom.2000.832180.

Li, Bo, Yijian Pei, Hao Wu, and Bin Shen. 2014. "Resource Availability-Aware Advance Reservation for Parallel Jobs with Deadlines." *Journal of Supercomputing* 68 (2): 798–819. https://doi.org/10.1007/s11227-013-1067-8.

Nawaz, Rab, Wu Yang Zhou, Muhammad Usman Shahid, and Osman Khalid. 2017. "A Qualitative Comparison of Popular Middleware Distributions Used in Grid Computing Environment." *2nd International Conference on Computer and Communication Systems, ICCCS 2017*, 36–40. https://doi.org/10.1109/CCOMS.2017.8075262.

Pujiyanta, Ardi, Lukito Edi Nugroho, and Widyawan. 2020. "Resource Allocation Model for Grid Computing Environment." *International Journal of Advances in Intelligent Informatics* 6 (2): 185–96. https://doi.org/https://doi.org/10.26555/ijain.v6i2.496.

———. 2022. "Job Scheduling Strategies in Grid Computing." *International Journal on Advanced Science, Engineering and Information Technology* 12 (3): 1293–1300. https://doi.org/10.18517/ijaseit.12.3.10147.

Shukla, Anju, Shishir Kumar, and Harikesh Singh. 2019. "An Improved Resource Allocation Model for Grid Computing Environment." *International Journal of Intelligent Engineering and Systems* 12 (1): 104–13. https://doi.org/10.22266/IJIES2019.0228.11.

Singh, Manjeet. 2019. "An Overview of Grid Computing." *Proceedings - 2019 International Conference on Computing, Communication, and Intelligent Systems, ICCCIS 2019* 2019-Janua: 194–98. https://doi.org/10.1109/ICCCIS48478.2019.8974490.

Sulistio, Anthony, Uros Cibej, Sushil K. Prasad, and Rajkumar Buyya. 2009. "GarQ: An Efficient Scheduling Data Structure for Advance Reservations of Grid Resources." *International Journal of Parallel, Emergent and Distributed Systems* 24 (1): 1–19. https://doi.org/10.1080/17445760801988979.

Sulistio, Anthony, Kyong Hoon Kim, Rajkumar Buyya, Ahuva W. Mu'alem, Dror G. Feitelson, Peng Xiao, Zhigang

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

Hu, et al. 2015. "An Adaptive Scoring Job Scheduling Algorithm for Grid Computing." *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5 (1): 68–72. https://doi.org/10.1177/1094342006068414.

Umar, Rusydi, Arun Agarwal, and C. R. Rao. 2012. "Advance Planning and Reservation in a Grid System." *Communications in Computer and Information Science* 293 PART 1: 161–73. https://doi.org/10.1007/978-3-642-30507-8_15.

Wu, Libing, Ping Dang, Tianshui Yu, and Lei Nie. 2013. "Research on Efficient Non-Slotted Tree Structures for Advance Reservation." *Communications in Computer and Information Science* 401: 50–61. https://doi.org/10.1007/978-3-642-53959-6_6.

HASIL CEK_Pujiyanta_MPI jobs, grid resources, data structures

ORIGINALITY REPORT

24%	22%	10%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.uohyd.ac.in Internet Source	9%
2	Submitted to Universitas Pancasila Student Paper	2%
3	www.insightsociety.org Internet Source	2%
4	"Frontiers in Internet Technologies", Springer Nature, 2013 Publication	1%
5	gridbus.csse.unimelb.edu.au Internet Source	1%
6	Ardi Pujiyanta, Lukito Edi Nugroho, Widyawan. "Advance Reservation for Parametric Job on Grid Computing", 2019 Fourth International Conference on Informatics and Computing (ICIC), 2019 Publication	1%
7	doczz.net Internet Source	1%

8	biblio.ugent.be Internet Source	1 %
9	www.ijain.org Internet Source	1 %
10	Ardi Pujiyanta, Fiftin Novianto. "Job Scheduling on Grid Computing Using First Fit, Best Fit, and Worst Fit", Khazanah Informatika : Jurnal Ilmu Komputer dan Informatika, 2022 Publication	1 %
11	media.neliti.com Internet Source	1 %
12	eprints.uad.ac.id Internet Source	<1 %
13	Liu Feng, Guo Wei-Wei. "Retracted: Research and Design of Task Scheduling Method Based on Grid Computing", 2017 International Conference on Smart City and Systems Engineering (ICSCSE), 2017 Publication	<1 %
14	www.mdpi.com Internet Source	<1 %
15	"Cognitive Internet of Medical Things for Smart Healthcare", Springer Science and Business Media LLC, 2021 Publication	<1 %

16	Lars-Olof Burchard. "Networks with Advance Reservations: Applications, Architecture, and Performance", Journal of Network and Systems Management, 2005 Publication	<1 %
17	docobook.com Internet Source	<1 %
18	epdf.pub Internet Source	<1 %
19	garuda.kemdikbud.go.id Internet Source	<1 %
20	www.uniassignment.com Internet Source	<1 %
21	Communications in Computer and Information Science, 2012. Publication	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		