CORS - A Cost Optimized Resource Reservation Scheme for Grid

Rifat Shahriyar, Md. Mostofa Akbar, M. Sohel Rahman, Md. Faizul Bari and Shampa Shahriyar
Department of Computer Science and Engineering (CSE)
Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh

Abstract— *The basic objective of grid computing is to support resource sharing among individuals and institutions within a networked infrastructure. Managing various resources in highly dynamic grid environments is a complex and challenging problem. Some approaches apply algorithms for resource management to grid but fails to provide any generalized solutions. Most of the approaches are based on a simple architecture considering computer as the main resource. But the real architecture of grid computing is a complex one especially if we consider various resources of a computer (e.g., processor, memory etc.) during resource management. In grids, sometimes assurance is needed for successful completion of jobs on shared resources. Such guarantees can only be provided by reserving resources in advance. So resource reservation is an integral part of resource management system for grid. Moreover the cost for providing resource as services will play a significant role in near future when resource sharing will be popular and inevitable. In this paper we provide a future reservation supported and cost optimized novel resource management system (CORS) for grid environment considering its real complex architecture. We further conduct a detailed performance evaluation with comparison on real workload traces for grid.*

Keywords: Grid Computing; Resource Reservation; Segment Tree; Parallel Workloads Archive

1. Introduction

Grid systems have emerged as promising next-generation computing platforms that enable the building of a wide range of collaborative problem-solving environments in industry, science and engineering [1]. The idea of grid computing was initially motivated by processing power and storage intensive applications. Its basic objective is to support resource sharing among individuals and institutions within a networked infrastructure. Resources that can be shared are processing capacity, storage, communication networks and bandwidth, data, software and licenses etc.

Managing various resources in highly dynamic grid environments is a complex and challenging problem. There exist works for resource management in different areas of computer science. Some approach uses data structures [2] [3] [4] and algorithms [5] for resource management to apply in grid but fails to provide any generalized solutions for grid environment. Most of the approaches are based on a simple architecture considering computer as the main resource in their system. But the real architecture of grid computing is a complex one if various resources of any computer are to be taken into account during resource management. And indeed as the use of grid as a computing environment increases at a higher rate, these complex but real scenario must be taken into account. In grids sometimes assurance is needed for successful completion of jobs on shared resources. Such guarantees can only be provided by reserving resources in advance [6] [7]. So resource reservation is an integral part of resource management system for grid. Moreover grids are used as a voluntary service now a days. But with the recent improvements in architecture and usage, situation is predicated not be the same. Cost for providing resource as services will play a significant role in near future when resource sharing will be popular and inevitable. Clearly a complete resource management system for grid computing is required to support all the above mentioned features. This gives the motivation of this work where the goal is to provide a future reservation supported and cost optimized novel resource management system for grid environment considering its real complex architecture.

The main contribution of this work is a distributed, future reservation supported and cost optimized resource management system (CORS) for grid environment. Most of the existing approaches are based on a simple architecture considering computer as the main resource. But the real architecture of grid computing is a complex one especially if we consider various resources of a computer (e.g., processor, memory etc.) during resource management. That means various resources of a single computer can be shared by many participators in grid. Our system does that which is also one the contribution of this work. We perform a detailed performance evaluation of our prototype and compare it with an existing system using real workload traces. Superiority of our scheme is established from the comparative analysis presented on the experimental results on the workload traces. The rest of the paper is organized as follows. Section 2 illustrates our proposed resource management scheme and the data structures used by this scheme. Section 3 contains the experimental results along with comparative study against the state of the art system. We briefly conclude in Section 4 describing the key contributions of this work followed by some future research directions.

2. Proposed Resource Management Scheme of CORS

We proposed a new resource management scheme for grid computing environment considering the complex real life grid architecture. In this section, a detailed description of the system architecture of our resource management scheme is presented with an illustrative example. The proposed data structure is also described with example. We start with the problem statement in the next subsection.

2.1 Optimization Problem for the Resource Management Scheme

Grid applications can be broken down into a number of jobs. It is the responsibility of the job broker to break down the jobs of the applications. Each job of an application requires some grid resources to perform their operations. The participating computing nodes of grid usually provide the required resources of any job at a particular cost. So each of the resources of any computing nodes will have cost associated with it.

Let there be n applications in the grid termed as $A_1, A_2 \cdots A_i \cdots A_n$. An application A_i has a total of C_{A_i} jobs. Assume that the jobs of application A_i are $J_1, J_2 \cdots J_j \cdots J_{C_{A_i}}$. The participating computing nodes of the grids are $N_1, N_2 \cdots N_k \cdots N_l$ and the resources provided by them are $R_1, R_2 \cdots R_r \cdots R_m$. We assume that all the participating nodes will provide all the grid resources according to availability. To make the problem description simple, let us consider that the job J_j of the application A_i requires W amount of the resource R_r. The available amount of the resource R_r in the nodes $N_1, N_2 \cdots N_l$ are $w_1, w_2 \cdots w_l$ and the corresponding costs are $c_1, c_2 \cdots c_l$. It is not always possible for a single computing node to completely serve a resource request. In most of the cases, a number of computing nodes jointly serve a resource request. The serving amount from the nodes are assumed as $s_1, s_2 \cdots s_l$. If a particular node N_σ does not serve the job then the serving amount $s_\sigma = 0$. Now the total cost to serve a resource request is the sum of all individual computing nodes' service cost for their resource. So the total cost of the request will be $\sum_{k=1}^{l} c_k s_k$. The constraints need to be satisfied are as follows:

1) $\sum_{k=1}^{l} s_i = W$, i.e., a particular job gets exactly W amount of resource from the grid.
2) $\sum_{k=1}^{l} w_k > W$, i.e., there is available resource in the grid for a job.

Now the objective is to minimize the total cost $\sum_{k=1}^{l} c_k s_k$ to serve a resource request for a job of an application. Besides the objective of minimizing the cost it is also expected to reduce the number of participating nodes to deliver resource for a particular job. This will reduce the bottleneck for remote communication to the participating nodes. This additional objective can be formulated as follows:

$$\text{minimize } \sum_{k=1}^{l} f(s_k) \text{ where}$$

$$f(s_k) = \begin{cases} 1 & \text{if } s_k > 0 \\ 0 & \text{if } s_k = 0 \end{cases}$$

Here $f(s_k)$ is a boolean function indicating the presence of a node in serving a job.

2.2 Resource Management Scheme

The overall system architecture of our proposed resource management scheme (CORS) is shown in Figure 1. The components of the system, messages and their sequences to run the system are described by the caption of the blocks of Figure 1. Our proposed resource management scheme consists of the following phases:

Start Phase: The resource management will be controlled and coordinated by a set of computing nodes (computer) termed as Principal Resource Manager (PRM). The PRMs will be selected according to the grid administrators decision based on the configuration of the participating nodes. The PRM will be given a list of resources by the administrators that can be provided by the participating nodes of the grid environment. Each resource will be given a unique id named $ResourceId$ for grid environment.

Initialization Phase: When a node wants to participate in the grid it will send a message named msg_init to any one of the $PRMs$. Each PRM has a list of participating computing nodes that will be synchronized amongst all the PRM. The PRM will accept the node and add it to the list. The node will be given a unique id named $NodeId$ that will help to identify it in the grid environment. Each participating node will have a list of resources to provide service to the grid environment. This list will be a subset of the list maintained by the PRM. The given $NodeId$ for any node is the same as the index of the node in the list maintained by PRM. Thus we can find the reference of any node through any of the PRMs in constant time. Each resource of a node will be given a $ResourceId$ which is also the same as the index of the resource in the node's resource list. Thus we can find the reference of any resource of a given node in constant time. Each node can be considered as its own resource manager (RM). Any participating computing node can be selected as PRM.

Request Phase: Any application on the grid can be broken down into a number of jobs. An application sends a message named msg_app to job broker so that job broker can break it down into a number of different jobs. The jobs usually request resources from the grid. The request will be initiated by the job broker. Job broker will forward the request to any one of the PRMs so that the request processing is distributed among the PRMs. The request mainly contains resource identifier, starting time, and ending time. The PRM will propagate the request to the all the participating nodes. . Any single job can issue request for multiple resources. Then the job broker can forward request

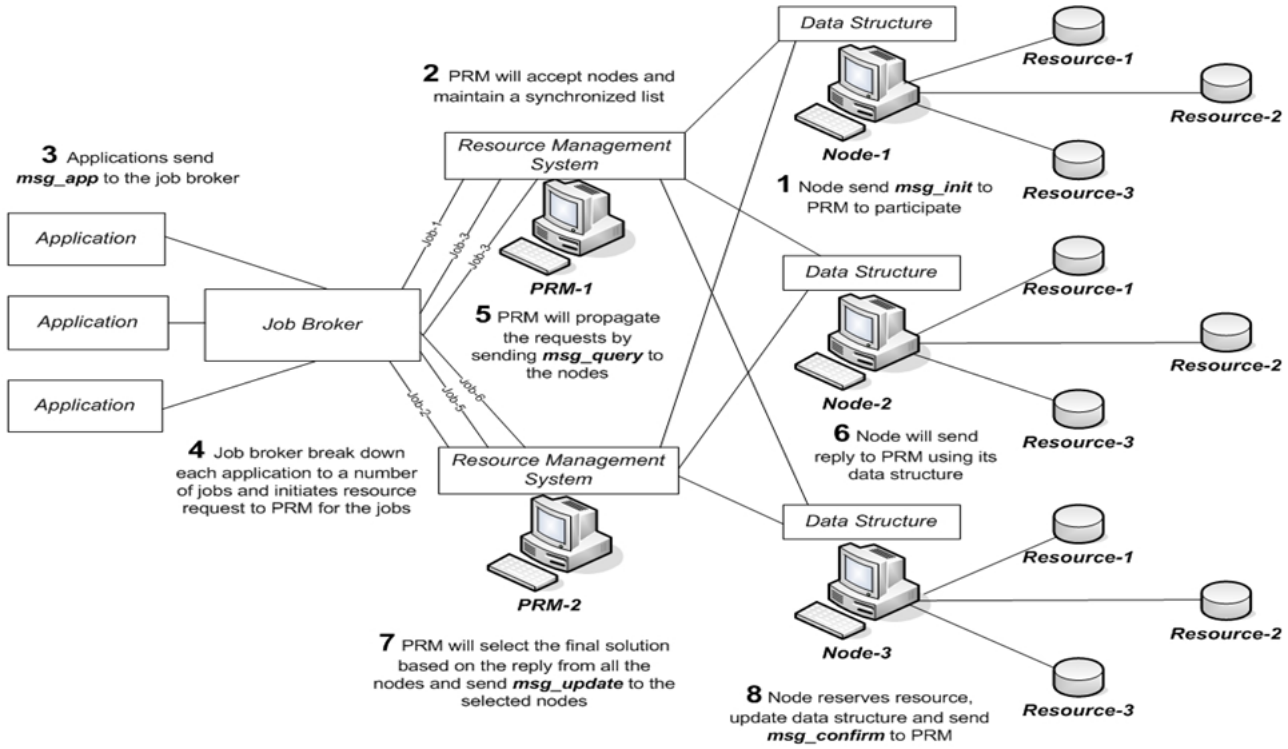

Fig. 1
ARCHITECTURE OF OUR PROPOSED SYSTEM CORS

for each resource to different PRM. So multiple PRMs can process request for a single job.

Search Phase: PRM will forward the request to each participating nodes by sending messages to the nodes. The messages sent to the nodes from PRM are generally named as msg_query. This is a parameterized message and based on the parameter a node replies with specific resource information, cost associated with a specific resource and available amount of specific resource in a given time frame. Some of the participating nodes may not provide the searched resource and they will be out of the search immediately. They will ignore the message. The nodes that provide the searched resource will receive the message. For each resource of each computing node, there will be an appropriate data structure to hold the information of used and available amount of resources in specific time frames. Then queries and corresponding updates will be carried out by the node in its own data structures. The data structure maintained by each participating node will not be replicated or copied to the PRM. The PRM will have only the reference of the nodes in the list and through that reference it can virtually have knowledge of the nodes' data structure. In this way multiple PRMs can have access to the most recent state of all of the resources without any space overhead. This is the main computation phase of our proposed resource management scheme.

Reply Phase: After searching, the results will be returned to the PRM by the nodes. Each request will contain a $request\_time$ associated with it. PRM will wait for the result for a specified threshold amount of time from the $request\_time$. Job broker will also wait for the replies from PRMs for a specified threshold amount of time from the $request\_time$ for the job that requires multiple resources. The result contains the notification whether the specific request can be served by this node or not. After getting the search result from all the nodes PRM will have a list of candidate nodes to serve the resource request. Now PRM needs to select the set of nodes that minimizes the total cost to serve the request. As we will show this problem can be mapped to well known fractional knapsack problem. The application of fractional knapsack problem in resource management is a novel idea for grid which we introduce here to guarantee cost minimization. Clearly the set of selected nodes will ultimately serve the request.

Reservation Phase: Once PRM has the list of selected nodes, it then sends a message named msg_update to each of the selected node to reserve required resource and update the data structure of the node. Upon receiving the message the node tries to update its data structure. If the update is successful then it will send a confirmation message named

$msg\_confirm$ to the PRM in reply. But sometimes updating may fail due to unavailability of resource as follows. The PRMs work in distributed manner. In the Search phase only available resource is searched but no update is made. So it may happen that another job acquires this specific resource of the node through any other PRM. That is why a confirmation ensures successful resource reservation.

For each resource of each computing node, there must be an appropriate data structure to hold the information of used and available amount of resources in specific time frames. Each element of the data structure will represent the (starting time, ending time, available amount) information for any resource. We know that the tree data structures are very much efficient for searching, inserting and deleting of elements. The segment tree structure, introduced by Bentley [8], is a balanced binary tree data structure that is used to store segments or intervals. We can map reservation supported resource management problem of our system to the segment tree with a little modification. The s and t, with $s < t$, of the segment tree $V(s,t)$ can be mapped into the starting time and ending time of a session where starting time $<$ ending time. We add a field (available resource amount of a segment) to each leaf node. So each leaf node in the segment tree will contain starting time, ending time and amount of specific resource available (between the starting time and ending time). Leaves of the segment tree contain all the segments and the available resource amount. The internal node of the tree contains only the interval of its child node. The space usage of segment tree is $O(nlogn)$ where n is the total number of nodes and searching for a specific interval requires $O(logn+k)$ time, where k is the number of reported segments. It does not depend on the number of intervals.

2.3 Fractional Knapsack Problem

In the fractional knapsack problem we are given a set I of n items having weights $w_1, w_2, \ldots w_n$ and costs $c_1, c_2, \ldots c_n$ respectively. We need to select items from I, with weight limit K, such that the resulting cost (value) is maximum. Most of the Knapsack variants are NP-Hard problems and the greedy solution to these problem leads to suboptimal or approximate solution but a greedy strategy does provide optimal solutions to the fractional knapsack problem [9]. We map the optimization problem for the resource management scheme to fractional knapsack problem. A resource of a node can be considered as an $item$ and its associated cost can be considered as $value$. We need to sort the resources of the nodes by increasing cost per unit resources as we need to minimize the total cost.

2.4 An Illustrative Example

Consider a grid environment where two Principal Resource Managers (PRM) are working named PRM_1 and PRM_2. The provided resource list is $R = \{R_1, R_2, R_3 \ldots R_n\}$. The participating node list is $N = \{N_1, N_2, N_3 \ldots N_n\}$. Figure 2 depicts the overall scenario of the system. Here we can see the resource provided by a specific node N_1. For each resource of N_1 there is a segment tree to maintain the available amount of resources in a specific time frame.

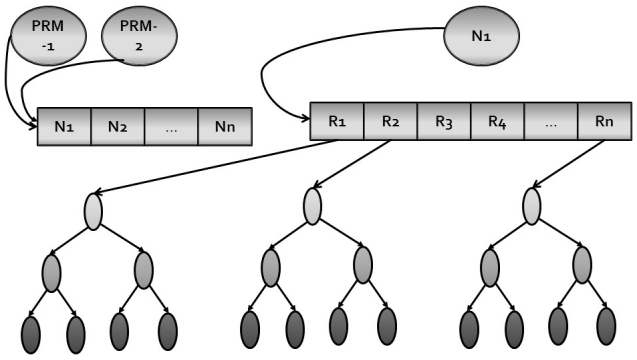

Fig. 2
OVERALL SYSTEM SCENARIO OF CORS

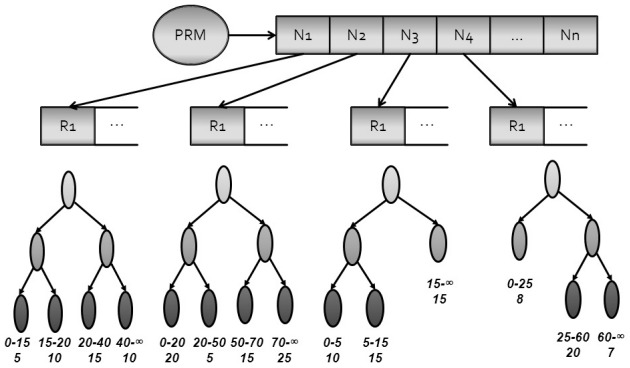

Fig. 3
NODE AND ITS DATA STRUCTURE

Let us consider that Application A_1 contains three jobs termed J_1, J_2 and J_3. Now J_1 requires 25 units of resource R_1 for the time frame of $(25, 45)$. At this point J_1 will send the request to the PRMs. Recall that PRM have a synchronized list of participating nodes. It will forward the query to the participating nodes. Figure 3 depicts the scenario of the nodes and corresponding resource R_1. Here separate time intervals and corresponding available amount of resource is shown with the leaf nodes. As can be seen that Node N_1, N_2, N_3 and N_4 are providing resource R_1. The corresponding data structure is also shown in the figure. Here the query will be $(R_1, 25, 45)$. Assume that the query is passed to each node data structure, the reply is listed in Table 1. After the search is completed, PRM will receive the above candidate list to serve the request for R_1. Subsequently the nodes are selected according to the fractional knapsack solution, as

Table 1
CANDIDATE NODES AND AVAILABLE AMOUNT

Node	Amount	Details	Cost
N_1	10	minimum amount of time frame (20, 40) and (40, ∞)	4.0
N_2	5	amount of time frame (20, 50)	4.25
N_3	15	amount of time frame (15, ∞)	3.75
N_4	20	amount of time frame (25, 60)	3.5

Table 2
SELECTED NODES FROM THE CANDIDATE LIST

Node	Amount	Cost
N_4	20	20 $\times$ 3.5 = 70
N_3	5	5 $\times$ 3.75 = 18.75
	Minimum Total Cost	88.75

shown in Table 2. So an add request will be sent to N_3 and N_4 and their corresponding segment tree will be updated as shown in Figure 4. The updated resource usages are shown in black shades. This concludes the resource reservation for the job. The application will then start running according to its starting time using these reserved resources. The details of the algorithms related to our proposed resource management scheme CORS and their complexity is not provided here due to page limitation. Their details can be found here [10].

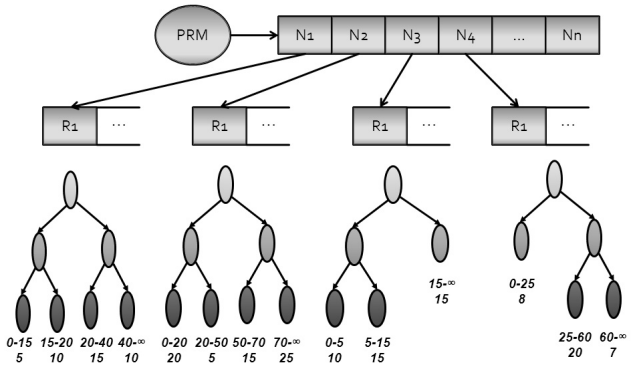

Fig. 4
NODE AND ITS DATA STRUCTURE

3. Experimental Results

In this section we present the experimental results of our proposed system.Through experiments we study the behaviour of our approach and evaluate its performance based on some performance metrics and also compare the performance of our approach to an existing system. A simulator of the system is implemented using Java. The simulation is run using a computer having Intel Pentium-IV Dual Core 1.60GHz processor, 2 GB of memory and Windows XP operating system. We need to implement a new simulator because no existing simulator considers the complex grid architecture. In the existing simulators computer is considered as the only resource but in our system we need to consider various hardware and software of a computer as resources.

Node Selection Rules: The following rules are considered for assigning priority in selecting the next node to serve the request.

- *Max-Res*: This rule prioritizes the nodes that have maximum available resources. In this way number of connection establishment can be reduced.
- *Min-Res*: This rule prioritizes the nodes that have minimum available resources. In this way number of connection establishment can be increased.
- *Min-Cost*: This rule prioritizes the nodes that leads to the minimization of total cost.

Measurement Metrics: The metrics considered for evaluation are *TotalConnection* and *TotalCost*. We have also considered total memory consumption and running time.
TotalConnection: The term *TotalConnection* means the number of nodes required to completely serve a request. The requesting node needs to connect to these nodes. That is why we termed it as *TotalConnection*.
TotalCost: The term *TotalCost* means the total cost required to completely serve a request. This is the summation of all the individual cost of different nodes that serves the request.

Comparison with Sulistio's Resource Management Scheme on Real Workloads: We compare our system CORS with an existing system for resource management in grid computing. The work done by Sulistio et al. [11] is the most appropriate to compare because it is the most recent work on resource reservation for grid. This work provides a new data structure for reservation using the Calendar Queue. There is no cost based framework exists for resource reservation in grid and this is also true for Sulistio's system. So we need to assume a default cost model. We developed Sulistio's system to minimize the *TotalConnection* by giving priority to the next device to be selected according to the rule Max-Res described before. It is guaranteed that Sulistio's system's *TotalConnection* will be minimized. On the other hand in our system we have been successful to achieve minimum cost solution to serve a grid request at the cost of a very small or no increase of *TotalConnection* from the minimum. Here we present results using real workload data for grid. Parallel Workloads Archive [12] contains an archive of information regarding the workloads on parallel machines and grids. It contains raw workload logs from various machines around the world. We choose three workloads, namely DAS2-fs0, LPC-EGEE and SDSC-BLUE. Details of the workloads are available at [12]. The main reason behind choosing

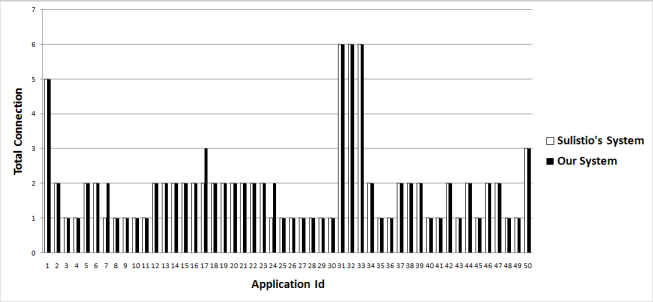

Fig. 5
TOTALCONNECTION REQUIRED FOR WORKLOAD DAS2FS0 USING SULISTIO'S SYSTEM AND OUR PROPOSED SYSTEM CORS

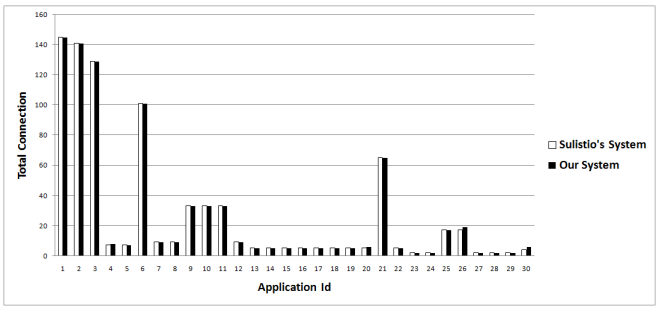

Fig. 7
TOTALCONNECTION REQUIRED FOR WORKLOAD SDSC-BLUE USING SULISTIO'S SYSTEM AND OUR PROPOSED SYSTEM CORS

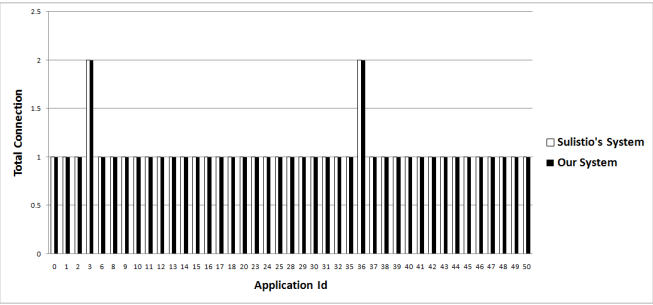

Fig. 6
TOTALCONNECTION REQUIRED FOR WORKLOAD LPC-EGEE USING SULISTIO'S SYSTEM AND OUR PROPOSED SYSTEM CORS

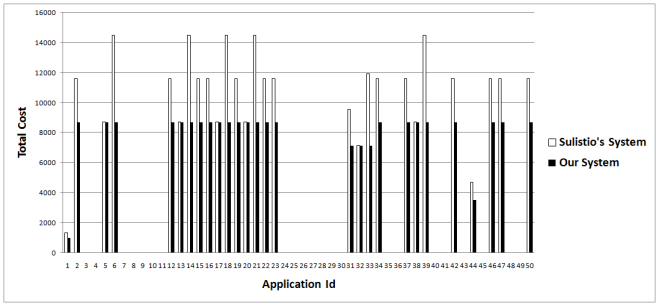

Fig. 8
TOTALCOST REQUIRED FOR WORKLOAD DAS2FS0 USING SULISTIO'S SYSTEM AND OUR PROPOSED SYSTEM CORS

these three is that these workloads have been considered by Sulistio in their simulation.

Evaluation with Respect to TotalCost and TotalConnection: We simulate our proposed system CORS and Sulistio's system using the above mentioned three workloads considering 50 sample applications.

In Figure 5 to Figure 7 we observe that the *TotalConnection* of Sulistio's system and our proposed system are equal for most of the applications. There are differences in *TotalConnection* for a few applications [3 applications in Figure 5 and 4 applications in Figure 7]. Here difference occurs for those applications whose jobs require huge amount of resources compared to the other applications of the same grid environment. We observe the presence of larger connection in Figure 7 compared to the other figures. It is also observed that in Figure 7 there are applications with various connection requirements. This justifies the high capacity of SDSC-BLUE and its multipurpose use by high, medium and low profile users in terms of resource requirement. In Figure 8 to Figure 10 we observe that the *TotalCost* of Sulistio's system is much greater than our proposed system as we expected.

Analysis of the Result: *TotalCost* of our system is guaranteed to be minimum as we use fractional knapsack to minimize the total cost. But the interesting point is the margin of difference with the cost of Sulistio's system. The *TotalCost* of our proposed system is much less than Sulistio's system for all the workloads. *TotalConnection* of our system will not be minimum because we consider minimizing the *TotalCost*. But we tried to maintain *TotalConnection* as small as possible so that the increasing *TotalConnection* does not be a bottleneck. It is observed from the presented charts that we achieve the goal to maintain the difference as minimum as possible. For almost all the workloads *TotalConnection* for Sulistio's system and our proposed system are the same. This is because the nodes that provide the resources in a grid environment are mostly of same configurations and the jobs of the applications in a grid normally requires similar amount of resources. Grid applications are normally broken down into similar type of jobs by the job broker so that the application gets fair share of the resources. The details of how the job broker works is out of the scope of this research. However to justify the *TotalConnection* scenarios we briefly review it. In the grid environments most of the applications are similar in nature. So the job broker usually breaks down all the applications to same types of jobs where each job requires similar amount of resources. In such cases *TotalConnection*

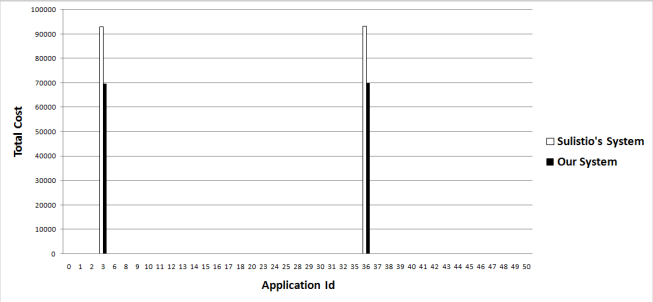

Fig. 9
TOTALCOST REQUIRED FOR WORKLOAD LPC-EGEE USING SULISTIO'S SYSTEM AND OUR PROPOSED SYSTEM CORS

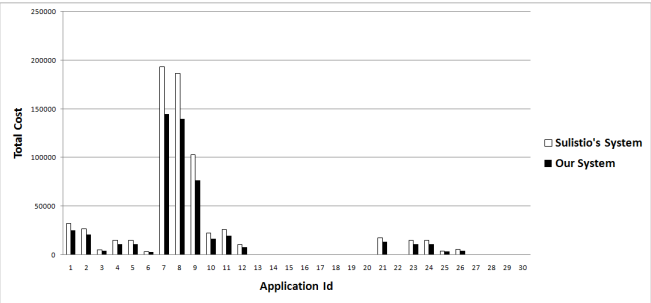

Fig. 10
TOTALCOST REQUIRED FOR WORKLOAD SDSC-BLUE USING SULISTIO'S SYSTEM AND OUR PROPOSED SYSTEM CORS

for Sulistio's system and our proposed system are the same. Sometimes there are exceptions in the workloads where a big sized application requiring huge amount of resource is broken down into a single job. This might happen due to the constraint that the application cannot to broken down into small jobs. Now consider a scenario where a particular node with the comparatively higher cost has the highest available resource. According to our proposed algorithm this particular node will not be chosen for consuming all the resources. But the Sulistio's algorithm will consume this resource to maintain *TotalConnection* minimum. That is why we observe substantial difference in *TotalConnection* in several exceptional cases. But generally it is observed that this difference is negligible. The memory consumption and running time of our proposed system CORS is also better than Sulistio's system. That means CORS memory consumption is lower and running time is less than that of Sulistio's system. The main reason behind this is the use of appropriate data structures and efficient algorithms in our proposed system. Due to page limitation the details of memory consumption and running time are not provided here. The details can be found here [10].

4. Conclusion

The main contribution of this work is a cost optimized complete resource management system with reservation support for grid computing. Resource management is not a new research area for grid computing but still there are lot of challenges and unsolved problems. Managing resources with negotiation is one of the open issues in grid resource management. Sometimes effective negotiation for flexible quality of service (QoS) can ensure more accepted jobs in grid system with full resource utilization. We have introduced a cost optimization model for resource management in grid computing. Future works can be done here to incorporate negotiation for cost between resource provider and resource requester (applications or jobs). The computing nodes that provide resource for grid also run local applications in their own operating environment. Works can be done how to optimally balance the distribution of resources for local and grid applications so that the local applications can not be affected by its services provided to the grid. Future works can also be done on resource management by considering the topology of the grid. In that case communication bandwidth requirement and latency will affect the resource management techniques.

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[2] L.-O. Burchard, "Analysis of data structures for admission control of advance reservation requests," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 413–424, 2005.

[3] A. Brodnik and A. Nilsson, "Static data structure for discrete advance bandwidth reservations on the internet," *Computer Research Repository(CoRR)*, vol. cs.DS/0308041, 2003.

[4] Q. Xiong, C. Wu, J. Xing, L. Wu, and H. Zhang, "A linked-list data structure for advance reservation admission control," in *In Proceedings of 3rd International Conference on Networking and Mobile Computing(ICCNMC)*, 2005, pp. 901–910.

[5] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *In Proceedings of the International Workshop on Quality of Service*, 1999, pp. 27–36.

[6] W. Smith, I. Foster, and V. Taylor, "Scheduling with advanced reservations," in *In Proceedings of IEEE International Parallel and Distributed Processing Symposium(IPDPS)Š00*, 2000, pp. 127–132.

[7] L. Yuan, C.-K. Tham, and A. L. Ananda, "A probing approach for effective distributed resource reservation," in *QoS-IP 2003: Proceedings of the Second International Workshop on Quality of Service in Multiservice IP Networks*. London, UK: Springer-Verlag, 2003, pp. 672–688.

[8] J. Bentley, "Solution to klee's rectangle problems," *Techical Report, Carnegie-Mellon University, Pittsburgh*, 1975.

[9] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[10] R. Shahriyar, "http://teacher.buet.ac.bd/rifat/MSC.pdf," *A Distributed Optimized Resource Reservation Scheme for Grid Computing, M.Sc. Engg. Thesis, Bangladesh University of Engineering and Technology*, 2010.

[11] A. Sulistio, U. Cibej, S. K. Prasad, and R. Buyya, "Garq: An efficient scheduling data structure for advance reservations of grid resources," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 24, no. 1, pp. 1–19, 2009.

[12] P. W. Archive, "http://www.cs.huji.ac.il/labs/parallel/workload."