Grid Scheduling with buddy-based resource discovery

Mahmoud Naghibzadeh
Dept. of Computer Engineering
Ferdowsi University
Mashhad, Iran
naghibzadeh@um.ac.ir

Javad Hamidzadeh
Dept. of Computer Engineering
Ferdowsi University
Mashhad, Iran
Ja_ha47@stu-mail.um.ac.ir

Saied Abrishami
Dept. of Computer Engineering
Ferdowsi University
Mashhad, Iran
s-abrishami@um.ac.ir

Abdolreza Savadi
Dept. of Computer Engineering
Ferdowsi University
Mashhad, Iran
savadi@ferdowsi.um.ac.ir

***Abstract*—Vast majority of computer users are generously willing to share their computer resources with needy organizations and individuals to solve their computational, data storage, and communicational problems. The recently emerging Grid technology is providing the required platform for the coordinated resource sharing and problem solving among individual computer users as well as dynamic multi-institutional virtual organizations. Resource discovery is a preliminary step towards distribution of work load to resources in order to reach the required quality of service, for example, to minimize the completion time of tasks. In this research we have introduced a buddy-based resource discovery technique. Two, or more devices, are buddies if one can be used in place of the other, whenever the former is unavailable or it is so busy that by waiting for it the required quality of service will not be achieved. The simulation results of the proposed technique showed that buddy-based resource discovery is a promising approach.**

I. INTRODUCTION

Vast majority of computer users are generously willing to share their computer resources with needy organizations and individuals to solve their computational and communicational problems. It is up to the field's specialists to make this sharing pleasant, safe, efficient, and economical to both parties. The recently emerging Grid technology is providing the required platform for the coordinated resource sharing and problem solving among individual computer users as well as dynamic multi-institutional virtual organizations.

Local resources are usually preferred over nonlocal ones with similar capabilities and performance. When the resources to be shared are processing power, there are two general strategies to think about, load balancing and load sharing.

In load balancing strategy, requested tasks are distributed amongst all existing resources such that the all resources' load are equal. An ideal load balancing algorithm thus distributes the requests into existing resources so that all the resources finish their tasks simultaneously [1].

The goal of load sharing strategy is to prevent the situations in which some resources are idle where the other some other resources have a queue of waiting tasks. It tries to reduce the load on the heavily loaded resources, only. The strategy doesn't seek to make sure all resources are equally-loaded [2].

Both, load balancing and load sharing strategies, have been extensively studied and it has been shown that their usage greatly improves the average completion time of set of tasks submitted to overall system being either a distributed system or a Grid infrastructure. However, the performance of these strategies is not necessarily the same and one may outperform the other, given the circumstances.

Load sharing algorithms are classified into static and dynamic. Dynamic algorithms have the potential to outperform static algorithms by using the current system information for task allocation decisions. Different approaches to dynamic load sharing are based on sender-initiation, receiver-initiation and domain based load sharing [1, 2]. Most of the new advanced in load sharing algorithms are based on decentralization hierarchy. In [4], a dynamic load sharing scheme called Dual Layered Load

Sharing (DLLS) is proposed. It accomplishes its goal by using two levels of load sharing. At the first level, load sharing is done on neighboring nodes, and at the second level, the sharing is done between different neighborhoods. In [5] a size-based scheme called it Least Flow-Time First (LFF-SIZE), is proposed in which a multi-section queue is used to separate larger tasks from smaller ones.

No matter what the load sharing/load balancing technique is, resource discovery is a preliminary step towards this purpose. An efficient resource discovery technique significantly improves the performance of any load balancing or load sharing technique.

In this paper we are proposing a buddy-based resource discovery. Section II describes the concept of buddy system and its implication to resource discovery. It provides two algorithms, buddy-building and resource discovery. In Section III, the proposed resource assignment is simulated using the Gridsim simulator. Section IV summarizes and suggests some immediate future works.

II. BUDDY-BASED RESOURCE DISCOVERY

Two, or more devices, are buddies if one can be used in place of the other, whenever the former is unavailable or it is so busy that by waiting for it the required quality of service is (probably) not achievable. The concept of buddy has been used in many fields, especially in the field of operating systems. One example is in assigning contiguous memory partitions to processes. In this method, the size of memory partitions is a power of two kilobytes. The memory allocator may split a partition into two equal size contiguous partitions called buddies. If necessary, each new partition could be further split into two smaller size buddies. A partition can be assigned to a requester as a whole. When a partition is no longer needed and it is freed, it is reunited with its buddy, if the buddy is also free, to form a larger partition. Two partition of the same size can be merged into one partition if the partitions are buddies. The process of reuniting buddies is continued as far as it is possible. In this context, buddies have exactly the same properties and each one can equally be assigned to the memory requesting process. It is this similarity which will be of our interest in developing computer buddies.

One or more buddies could be defined for every resource for which there is a possibility that it might not be up and running all the time or, in some circumstances, it might be in a high demand so that not all competing processes for the resource may perform timely or accurately. Buddies of a device must be selected so that, besides having the functional capabilities of the device itself, everyone has a high availability when needed. Although the ultimate goal of this research is to use the concept of buddy in Grid systems, we are considering a simplified scenario for the time being. A resource under investigation is a personal computer with standard units like CPU (or PUs), cache memory, main memory, secondary storage, etc. It might have other complementing units like tape CD drives, as well. A body of a PC is, hence, another PC with similar facilities. All computers are connected together via a network interconnection hardware and protocol. The whole structure should resemble a simplified Grid structure where nodes are all independent personal computers with different capabilities and, theoretically, spread over the globe. In this research we decided to define two buddies for every computer. Figure 1 shows the model of this organization.

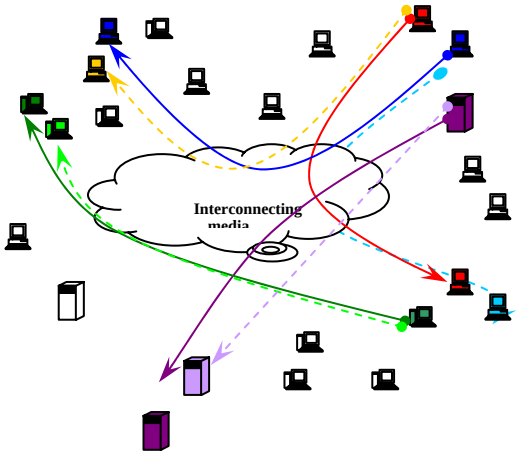

Figure 1: A simple model of a grid with buddies being connected

It is the case that the buddies of a system are not selected randomly, but rather it is most important to carefully agree upon the required criteria to guarantee a high probability of the availability of a buddy whenever it is needed. Buddy designation criterion should be so developed to match the overall goals of the system being designed.

For simplicity, here, we consider the case where every process runs a sequential program, rather than a parallel one. The required quality of service is defined around the expected turnaround time of the process which runs the program. The buddy building criteria are considered to be the degree of the availability of the buddy and the degree of the matching of the properties of the buddy and the resource. For every resource we designate two buddies with the first one being preferred over the second. Figure 2 shows the buddy building algorithm. This algorithm has to be executed by every computer whenever it decides to find his buddies. It is usually executed once when the computer is connected, or reconnected, to the Grid. Besides, it could be executed whenever the availability of a computer's buddies are very low.

In buddy-building algorithm, for each resource R_i, i=1, 2, … n, a vector S_i (s_{i0} , s_{i1}, $s_{i2,}$ …, s_{ij}) represents its

attributes. For example, if the resource is a CPU then the attributes could be CPU power in instruction per second, cache size etc. Besides, another vector A_i (a_{i0}, a_{i1}, $a_{i2, \ldots}$, a_{i23}) represents the availability of the resource in each 24-hour time intervals of a 24-hour day. Therefore, we have assumed that the availability of a resource is fixed for each whole hour of a 24-hour. In addition, since the A-vector has only 24 components the availability of a resource is the same for all days of a year. These restrictions may be relaxed to better match with the real world's situation. This relaxation will be done in our future research as the system is completed and the test phase is successful. One more vector called MinReq represents the minimum requirements needed for a resource to be considered as a buddy of this resource. The proposed buddy-building method is formalized in Figure 2.

Buddy-building()

1- Compute R_0 (S_0, A_0) and MinReq for the resource

2- Obtain R_i (S_i, A_i), i=1,2, …, n-1, for all other same-type resources in the grid from the Grid Information System (GIS)

3- for i=1 to n-1 do

3-1 if (R_i< MinRequire) then continue with the next i

3-2 Compute the pair wise difference between specifications of R_0 and R_i as:

$$SpecDiff_i = \sum_j c_j \left(\left| s_{ij} - s_{0j} \right| \right)$$

where c_j is the importance level of the jth specification of the resource.

3-3 Compute the difference between availability of R_0 and R_i as:

$$AvailDiff_i = \sum_j (a_{ij} - a_{0j}), if\ a_{ij} > a_{0j}$$

3-4 Compute the similarity between R_0 and R_i as:

$$AvaDiff_i = \frac{AvailDiff_i}{SpecDiff_i}$$

4- Select two resources that have the highest availability_difference, AvaDiff, as the two buddies of this resource.

Figure 2: A high-level algorithm for building buddies

Whenever a request is received, if the required quality of service is not obtainable by the local resource, the first buddy is checked, first. This resource is selected if the required quality of service is attainable. Otherwise, the second buddy is checked. If the required quality of service is not attainable by the second buddy, we then use a different technique for selecting a resource to do the task. The overall resource discovery algorithm is summarized in Figure 3.

Resource-discovery ()

1- Compute LocalExpectedTurnaroundTime for the current task on the local machine

2- if (LocalExpectedTurnaroundTime <= DesiredTurnaroundTime) then

assign this task to local machine and exit

3- Compute Buddy1ExpectedTurnaroundTime for the current task on the first buddy

4- if (Buddy1ExpectedTurnaroundTime <= DesiredTurnaroundTime) then

assign this task to the first buddy and exit

5- Compute Buddy2ExpectedTurnaroundTime for the current task on the second buddy

6- if (Buddy2ExpectedTurnaroundTime <= DesiredTurnaroundTime) then

assign this task to the second buddy and exit

Figure 3: The proposed resource discovery algorithm

III. SIMULATION RESULTS

The novel proposal of resource discovery which is presented here is simulated to make sure it is worth being considered for further research and development. Twenty four time zones are considered around the globe. For each time zone five computers with random properties (operating system, CPU speed, cache memory capacity, and main memory capacity) are generated. Each computer is considered to have the capability of process migration (from or to).

We used the GridSim software (from the GridBus Project [6]) for performing simulations. Simulations are done for four cases. In the first case, we used Poisson distribution with mean equal to 1500 million instructions for the tasks length. Each task has a deadline equals to its execution time on a 100 MIPS computer. In this case, arrival rate also had a Poisson distribution but its mean is

considered to be variable. Figure 4 shows the simulation results. The horizontal axis represents the arrival rate mean. As it can be seen, the percentage of tasks assigned to the local processor decreases as the arrival rate mean increases. On the other hand, as the arrival rate mean increases the percentage of tasks being assigned to buddies increases. At some point, about 0.04 arrival rate mean, the increasing nature of the assignment to the first buddy has stopped and the assignment if mostly forwarded to the second buddy. At some other point, about 0.08 arrival rate mean, there in no increase in the percentages of tasks assigned to either buddies. From here on, the percentage of tasks assigned to either buddies does not show any increase. Therefore, the percentage of tasks being rejected starts to increase in a higher rate. This result matches very well with what is expected.

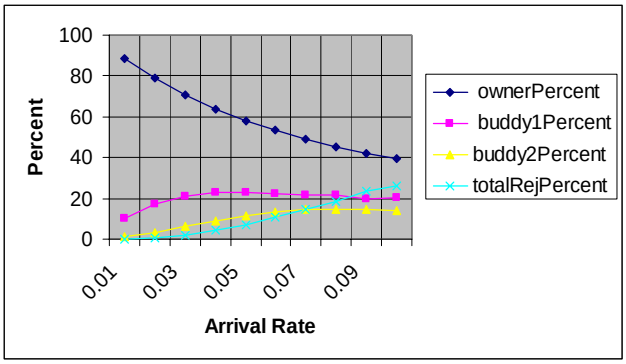

Figure 4: The percentage of assigned task for fixed task length and variable arrival rate

In the second case, arrival rate's mean time is fixed on 0.05 and the tasks length mean is considered as variable. Figure 5 shows the result. The results shows that with the increase in tasks length, the percentage of tasks assigned to the local processor decreased and this percentage increased for the buddies.

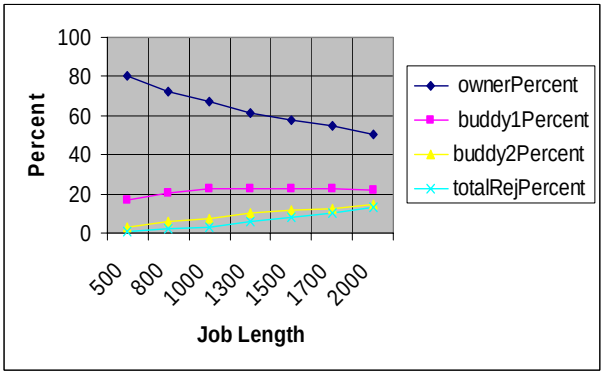

Figure 5: The percentage of assigned task for fixed arrival rate and variable task length

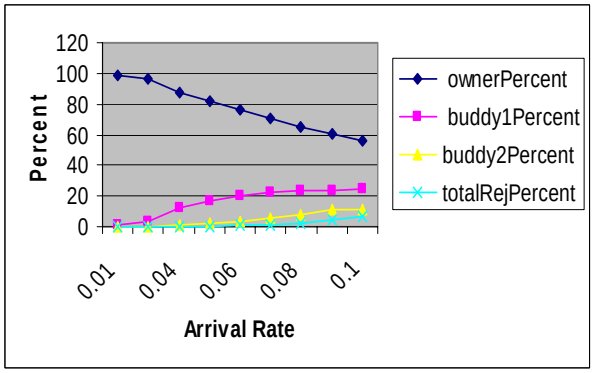

Figure 6: The percentage of assigned task for fixed task length and variable arrival rate with twice deadline time as Figure 4

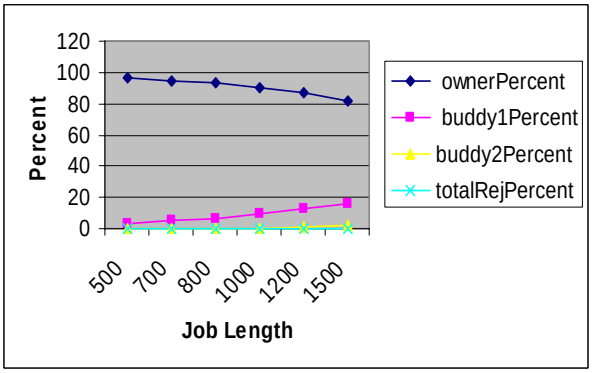

Figure 7: The percentage of assigned task for fixed arrival rate and variable variant task length with twice deadline time as Figure 5

Case 3 and 4 are analogous to case 1 and 2, respectively, except that the deadline is twice as the Case 1 and Case 2, respectively. The simulation results show that the second buddy has a lower significant in these two cases. See Figures 6 and 7.

The results show the reasonably high availability of buddies whenever a resource needs assistant from other resources. The low rejection rate signals that the need to use complementary resource discovery is low. The communication overhead used to discover a required resource is very low compared to blind search resource discovery algorithms.

IV. SUMMARY AND FUTURE WORKS

Within the Grid infrastructure resource discovery is a major step towards distribution of work load to resources in order to reach the required quality of service. We proposed a buddy-based resource discovery technique. Whenever a resource is unavailable, its buddy is a good choice to perform the required task. Two algorithms are developed for buddy-building and resource discovery. The proposed resource discovery based on buddies is

simulation and the results are presented. In the simulation the globe is divided into 24 time zones and for each time zone a small number of computers, i.e., five, is considered. It was concluded that buddy-based resource discovery is a promising approach. It is well understood that the actual number of computers in all time zones are not the same. A separate investigation is required to estimate these numbers and redo all the simulations with respect to this new information. Due to unavailability of high performance computes, in our experiments, the total number of nodes were 120. It is desirable to do the simulation for systems with higher number of computers.

REFERENCES

[1] B. Veeravalli, and W. Han Min, "Scheduling Divisible Loads on Heterogeneous Linear Dasiy Chain Networks with Arbitrary Processor Release Times," IEEE Transactions on Parallel and Distributed Systems, Vol. 15(3), March 2004.

[2] O. Krem, and J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms," IEEE Transactions on High Performance Computing, Vol. 4(3), 1992.

[3] G. Cybenko, " Dynamic Load Balancing for Distributed Memory Multiprocessors," Journal of Parallel and Distributed Computing, Vol. 7, pp. 279-301, 1989.

[4] G. Hura, S. Mohan, and T. H. Srikanthan, "On Load Sharing in Distributed Systems: A Novel Approach," Tranactions SDPS, Vol. 6, No. 1, pp. 59-81, 2002.

[5] G. Tari, J. Broberg, A.Y. Zomaya, and R. Baldoni, "A Least Flow-Time First Load Sharing Approach for Distributed Server Farm," J. of Parallel and Distributed Computing, Vol. 65, pp. 832 – 842, 2005.

[6] http://www.gridbus.org/gridsim/