The CSI Journal on
Computer Science and Engineering
Vol. 4, No. 2&4, 2006
Pages 17- 24
Regular Paper

Grid-JQA: A New Method for Matching in Grid Environment

Leila Mohammad Khanli [1], Morteza Analoui [2]

[1] Department of Computer Science, Tabriz University, Tabriz, Iran
[2] Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

Abstract

The Grid is an emerging technology for enabling resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations. This paper presents algorithms, methods, and software for a Grid resource manager, responsible for resource brokering and scheduling in Grids. The broker selects computing resources based on actual job requirements and a number of criteria identifying the available resources, with the aim to minimize the turnaround time for the individual application. In pervious work, we proposed Grid-JQA [8] [9]. In this work we propose an aggregation formula for the QoS parameters. The formula is a unit less combination of the parameters together with weighting factors. It is shown that the formula needs to put into a threshold consideration. A discussion on the threshold and its level is also provided. The paper is finalized by the results obtained from simulation and a comparison study.

Keywords: Distributed resource management, Grid-JQA, heterogeneous computing, Quality of Service, Matching.

1. Introduction

THE development of grid computing technologies [1] over the past several years has provided us a means of using and sharing heterogeneous resources over local/wide area networks, and geographically dispersed locations. This has resulted in the ability to form loosely coupled, high-performance computational environment comprising numerous scalable, fault tolerant, and platform-independent services across the entire Internet. The grid infrastructure provides a way to execute applications over autonomous, distributed and heterogeneous nodes by secure resource sharing among individuals and institutions. Typically, a user can submit jobs to a grid without necessarily knowing (or even caring) where it will be executed. It is the responsibility of the grid resource management system to distribute such jobs among a heterogeneous pool of servers, trying to optimize the resource usage and provide the best possible quality of service.

The Grid Java based Quality of service management by Active database (Grid-JQA) is a framework that aims to address the above mentioned requirements [8] [9]. Grid-JQA provides management for quality of service on different types of resources, including networks, CPUs, and disks. It also encourages Grid customers to specify their quality of service needs based on their actual requirements. The main goal of this system is to provide seamless access to users for submitting jobs to a pool of heterogeneous resources, and at the same time, dynamically scheduling in multi policy mode and monitoring the resource requirements for execution of applications. We proposed an optimal solution but not practical for comparing with other practical solution.

Active database systems support mechanisms that enable them to respond automatically to events that are taking place either inside or outside the database system itself. In the areas of parallel and Grid computing, most of resource management and application scheduling issues use traditional database management systems but until now any of them use active database. Traditionally, database systems have been viewed as repositories that store the information required by a grid resource manager, and that are accessed

scheduler. As, desktop grid resources are inherently volatile, so using active database rules on resource management and scheduling is the best idea.

Using ECA rules is our main contribution. There are several advantages in using ECA rules to implement an application's reactive functionality, rather than encoding it in application code. Firstly, ECA rules allow this functionality to be specified and managed within a rule base rather than being dispersed in diverse programs, thus enhancing the modularity, maintainability and extensibility of applications. Secondly, ECA rules have a high-level, declarative syntax and are thus amenable to analysis and optimization techniques which cannot be applied if the same functionality is expressed directly in application code. Thirdly, ECA rules are a generic mechanism that can abstract a wide variety of reactive behaviors, in contrast to application code that is typically specialized to a particular kind of reactive scenario.

The rest of the paper is structured as follows: Related work is presented in Section 2, in section 3 we introduce three methods for matching in the grid environment, we discuss a proof-of-concept implementation of a prototype and the preliminary performance results by simulation in Section 4, whereas Section 5 concludes with a summary of our work.

2. Related Work

Legion [2] takes an object-oriented approach to resource management, formulating the matching problem as an *object placement problem* [2]. The identification of a candidate resource is performed by an object mapper, whose recommendation is then implemented by a different object. The Legion system defines a notation [2] that is similar to classads, although it uses an object oriented type system with inheritance to define resources , as contrasted with the simple attribute-oriented Boolean logic of classads. Legion supports autonomy with a jurisdiction magistrate (JM), which may reject requests if the offered requests do not match the policy of the site being managed by the JM. While the JM gives a resource veto power, there is no way for a resource to describe those requests that it would rather serve.

UDDI [3] and eSpeak [4] are two specifications being defined to enable automation of business-to-business interactions. Both systems use XML as a specification language to describe services, and define a rich framework for service discovery. Like most other systems, neither UDDI nor eSpeak exports a symmetric interface to servers and customers. Furthermore, since the emphasis in these frameworks is on service discovery and not resource allocation, the matchmaker provides a list of candidate servers to the customer, who then chooses one or more servers based on subjective criteria.

The Condor [5] equivalent of an information system such as UDDI is a matchmaker that maintains a pool of ClassAds representing candidate resources and then matches each incoming request ClassAd with all candidates, returning a highest-ranked matching candidate. Redline [6] formulates the matching problem as a constraint satisfaction problem. These latter systems allow expression of resource groups, but they do not offer a concise method to express network topologies. Set-matching extends the ClassAds language to provide a multilateral matchmaking mechanism where the number of resources is not known a priory [7]. However, the set matching mechanism is not capable of marshaling a heterogeneous mix of resources.

Our work is similar to Condor. That is the resources advertise themselves to the broker and the users send their requests to the broker. Then the broker matches the resources and the tasks by using some policies. The differences between our work and Condor are after this time. Because, after matching the broker does not interfere in Condor. And the user works with resources directly. But in our work, the broker manages the activities till the results return to the user. The disadvantage of managing broker till the end is that the broker becomes too busy. For correcting this disadvantage, we use hierarchical broker, in a way that when the broker becomes too busy, one new broker is introduced in the low level and busy broker can send new application for execution to lower level broker. So the problem is solved by hierarchical broker.

The advantages of the broker management till end are:

1) For guarantying the QoS parameters, the broker should monitor till the results return to the user.

2) May be all requested resources are not free, so application can not start execution, but in our work as soon as one resource is free, the execution of user's task can be started. Gradually, the remained tasks can be executed when the other resources are released.

3) Adaptation is possible only in this method that the broker monitors till the end. As duplicate execution of strong tasks which are being executed in weak resources.

4) By using the broker, the user can be thin client that is the users can execute their application by PDA or Mobile handset considering the user want the power of processing like supercomputer. Because the broker manages for matching, monitoring, sending task, receiving results, and managing failed tasks, the user only sends the request and waits until it receives the results.

5) Managing failed tasks is simpler than the other methods. Because the broker can assign more priority to failed tasks and matches them as soon as possible.

6) Fault tolerance is possible. The broker can execute the tasks with more priority in duplicate mode.

3. Matching

One of the most problem in grid environment is the matching. Following model explains the problem:

1) The problem input
   1.1) A set of resources with their capabilities
   1.2) A set of tasks with their requirements
2) The problem output: Matching the best resource for each task
3) The problem purpose: Minimizing turnaround time

There are two managing models: 1) Local 2) Global. In Local model, the manager finds the best resource for each task locally regardless of other tasks. Whereas in Global model, the manager finds the proper resource for each task in relation with other tasks and the strong resource is assigned to strong task. (The task is strong / weak if it's requirement is high / low.) Whereas in local model the manager may assign the strongest resource to first task which may be the weakest

one. Therefore for the next task which may happen to be a strong task, the strong resource doesn't exist anymore. So it is better to apply the matching management in a global model.

We propose three methods for matching. They are compared using simulation in the next section. Three methods are:

1) *Optimum*
2) *Grid-JQA*
3) *Dup Grid-JQA*

We can consider:

- n: the task number
- m: the resource number
- k: the number of QoS parameters

As it was explained, the resources should introduce themselves to the broker. Each introduction includes resource specifications as CPU, Memory, etc. Using schedulers, such as [10], a resource can assign a percentage of CPU to a given machine effectively regulating the CPU usage of the group of processes encapsulated in it. And also it includes some parameters such as the bandwidth and the delay were calculated or estimated by the broker. Finally the broker inserts the record of QoS parameters for each resource into the database. We define the vector q^{res} which gives the capabilities of a resource as follows.

$$q^{res} = \left\langle q_1^{res}, q_2^{res}, \ldots, q_k^{res} \right\rangle \quad (1)$$

Generally it can be said that the client sends its request accompanied by vector of QoS parameters and the weights for the parameters as it appears in equations 2 and 3.

$$q^{task} = < q_1^{task}, q_2^{task}, \ldots, q_k^{task} > \quad (2)$$

$$W = < w_1, w_2, \ldots, w_k > \quad 0 <= w_i <= 1 \quad \sum_{i=1}^{k} w_i = 1 \quad (3)$$

Each weight is used to show the importance of each parameter. For example, if CPU is important for one task, the client will set 1 for the CPU weight and zero for the others.

Note that, q_i^{res} and q_i^{task} have the same unit, and the broker compares q_i^{res} with q_i^{task} for each i from 1 to k. If the resource provides the requirements needed for the task, it can be chosen as the best matched resource. We introduce *satisfy operator* ≋. Ri ≋ Tj means that the resource Ri can satisfy the task Tj and guarantees QoS parameters.

$$Ri \approx Tj = \left(\left(\sum_{l=1}^{k} \frac{q_l^{Res_i}}{q_l^{task_j}} \times w_l \right) \geq 1 \right) \quad (k = \text{the number of QoS parameters}) \quad (4)$$

The solution proposed here is that, we normalize the resource specification by the client requirement (q_i^{res} / q_i^{task}) and therefore the summation will be possible whereas the units of each parameter are different such as byte, bps, Mflops. When the resource capability exceeds the task demand, (q_i^{res} / q_i^{task}) is more than one.

Also, the client introduces a weight for each parameter to show the importance of the parameter. The weights range from 0 to 1 and the sum of all the weights is equal to one. We multiply the weight into (q_i^{res} / q_i^{task}) as mentioned in (4). Finally the best match resource will be the one that can provide the maximum of summation in (4).

Note that matching is done by aggregating QoS parameters which simplifies the matching method. You may say that using aggregated QoS parameters, some requirements may be neglected. This argument can be addressed by the introduction of the proper weights. If user assigns a higher weight for a QoS parameter, the requirement surely will be met.

The proposed aggregation in (4) has one more advantage, whereas the overheads and matching time is concerned. It is obvious that the matching time for aggregated QoS is a fraction of time when we do not aggregate the k papmeters..

The three methods for matching are proposed by using summation in (4).

3.1. Optimum

Assigning m resources to n tasks is the NP complete problem. The Optimum method has time complexity $O(n^3)$ to find the best matched resource for each task globally. Practically the Optimum method cannot be used because it has high overhead. We take into consideration the Optimum solution just for the sake of the comparison.

The problem is assigning m resources to n tasks with k QoS parameters. There are three matrices, one is T_{n*k} matrix given by (5) for task requirements, another is W_{n*k} matrix given by (6) for weight of requirements, and the other is R_{k*m} matrix given by (7) for resource capabilities. These matrices are shown in below.

$$T_{n*k} = \begin{bmatrix} q_1^{task_1} & q_2^{task_1} & \cdots & q_k^{task_1} \\ q_1^{task_2} & q_2^{task_2} & \cdots & q_k^{task_2} \\ \vdots & \vdots & \ddots & \vdots \\ q_1^{task_n} & q_2^{task_n} & \cdots & q_k^{task_n} \end{bmatrix} \quad (5)$$

$$W_{n*k} = \begin{bmatrix} w_1^{task_1} & w_2^{task_1} & \cdots & w_k^{task_1} \\ w_1^{task_2} & w_2^{task_2} & \cdots & w_k^{task_2} \\ \vdots & \vdots & \ddots & \vdots \\ w_1^{task_n} & w_2^{task_n} & \cdots & w_k^{task_n} \end{bmatrix} \quad (6)$$

$$R_{k*m} = \begin{bmatrix} q_1^{res_1} & q_1^{res_2} & \cdots & q_1^{res_m} \\ q_2^{res_1} & q_2^{res_2} & \cdots & q_2^{res_m} \\ \vdots & \vdots & \ddots & \vdots \\ q_k^{res_1} & q_k^{res_2} & \cdots & q_k^{res_m} \end{bmatrix} \quad (7)$$

We define WdT_{n*k} matrix as below.

$$WdT_{n*k} = \begin{bmatrix} w_1^{task_1} / q_1^{task_1} & w_2^{task_1} / q_2^{task_1} & \cdots & w_k^{task_1} / q_k^{task_1} \\ w_1^{task_2} / q_1^{task_2} & w_2^{task_2} / q_2^{task_2} & \cdots & w_k^{task_2} / q_k^{task_2} \\ \vdots & \vdots & \ddots & \vdots \\ w_1^{task_n} / q_1^{task_n} & w_2^{task_n} / q_2^{task_n} & \cdots & w_k^{task_n} / q_k^{task_n} \end{bmatrix} \quad (8)$$

$$V_{n*m} = WdT_{n*k} R_{k*m} \tag{9}$$

$$V_{n*m} = \begin{bmatrix} \sum_{i=1}^{k}\left(\frac{w_i^{task_1}}{q_i^{task_1}} q_i^{res_1}\right) & \sum_{i=1}^{k}\left(\frac{w_i^{task_1}}{q_i^{task_1}} q_i^{res_2}\right) & \cdots & \sum_{i=1}^{k}\left(\frac{w_i^{task_1}}{q_i^{task_1}} q_i^{res_m}\right) \\ \sum_{i=1}^{k}\left(\frac{w_i^{task_2}}{q_i^{task_2}} q_i^{res_1}\right) & \sum_{i=1}^{k}\left(\frac{w_i^{task_2}}{q_i^{task_2}} q_i^{res_2}\right) & \cdots & \sum_{i=1}^{k}\left(\frac{w_i^{task_2}}{q_i^{task_2}} q_i^{res_m}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{k}\left(\frac{w_i^{task_n}}{q_i^{task_n}} q_i^{res_1}\right) & \sum_{i=1}^{k}\left(\frac{w_i^{task_n}}{q_i^{task_n}} q_i^{res_2}\right) & \cdots & \sum_{i=1}^{k}\left(\frac{w_i^{task_n}}{q_i^{task_n}} q_i^{res_m}\right) \end{bmatrix} \tag{10}$$

So, equation (4) is based on multiplying WdT_{n*k} matrix to R_{k*m} matrix and the result is V_{n*m} matrix given by (9) and (10).

$V_{i,j}$ shows the value of (4) for assigning resource j to task i. If $V_{i,j} = 1$, the resource j exactly will provide the task i requirements. If $V_{i,j} < 1$, the resource j will be weaker than task i requirements. If $V_{i,j} > 1$, the resource j will be stronger than task i requirements.

We define MAX unary operator that it gets one n*m matrix and produces n*3 matrix. In the following, the operator semantic is explained.

$$B_{n*3} = \boxed{MAX}\ A_{n*m} \tag{21}$$

$$B_{n*3} = \begin{bmatrix} \underset{j=1}{\overset{m}{Max}}\left(A_{1,j}\right) & \underset{j=1}{\overset{m}{IMax}}\left(A_{1,j}\right) & \begin{cases} True & if B_{1,1} = 0 \\ False & else \end{cases} \\ \underset{j=1}{\overset{m}{Max}}\left(A_{2,j}\right) & \underset{j=1}{\overset{m}{IMax}}\left(A_{2,j}\right) & \begin{cases} True & if B_{2,1} = 0 \\ False & else \end{cases} \\ \vdots & \vdots & \vdots \\ \underset{j=1}{\overset{m}{Max}}\left(A_{n,j}\right) & \underset{j=1}{\overset{m}{IMax}}\left(A_{n,j}\right) & \begin{cases} True & if B_{n,1} = 0 \\ False & else \end{cases} \end{bmatrix} \tag{32}$$

Max returns the maximum value for each row of operand and *Imax* returns the index of maximum value in each row of operand. Third column of matrix B shows that if the first column of matrix B is zero, it will be True (i.e. this row's task is tested for matching) otherwise it will be False (i.e. this row's task is not matched yet). So we define matrix M_{n*3} as below:

$$M_{n*3} = \boxed{MAX}\ V_{n*m} \tag{43}$$

Figure 1 shows the Optimum algorithm that matches the best resource for each task globally. The algorithm finds the values of the assignment of each resource to n tasks at first. The purpose is assigning the maximum value resource for each task. Next, the algorithm finds the maximum value at each row which belongs to a task. It is possible that the resource with maximum value in one row may be also is maximum in other rows. Because of that, the algorithm chooses the row (task) which has minimum value (this task has maximum requirements). If the number of matching becomes equal to number of tasks (n) (that is the all of third column of matrix M is True), then the algorithm will finish.

```
matched = 0; V_n*m = WdT_n*k * R_k*m ;
while ( matched != n ) {
   M n*3 = MAX V n*m
    matched = FindMatch ( )
}
int FindMatch ( ) {
   int matched = 0;
   for ( i = 1; i <= n; i++) {
     if (M[i,3] == false ) {
          M[i,3] = true;          Min = M[i,1];          Ind = M[i,2];
          for (j = i + 1; j < n ; j++) {
            if ( M[j,2] = M[i,2])  { // resource M[j,2] is max for both task i
and task j
                    M[j,3] = true;
                    if (M[j,1] < Min )  {  //select minimum value for
matching with resource M[j,2]
                    Min = M[j,1];
                    Ind = M[j,2];
                }
            }
          }  // end of finding minimum
        V[i,*] = 0; V[*,Ind} = 0; // matched task i with resource Ind
          } else    matched ++;
     }
      return (matched);
}
```

Figure1. Optimum Algorithm

3.2. Grid-JQA

Grid-JQA has time complexity O(n) instead of O(n^3) for Optimum method and (as shown in next section by simulation), the difference between Grid-JQA and Optimum is acceptable. Grid-JQA uses threshold in (4) instead of finding maximum and minimum for each assignment (14).

$$Ri \text{ ₪ } Tj = \left(\left(\sum_{l=1}^{k} \frac{q_l^{Res_i}}{q_l^{task_j}} \times w_l\right) \geq Threshold\right) \quad (k = \text{the number of QoS parameters}) \tag{5}$$

The proposed solution is that instead of using maximum, we use a threshold (*Threshold* in (14)). If summation is more than the threshold, the broker will choose it as the best matched resource.

The broker starts calculating (14) for record R_1. It keeps calculation till finds R_i that satisfies the task. So the broker assigns R_i to the task. For next task to be matched, the broker starts from R_{i+1} in a round robin fashion, so that all resources can be covered.

Now, the problem is "what can be the amount of the threshold? ". For solving this problem, tacking into consideration that the active database is used, we introduce a rule in active database for calculating threshold in relation to environmental changes.

Suppose q_i^{res} equals q_i^{task} for all i from 1 to n (i.e. the requirements of task equal the capabilities of resource), then:

$$\sum_{k=1}^{n} \frac{q_k^{Ri}}{q_k^{Tj}} \times w_k = \sum_{k=1}^{n} 1 \times w_k = 1 \qquad (15)$$

So the minimum of threshold is 1.
If q_i^{res} equals P*(q_i^{task}) for all i from 1 to n, then :

$$\sum_{k=1}^{n} \frac{q_k^{Ri}}{q_k^{Tj}} \times w_k = \sum_{k=1}^{n} P \times w_k = P \qquad (16)$$

Therefore if we assign P to the threshold, it means that the selected resource should provide P times capability more than the requirements by the task.

Therefore when there are a lot of strong and free resources it is better that the broker assigns the strong resource to weak task. So the *Threshold* should be greater than one and the broker should select a resource precisely. In this method Grid-JQA acts like a Optimum method. But when there are a few strong resources it is better that to set the threshold equal one. So the broker should select a resource as soon as it finds a free resource and the broker does not wait for strong resources.

The value of threshold is changed by a rule of active database related to the number of times that equation 18 returns true or false. In this way if the number of true is more than false, the rule should be increased threshold for finding strong resource and otherwise should be decreased threshold for assigning resource to task as soon as possible.

Next section shows that Grid-JQA is similar to Optimum using simulation. But Grid-JQA has time complexity O(n) instead of time complexity O(n^3) for Optimum.

3.3. Dup Grid-JQA

After assigning m resources to n tasks, definitely some tasks are executed in weak resources which results is increasing the turnaround time of entire application. Dup Grid-JQA assigns free strong resources to tasks executed in weak resources in duplicate mode. Reference [11] shows that for the application with 100 tasks, 90% of the tasks are completed in about 39 minutes, but the application does not finish until 79 minutes have passed, which is almost identical to the turnaround time for a much larger application with 400 tasks.

There are two main causes of this plateau. The first cause is task failures that occur near the completion of the application. When a task fails, it must be restarted from scratch, and when this occurs near the end of the application execution, it will delay the application completion. The second cause is tasks assigned to slow resources. Once a task is assigned to a slow host, a FCFS scheduler without task preemption or replication capabilities will be forced to wait until the slow resource produces the result.

Dup Grid-JQA is a solution for the second cause. As when the 60% of execution time of application completes Dup Grid-JQA schedules remaining tasks in duplicate mode. In duplicate mode, the broker will send that results which is obtained first. Next section shows that duplicate method decreases the turnaround time significantly.

As the number of tasks gets large in comparison with the number of resources in the platform, the plateau becomes less significant, thus justifies the use of a FCFS strategy. However, for applications with a relatively small number of tasks, resource selection could improve the performance of application significantly.

4. Simulation

There are three subsections: 1) Application Definition 2) Resource definition 3) Matching methods and comparisons.

4.1. Application Definition

We use XML as input language. Figure 2 shows one example of application definition in simulation.

```
<Applications>
  <Application>
    <Name>a</Name>

         <CpuRequestMin>1000</CpuRequestMin>

         <CpuRequestMax>6000</CpuRequestMax>
            <CpuScoreMin>.9</CpuScoreMin>
            <CpuScoreMax>.9</CpuScoreMax>

<MemoryNeedMin>300</MemoryNeedMin>

<MemoryNeedMax>300</MemoryNeedMax>

   <BandwidthRequestMin>70</BandwidthRequestMin>

   <BandwidthRequestMax>70</BandwidthRequestMax>
              <CountMin>20</CountMin>
              <CountMax>20</CountMax>

         <CpuCycleMin>800000000</CpuCycleMin>

         <CpuCycleMax>800000000</CpuCycleMax>
         </Application>
     </Applications>
```

Figure 2. Application Definition

The random function is used to provide random number between Min and Max for each field in simulation so that the environment gets more similar to grid environment that rapidly changes.

4.2. Resource Definition

Figure 3 shows one example of resource definition by XML language. The requirement of memory must be completely satisfied by the resource, but the assignment of the memory more than requirements can not improve performance. Thus the requirement of memory is not used in aggregated QoS parameters in equation 14. Therefore the weight of bandwidth is the weight of CPU subtracted from one. The CPU power and bandwidth have a main role in improving

turnaround time and if the broker assigns stronger CPU or more bandwidth, this assignment will cause the reduction of turnaround time. The application definition consists of the following fields:

1) Min/Max of CPU power requirement in cycle/sec (CpuRequestMin/Max)
2) Min/Max of CPU requirement's weight (CpuScoreMin/Max)
3) Min/Max of memory requirement (MemoryNeedMin/Max)
4) Min/Max of bandwidth requirement (BandwidthRequestMin/Max)
5) Min/Max of the number of tasks in application (CountMin/Max)
6) Min/Max of size of tasks in cycle/sec (CpuCycleMin/Max)

```
<Resources>
   <Resource>
        <Name>r1</Name>
        <CpuPowerMin>1000</CpuPowerMin>
        <CpuPowerMax>6000</CpuPowerMax>
        <MemoryMin>400</MemoryMin>
        <MemoryMax>400</MemoryMax>
        <BandwidthMin>100</BandwidthMin>
        <BandwidthMax>100</BandwidthMax>

    <StopQueueTimeMin>100000</StopQueueTimeMin>

   <StopQueueTimeMax>100000000</StopQueueTimeMax>
     </Resource>
  </Resources>
```

Figure 3. Resource Definition

The resource definition consists of the following fields:

1) Min/Max of CPU cycle capabilities in cycle/sec (CpuPowerMin/Max)
2) Min/Max of memory in KB (MemoryMin/Max)
3) Min/Max of available bandwidth from the resource to the broker (BandwidthMin/Max)
4) Min/Max of availability duration for the resource (StopQueueTimeMin/Max)

As the application definition, the random function generates the random number between Min and Max for each above fields.

4.3. Matching Methods and Comparisons

In simulation we use following five methods for matching:

1) The General method that matches resources with tasks in FIFO mode (first task to first free resource) regardless of QoS parameters.
2) The Optimum method, as described in section IV, selects the best resources for tasks but it has time complexity $O(n^3)$.
3) The Grid-JQA, our proposed solution, does matching almost like Optimum method but it has much less time complexity (O(n)) than Optimum matching.
4) Dup Grid-JQA, out proposed solution, that adds new feature to Grid-JQA. This feature is duplicate execution of delayed tasks (i.e. executed in weak resources).
5) The Wait method. In all other methods, if the broker does not find the proper resource, it will assign the best available resource to task. So the task will not wait for the proper resource. But in the Wait method tasks wait until the proper resource is found.

In this simulation, we assume that the CPU cycle is from 1000 to 6000, and all resources have same amount of memory and bandwidth. The number of tasks in application is 20 and the CPU weight is 0.9. All tasks require same amount of memory and bandwidth.

We choose the CPU cycle requirement of application in range 1000 to 6000 which is similar to the range chosen for resource's CPU cycle power. We do simulation 6 times and each time we consider the amount of CPU cycle requirement 1000, 2000, 3000, 4000, 5000 and 6000. For each CPU cycle requirement, the simulation is repeated 100 times and finally we use the average turnaround time.

Figure 4 shows the result of simulation for 6000 CPU cycle requirement with resource number from 10 (half of the task number) to 60 (three time more than the task number). After this, in all figures, horizontal axis indicates the number of the resources, and the vertical axis indicates the turnaround time in msec.

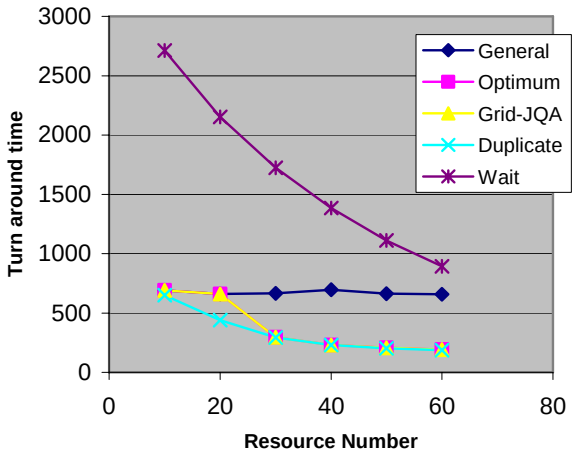

Figure 4. 6000 CPU cycle request

The simulation results show that: 1) Executing in weak resource in comparison with waiting to proper resource produces less turnaround time. 2) For strong requirement Grid-JQA is like Optimum method. Because, if Grid-JQA does not find the proper resource, it will assign the best resource as like Optimum. 3) General method does not change after 20 free resources, because it uses the 20 first free resources. 4) Dup Grid-JQA has minimum turnaround time when the number of free resource is from 10 (half of the task number) to 30 (1.5 times more than the task number). Because there are some tasks executed in weak resources (e.g. 1000-2000 CPU cycle/sec), they can find chance of executing in strong resources (e.g. 5000-6000 CPU cycle/sec). So the turnaround time is decreased. 5) When the number of free resources is less than the number of tasks, the Optimum method and Grid-JQA do not have better performance than the General method. They improve the results when the number of free resources gets more than the number of tasks. This situation is more likely because we suppose the environment has a lot of free resources (e.g. Internet). 6) Also, when the number of free resources is more than two times of the task number, the results are not improving.

Figures 5, 6, 7, 8 and 9 are for CPU cycle requirements in application for 5000, 4000, 3000, 2000 and 1000.

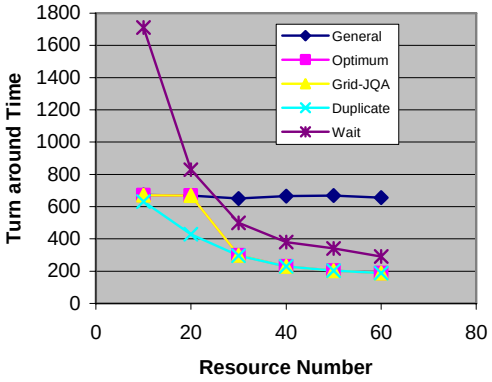

Figure 5. 5000 CPU cycle request

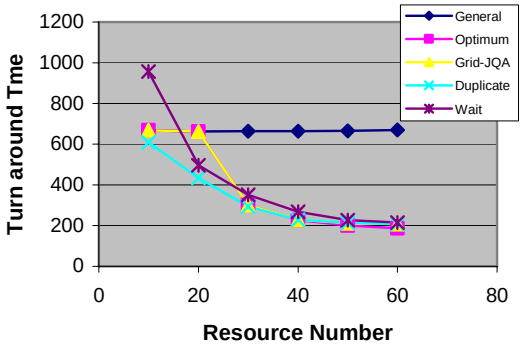

Figure 6. 4000 CPU cycle request

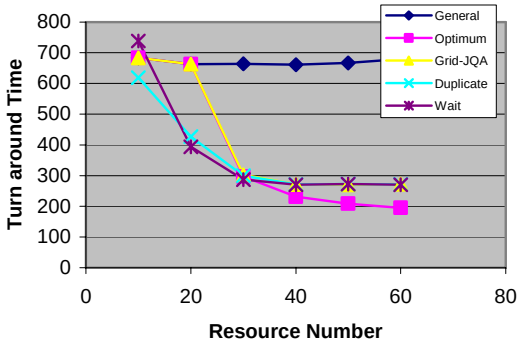

Figure 7. 3000 CPU cycle request

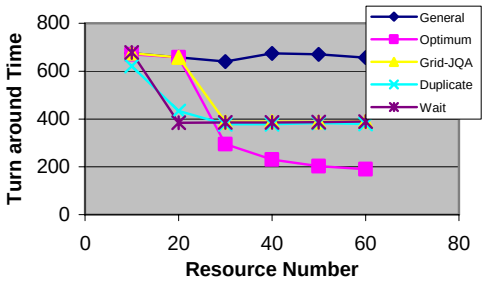

Figure 8. 2000 CPU cycle request

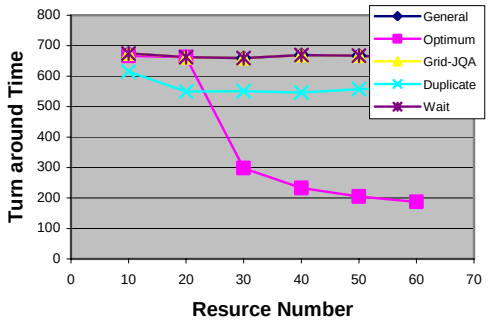

Figure 9. 1000 CPU cycle request

Comparing above figures, we can find following results:

1) Most of the time, Grid-JQA has less turnaround time than Wait method (Figures 4, 5, and 9). Only when the CPU cycle request is low (e.g. 2000 and 3000) and the number of free resource is from 10 (half of the task number) to 30 (1.5 times ore than the task number), the Wait method has less turnaround time (Figures 7 and 8). So executing in weak resource is better than waiting for proper resource.

2) Comparing Grid-JQA and Optimum method shows that they are same for strong request (e.g. 4000-6000) in figures 4, 5 and 6. The difference for 3000 CPU cycle request is ignorable. The difference between Grid-JQA and Optimum is for low requests (e.g. 1000-2000) that Optimum method has less turnaround time than Grid-JQA. But two points should be noted: First the user has low request so the execution time is longer and if the user wants less execution time, it should require more capabilities. Second, in this simulation the threshold is one, but as explained before, the threshold can be changed dynamically in related to environment changes so Grid-JQA can be similar to Optimum method.

For example if the CPU cycle request is 1000 and the threshold is 2, as shown in figure 10 the difference between Grid-JQA and Optimum will be decreased. By increasing the threshold, the difference decreases. So with changing the threshold, Grid-JQA approaches to Optimum method.

3) Grid-JQA, our proposed solution, can provide the results close to the Optimum method and has the advantage of time complexity O(n) instead of O(n3).

4) Most of the time Dup Grid-JQA has less turnaround time in comparison with other methods. And also it has less cost in comparison with retrying and check pointing.

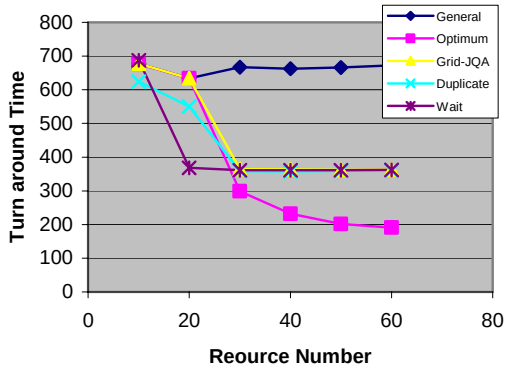

Figure 10. 1000 CPU cycle request when threshold is 2

Finally, we do simulation 100 times with resources randomly between (1000 – 6000) and application with requests randomly between (1000 – 6000) with 20 tasks. Figure11 shows the results of this simulation.

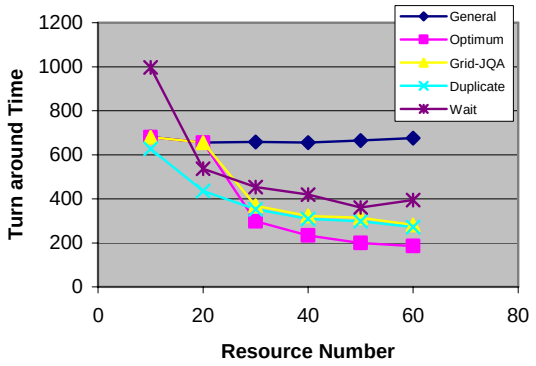

Figure 11. Simulation for random request (1000-6000) and random capabilities (1000-6000)

Final results are:

1) Wait method has not better performance and starting execution in available resource is better than waiting for a suitable resource.

2) Adaptation (Dup Grid-JQA) has good results that is tasks executed in weak resources, can be executed in duplicate mode in strong resources. And the broker uses the results produced sooner.

So Dup Grid-JQA is suitable and reliable method for matching in grid environment.

5. Conclusions

This paper discusses the matching methods in grid environment when QoS parameters are concerned. The Optimum method for matching n tasks with m resources is proposed by using matrices formalism. Grid-JQA and Dup Grid-JQA are proposed and compared with Optimum method. The Optimum method has time complexity $O(n^3)$ that is decreased to O(n) by using Grid-JQA with ignorable differences. Meanwhile, the Wait method is evaluated and the simulation results show that executing in available resource has better performance than waiting for proper resource. Finally, Dup Grid-JQA has minimum turnaround time and it is the best solution for matching in grid environment.

References

[1] I. Foster, and C. Kesselman, *The GRID: Blueprint for a New Computing Infrastructure*, 2nd Edition. Morgan-Kaufmann, San Mateo, CA, 2004.

[2] M. J. Lewis, and A. Grimshaw, *The Core Legion Object Model*. In Proc. of the Fifth IEEE Int'l Symposium on High Performance Distributed Computing, August 1996.

[3] Uddi.com. *UDDI: Technical White Paper. http://www.uddi.*com/pubs/Iru UDDI Technical White Paper.pdf.

[4] Hewlett Packard Inc. espeak: *The Universal Language of E-Services*. http://www.e-speak.net/.

[5] R. Raman, M. Livny, and M. Solomon, *Policy driven heterogeneous resource co-allocation with gangmatching*. In HPDC-12, 2003.

[6] C. Liu, and I. Foster, *A constraint language approach to matchmaking*. In International workshop on Research Issues on Data Engineering (RIDE 2004), 2004.

[7] C. Liu, L. Yang, I. Foster, and D. Angulo, *Design and evaluation of a resourceselection framework for grid applications*. In Proceedings of the Eleventh IEEE Symposium on High-Performance Distributed Computing, July 2002., 2002.

[8] L. M. Khanli, and M. Analoui, *Grid-JQA - a new architecture for QoS-guaranteed grid computing system*, Proc. of the 14th Euromicro Conference on Parallel, Distributed and Network-based processing, France, PDP2006, Feb 15-17, 2006.

[9] L. M. Khanli, and M. Analoui, *Grid-JQA - grid java based quality of service management by active database*, Proc. of the 4th Australian Symposium on Grid Computing and e-Research, AusGrid06, Jan.2006.

[10] *Xen Scheduler Howto.* 2005: http://xen.terrabox.com/index.php/Sched-HOWTO.

[11] D. Kondo, *Scheduling task parallel applications for rapid turnaround on desktop grids* , Ph.D. thesis, university of California, San Diego, 2005.

Leila Mohammad Khanli received his B.S. (1995) from Shahid Beheshti University Tehran, Iran, M.S. (2000) from IUST (Iran University of Science and Technology) University, Ph.D. degree (2008) from IUST. All are in computer engineering. She is currently assistant professor in the Department of Computer Science at Tabriz University. Her research interests include grid computing and Quality of Service management.
E-mail: l-khanli@tabrizu.ac.ir

Morteza Analoui received a B.Sc. degree in electrical engineering from Iran University of Science & Technology and a Ph.D. degree in telecommunication from Okayam University. He is currently assistant professor in the Department of Computer Engineering at Iran University of Science & Technology, where he is also director of the Networking Laboratory and the director of IT Business Incubator. Dr Analoui has been an assistant professor at Electrical and Electronic Engineering of Okayama University, Japan, and Electrical & Computer Engineering Department of Tarbiat Modaress University, Tehran, Iran. His research interests include modeling and performance evaluation, network protocols and architecture, network measurement, multimedia communication, and pattern recognition.
E-mail: Analoui@iust.ac.ir