# Selection Algorithm for Grid Services based on a Quality of Service Metric

Simone A. Ludwig and S.M.S. Reyhani
*Department of Computer Science, University of Saskatchewan*
*{ludwig,mreyhani}@cs.usask.ca*

## Abstract

*Grid computing is one of the main paradigms for resource-intensive scientific applications. It enables resource sharing and dynamic allocation of computational resources, thus increasing access to distributed data, promoting operational flexibility and collaboration, and allowing service providers to scale efficiently to meet variable demands. Large-scale grids are complex systems composed of many components from different domains. Quality of service (QoS) in such environments is an important issue due to the distributed nature of the services. In this paper we propose an allocation algorithm for matching service requests of Grid users with Grid services based on a QoS metric using either matchmaking or a well-tested genetic algorithm: NSGA-II. Experiments are performed and results are discussed for both approaches.*

## 1. Introduction

The computational speed of individual computers has increased by about 1 million times in the past 50 years. However, they are still not fast enough for more and ever more scientific problems. For example, in a few physics applications, data is produced by the fastest contemporary supercomputer. The analysis of this data would need much more computational power than presently available. [1]

The goal of High Performance Computing is to increase the effectiveness and applicability of high-performance methods and infrastructure across a whole range of application areas in science, engineering, medicine, industry and commerce. In the mid 1990s, Ian Foster and Carl Kesselman proposed a distributed computing infrastructure for advanced science and engineering, which they called "The Grid". The vision behind the Grid is to supply computing and data resources over the Internet seamlessly, transparently and dynamically when needed, such as the power grid supplies electricity to end users. The Grid originated from trying to solve the information and computational challenges of science [2].

As the application areas for high performance computing technology become ever wider and economically more important while the supporting infrastructure becomes ever more powerful but complicated, the need for a well-founded and sophisticated approach becomes ever more apparent. One of them is service computing, meaning the provisioning of services for high performance computing.

Resource discovery and as a result also service discovery is an important issue for the Grid in answering the questions of how a service requester finds the resources/services needed to solve its particular problem and how a service provider makes potential service requesters aware of the computing resources it can offer. Service discovery is a key concept in a distributed Grid environment. It defines the process for locating service providers and retrieving service descriptions. The problem of service discovery in a Grid environment arises through the heterogeneity, distribution and sharing of the resources in different Virtual Organizations (VOs) [3].

Some examples of service matching in the Grid area are the workflow generator [4], the ontology-based resource selector [5] and the semantic matchmaker [6]. The *workflow generator* addresses the problem of automatically generating job workflows for the Grid. Deelman et al. have developed two workflow generators. The first one maps an abstract workflow defined in terms of application-level components to the set of available Grid resources. The second generator takes a wider perspective and not only performs the abstract to concrete mapping but also enables the construction of the abstract workflow based on the available components.

The *ontology-based resource selector* exploits ontologies, background knowledge, and rules for solving resource matching in the Grid. The aim being to overcome the restrictions and constraints of resource descriptions in the Grid. In order to make the matchmaking more flexible and also to consider the structure of VOs the framework consists of ontology-based matchmakers, resource providers and resource consumers or requesters. Resource providers periodically advertise their resources and capabilities to one or more matchmakers using advertisement messages. The user can then activate the matchmaker by submitting a query asking for resources that satisfy the request specification.

The *semantic matchmaker* proposes a solution to achieve interoperability for service discovery in a Grid environment. The matchmaking framework for service discovery is based on three selection stages which are context, semantic and registry selection. The proposed matchmaking framework provides a better service discovery process by using service semantics stored in ontologies instead of only performing service name matches which common service discovery systems are restricted to. The framework permits Grid applications to specify the criteria a service request is matched with and enables interoperability for the matchmaking process.

Many applications utilizing the Grid computing infrastructure require the simultaneous allocation of resources, such as compute servers, networks and memory capacity, disk storage and other specialized resources. Collaborative working and distributed visualization provide two examples of such applications. In such applications, the user generally executes a service remotely, and expects to receive a result within a particular time period. There is therefore a more stringent requirement on such services to conform to particular Quality of Service (QoS) criteria – as not meeting such criteria is paramount to the application producing incorrect results.

QoS metrics should be specified with reference to Grid services, and not just to the network connecting these services (which has been the focus of related research e.g. in multimedia applications). With the emerging interest in Service-Oriented Grids, resources may be advertised and traded as services based on a Service Level Agreement (SLA). A user requesting a particular service must identify QoS requirements (such as networking parameters (e.g. bandwidth), computational parameters (e.g. number of processors), the duration over which the service is required, and the expected cost the user is willing to pay). Similarly, a service provider must annotate their service with QoS capabilities. Some of these capabilities are directly managed by the service provider (such as cost), whilst others are obtained through third party resource and network managers [7].

We are envisioning a service rich environment, where service requesters are represented by intelligent agents. If interaction between agents is automated, it is necessary for these agents to be able to automatically discover services and choose between a set of equivalent (or similar) services. In such a scenario QoS attributes as availability, reliability etc. are essential properties to guarantee a smooth execution of services. In our scenario, many service requests coming in have to be satisfied at the same time. The matchmaking service dealing with the incoming requests is being evaluated based on performance and scalability.

One very important investigation to make high performance computing worthwhile is the guarantee of the delivery of a service. This paper compares the approach of matchmaking for the selection of services based on a QoS metric to the use of a genetic algorithm to find the best matches. The results reveal the point at which the genetic algorithm outperforms matchmaking given a certain number of requests and services. In the next section we describe the background to the QoS metric chosen for the experiments, Section 3 describes the two approaches and outlines the application scenario. In Section 4 the experiment setup is described and the results delineated. Finally, in Section 5 we summarize the findings and draw conclusions.

## 2. QoS Metric

We adopt a Grid services quality model based on a set of quality criteria (i.e. non-functional properties) that are adopted by all Grid and also Web services, for example, their pricing and reliability. Although the adopted quality model has a limited number of criteria (for the sake of illustration), it is extensible: new criteria can be added without fundamentally altering the service selection techniques built on top of the model. In particular, it is possible to extend the quality model to integrate non-functional service characteristics such as those proposed in [8]. In this section, we present the quality criteria in the context of services. For each criterion, we provide a definition, indicate its granularity, and provide rules to compute its value for a given service. Five generic quality criteria for elementary services are considered [9,10]: (1) *execution duration*, (2) *execution price*, (3) *reputation*, (4) *reliability*, and (5) *availability*.

*Execution duration*: The execution duration $q_{du}(s, op)$ measures the expected delay in seconds between the time when the execution starts and the time when the execution ends. The execution duration is computed by $q_{du}(s,op) = T_{process}(s,op) + T_{trans}(s,op)$, considering the execution duration as the sum of the processing time $T_{process}(s,op)$ and the transmission time $T_{trans}(s,op)$. Services advertise their processing time or provide methods to enquire about it. The transmission time is estimated based on past executions of the service operations, i.e., $T_{trans}(s,op) = \frac{\sum_{i=1}^{n} T_i(s,op)}{n}$, where $T_i(s,op)$ is a past observation of the transmission time, and $n$ is the number of execution times observed in the past.

*Execution price*: The execution price $q_{price}(s,op)$ is the amount of money that a service requester has to pay for executing operation $op$ of a service $s$. Service providers either directly advertise the execution price of their operations, or they enquire about it.

*Reputation*: Reputation $q_{rep}(s)$ of a service $s$ is a measure of its trustworthiness. It mainly depends on end user's experiences of using service $s$. Different end users may have different opinions about the same service. The value of the reputation is defined as the average ranking given to the service by end users, i.e., $q_{rep} = \frac{\sum_{i=1}^{n} R_i}{n}$, where $R_i$ is the end user's ranking on a service's reputation, $n$ is the number of times the service has been graded. End users are normally given a range to rank services, for example $[0,5]$.

*Reliability*: The reliability $q_{rel}(s)$ of a service $s$ is the probability that a request is correctly responded to within the maximum expected time frame. Reliability is a measure related to hardware and/or software configuration of services and the network connections between the service requesters and providers. Reliability is computed from historical data (if available, if not then assume a threshold value of 0.5) about past invocations using the expression $q_{rel}(s) = \frac{N_c(s)}{K}$, where $N_c(s)$ is the number of times that the service $s$ has been successfully delivered within the maximum expected time frame, and $K$ is the total number of invocations.

*Availability*: The availability $q_{av}(s)$ of a service $s$ is the probability that the service is accessible. The value of the availability of a service $s$ is computed as follows: $q_{av}(s) = \frac{T_a(s)}{\theta}$, where $T_a(s)$ is the total amount of time (in seconds) in which service $s$ is available during the last $\theta$ seconds ($\theta$ is a constant set by an administrator of the service community).

## 3. Approach

Our approach is based on finding the best service request – matched service pairs with the highest match score based on the QoS parameters are listed above. Consider we have many different services with certain QoS parameters and many service requesters specifying their service requests as a range with lower and upper bound for each QoS attribute assuming the average value to be the preferred one. The aim is to find an algorithm which matches service requests and services, achieving the best match score for each of them in a reasonable time. We compare two different approaches: matchmaking and a genetic algorithm.

### 3.1. Matchmaking

Matchmaking is concerned with matching service requests with services based on a range of QoS attributes. We assume that services have a fixed value ($S_i$) for all the attributes whereas the service requests have a range [$RU_i \ldots RL_i$]. The match value for each attribute is calculated as:

$$MV_i = \begin{cases} 0 & \text{for } RU_i < S_i \text{ or } RL_i > S_i \\ \omega_i \cdot \left( \frac{RU_i - S_i}{RU_i - RL_i} \right) & \text{for } RL_i \le S_i \le RU_i \end{cases} \quad (1)$$

Whereby:

$RU_i$: represents the service request's upper value for attribute $i$; $RL_i$: is the service request's lower value for attribute $i$; $S_i$: represents the service's value for attribute $i$; $\omega_i$: is the weight value for attribute $i$.

The overall match score is the sum of all match values divided by the number of QoS attributes:

$$MS = \frac{\sum_{i=1}^{n} MV_i}{n} \quad (2)$$

The matchmaking algorithm (Figure 1) implemented uses equations (1) and (2) to calculate the match score for each service request and service. As each service request might match several services the assignment of one service request with one service is carried out as follows: The algorithm takes the first service request and determines the service with the highest match score. This service is removed from further consideration, as a service requester has been found. Then, the best match score to the remaining services is determined for the second service request; that service is then removed, and so on until each service request is matched to a service.

The main strength of this approach is that each service requester is only considered once, while the main weakness is that service requesters later in the list are very likely to get assigned with a service having a worse match score. This is where the genetic algorithm can potentially enhance the assignment procedure, as it does not achieve matches in a linear fashion.

```
Calculation of Match Score:
for all requests do
    for all services do
        for all QoS attributes do
            calculate_MatchValue()
        end for
        calculate_MatchScore()
        store_Vec_Service_MatchScore()
    end for
    store_Vec_Match_Pair_MatchScore()
end for
return Vec_Match_Pair_MatchScore

Best Request-Service Pair Match Assignment:
for all requests do
    for all services matching a request do
        if matchScore is higher than previous
            & service is not taken then
            assign_Request_Service_MatchScore()
        end if
    end for
    store_Request_Service_MatchScore()
end for
return Vec_Request_Service_MatchScore
```

**Figure 1. Matchmaking Algorithm**

## 3.2. Genetic Algorithm

A genetic algorithm (GA) is a heuristic used to find approximate solutions to difficult-to-solve problems by applying the principles of evolutionary biology to computer science. GAs use biologically-derived techniques such as inheritance, mutation, natural selection, and recombination (or crossover) [11]. GAs are typically implemented as a computer simulation in which a population of solutions (or individuals) to an optimization problem evolve towards better solutions. This is possible as each solution is a chromosome which can undergo genetic modification.

In this study, each chromosome is a randomly generated permutation of integers, where each integer represents a given service requester. The length of the chromosome is therefore equal to the number of service requesters in the given test circumstance. Given the services are a static integer permutation from 1 to *N*, each chromosome determines which service requester is matched to which service. The number of individuals in the population is an alterable test parameter.

The process starts with a population of completely randomly generated individuals. In each generation, the fitness of each population member is evaluated. The fittest individuals, in terms of best match score - for example, form an archive population, where the best solutions found so far are saved. As even the quality of solutions here can range widely - particularly in earlier generations, members compete in binary tournaments, with winners forming a mating pool. Two parents are randomly selected from the pool, and undergo cycle crossover [12] and mutation to form two children. This is repeated until the new population of size *N* is filled. The new population is evaluated; its members compete for inclusion in the archive, and the process repeats until either a set number of generations are completed, stagnation, or termination criteria is met.

The genetic algorithm used in our study is NSGA-II. NSGA-II is a fast elitist non-dominated sorting genetic algorithm. For an excellent overview of evolutionary optimization techniques, the interested reader is referred to [13], and for a full description of NSGA-II to [14]. In brief, in NSGA-II the most fit individuals from the union of archive and child populations are determined by a *ranking* mechanism (or *crowded comparison operator*) composed of two parts. The first part 'peels' away layers of non-dominated fronts, and ranks solutions in earlier fronts as better. The second part computes a dispersion measure, the *crowding distance*, to determine how close a solution's nearest neighbors are, with larger distances being better. It is employed here to search for better match sequences, which guide the evolutionary process toward solutions with better objective values. In this paper, NSGA-II is allowed to continue until the best objective measures in the archive do not improve for 50 generations, after which the non-dominated solutions are saved.

Non-dominated solutions are desirable in the sense that it is impossible to find another solution in the set which improves the value on any objective (i.e., number of service requests and services which match on all five objectives or match function score) without simultaneously degrading the quality of the other objective, and is formally defined as follows:

**Definition 1** *(Non-dominated) Let* $o_1, o_2, \ldots, o_n$ *be objective functions which are to be maximized. Let* $S$ *be the set of obtained solutions.* $s \in S$ *is dominated by* $t \in S$ *(denoted* $t \succ s$*) if* $\exists j$, $j \in \{1, \ldots, n\}$*, such that* $o_j(t) > o_j(s)$ *and* $\forall i$, $1 \le i \le n$, $o_i(t) \ge o_i(s)$*. A non-dominated solution is therefore any solution* $s \in S$ *which is not dominated by any other* $t \in S$*.*

The set of all possible non-dominated solutions from the entire search space constitute the *Pareto front*.

Solutions which lie on the Pareto front represent the best trade-off between the number of total matches and match function score. Due to search space size, it is only feasible to approximate the true Pareto front for a non-trivial problem scenario. As the best permutation could be any ordering, the search space size here is $2^N$, where $N$ is the number of service requesters and services.

## 4. Experiments and Results

The ability of two algorithms was compared on how effectively they matched service requesters to an equal number of services on five objective measures with random ranges. The five objective measures or QoS attributes were: price, duration, reputation, reliability and availability.

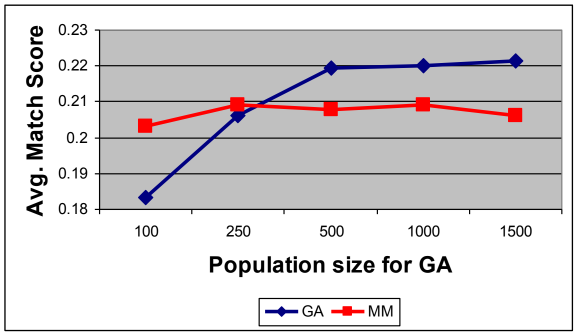

**Figure 2. NSGA-II vs. Matchmaking for 20 Requesters and Services**

The number of service requests and services were set at 20, 40, 60, 80, and 100. The number of population members in NSGA-II was set to 100, 250, 500, 1000, and 1500. For each trial of NSGA-II, the matchmaking algorithm was run using the same randomly generated data set. Ten trials at each potential combination were run, which led to a total of 250 trials (5 requester/service sizes times 5 population sizes times 10 trials).

In addition to match function scores, a performance measurement was taken (i.e., execution time in seconds) and rate of convergence for NSGA-II was measured. Figure 2 shows the match score distribution for NSGA-II and the matchmaking for 20 requesters and services. We see that with 20 requesters and services, NSGA-II outperformed matchmaking at population sizes greater than 350.

In Figure 3 we see the match score distribution for 40 requesters and services. Here NSGA-II outperformed matchmaking starting at a population size of 450.

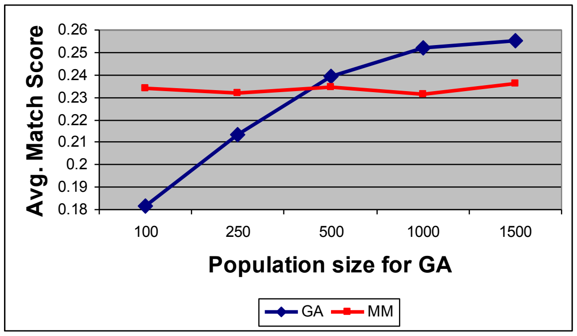

**Figure 3. NSGA-II vs. Matchmaking for 40 Requesters and Services**

Figure 4 shows the distribution for 60 requesters and services, where NSGA-II outperformed matchmaking starting at a population size of 800.

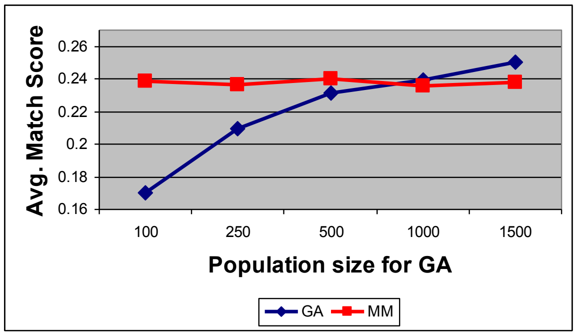

**Figure 4. NSGA-II vs. Matchmaking for 60 Requesters and Services**

For 80 requesters and services, NSGA-II outperformed matchmaking at a population size of 1000 (see Figure 5), and at population size of 1300 for 100 requesters and services (see Figure 6).

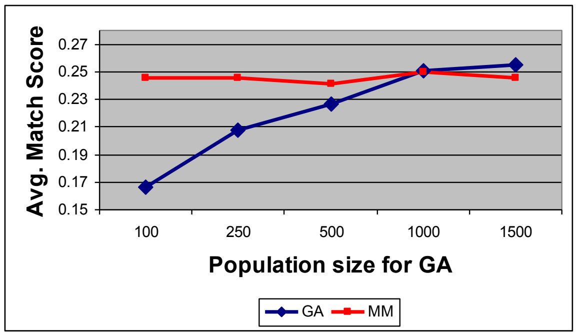

**Figure 5. NSGA-II vs. Matchmaking for 80 Requesters and Services**

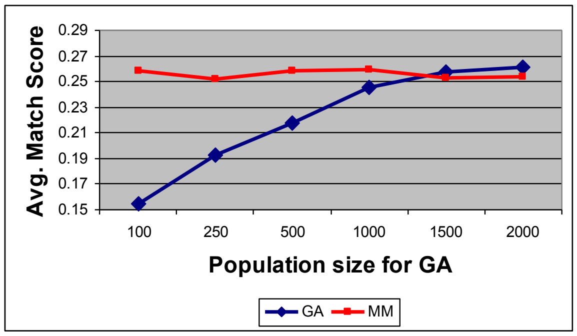

**Figure 6. NSGA-II vs. Matchmaking for 100 Requesters and Services**

The average time to complete each trial can be seen in Table 1 for NSGA-II - in relation to the number of requesters and services, while the overall average time in seconds for matchmaking was 0.11 seconds. Therefore, while matchmaking is faster than NSGA-II, NSGA-II is still reasonably fast considering high solution quality is very desirable.

The rate at which NSGA-II converged was similar to the average execution time in seconds, with an average convergence in 36.82 generations with 20 requesters and services, 74.54 for 40, 98.24 for 60, 106.34 for 80, and 122.87 for 100. In GA terms, this is reasonably fast, and suggests the chosen representation was suitable.

We can see from the overall comparison of the two approaches that the matchmaking returns similar values repeatedly given the same number of requesters and services. This is expected, as the search is linear, whereas NSGA-II shows an exponential distribution, as it contends with a growing search space.

Because of the nature of results (i.e., they are suitable to linear regression), it is possible to accurately estimate the point at which the GA will outperform matchmaking, and how long it will take to perform the search, given a certain number of requesters and services.

| Number of Requesters and Services | Population Size (Outperform point) | Execution Time for NSGA-II in seconds |
|---|---|---|
| 20 | 350 | 31.91 |
| 40 | 450 | 49.48 |
| 60 | 800 | 65.83 |
| 80 | 1000 | 77.15 |
| 100 | 1300 | 92.82 |

**Table 1. Average Measurement Results**

Table 1 shows the average measurement results in numerical format, while Figure 7 and 8 show them graphically. In particular, the relationship between the number of requesters and services and the population size where the GA outperforms matchmaking is provided. Figure 7 shows this relation and a linear regression and equation for determining additional scenarios. For example, if there were 200 requesters and services, the GA can be expected to outperform matchmaking given a population size of 2,685.

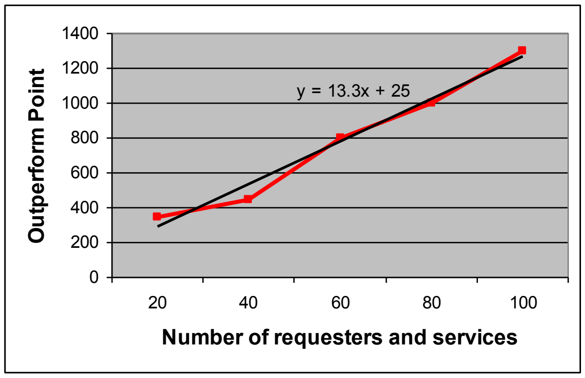

**Figure 7. Outperform Point vs. Number of Requesters and Services**

In Table 1 the execution time in comparison with the number of requesters and services is also shown. Figure 8 shows this relation and a linear regression and equation for determining additional scenarios. For example, if there were 200 requesters and services, the GA can be expected to outperform matchmaking in 172 seconds.

Three points were measured which confirm the calculated execution time regression line. The number of requesters and services was 200, 500 and 2000 which resulted in execution times of 186.79, 378.12 and 1698.81 seconds respectively. The average mean error for these three measurement points was 8.08%

which shows that the regression line is quite a good approximation.

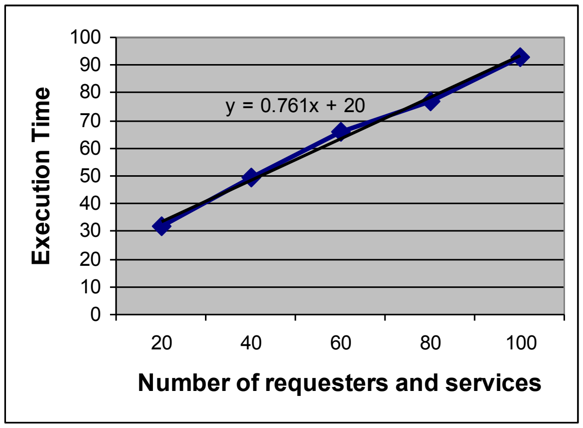

**Figure 8. Execution Time vs. Number of Requesters and Services**

## 5. Conclusion

The goal of High Performance / Grid Computing is to increase the effectiveness and applicability of high-performance methods and infrastructure across a whole range of application areas. This paper investigated the selection of service requests using two approaches to find the best matches for a single service depending on a variable but equal number of requesters and services. The two approaches chosen were matchmaking and a GA: NSGA-II. The measurements revealed that NSGA-II outperforms matchmaking when a suitable population size is used. Two equations were derived from the results which allow good estimations of both the size of the population to use and the average speed of execution. In general, larger number of requesters and services means greater population sizes and more time for execution. In conclusion, considering matchmaking is commonly used for service matching, evidence here suggests that if high solution quality is paramount, GAs can do a better job in a reasonable amount of time.

## 6. Acknowledgements

The authors would like to thank L. Raisanen for a first implementation of the Genetic algorithm used for this evaluation.

## 7. References

[1] S. A. Ludwig and S. M. S. Reyhani, "Introduction of Semantic Matchmaking to Grid Computing", Journal of Parallel and Distributed Computing, vol. 65, no.12, pp. 1533-1541, 2005.
[2] C. Goble, "The Grid - From concept to reality in distributed computing", Bioinformatics World Article, 2003.
[3] S. A. Ludwig and S. M. S. Reyhani, "Semantic Approach to Service Discovery in a Grid Environment", Journal of Web Semantics, vol. 4, no. 1, pp. 1-13, 2006.
[4] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments", Journal of Grid Computing, Vol. 1, No. 1, pp 9--23, 2003.
[5] H. Tangmunarunkit, S. Decker and C. Kesselman, "Ontology-based Resource Matching in the Grid - The Grid meets the Semantic Web", Proceedings of Twelfth International World Wide Web Conference 2003. Budapest, Hungary. May 2003.
[6] S. A. Ludwig et al., "A Grid Service Discovery Matchmaker based on Ontology Description", Proceedings of 2nd International EuroWeb2002 Conference, Oxford, UK, December 2002.
[7] R. Al-Ali, O. Rana, D. Walker, S. Jha and S. Sohail, "G-QoSm: Grid Service Discovery Using QoS Properties", Computing and Informatics Journal, 21 (4), 2002.
[8] J. O'Sullivan, D. Edmond and A. Hofstede, "What's in a Service", Distributed and Parallel Databases, 12(2-3):117-133, September 2002.
[9] A. van Moorsel, "Metrics for the internet age: Quality of experience and quality of business", Proceedings of 5th Performability Workshop, September 2001, Erlangen, Germany.
[10] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Z. Sheng, "Quality driven web services composition", Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, 2003.
[11] M. Mitchell, "An Introduction to Genetic Algorithms (Complex Adaptive Systems)", The MIT Press, ISBN 0-262-63185-7, 1998.
[12] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley, "A comparison of genetic sequencing operators", In Rick Belew and Lashon Booker, editors, Morgan Kaufman, Proceedings of the Fourth International Conference on Genetic Algorithms, pages 69-76, San Mateo, CA, 1991.
[13] K. Deb, "Multi-objective optimization using evolutionary algorithms", Wiley, Chichester, England, 2001.
[14] K. Deb, S. Agrawal, A. Pratap and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II", In Lecture Notes in Computer Science, volume 1917, pages 848-849, 2000.