RC24209 (W0703-030) March 7, 2007
Computer Science

IBM Research Report

A Hybrid Linear Programming and Evolutionary Algorithm Based Approach for On-line Resource Matching in Grid Environments

Pawel Garbacki
Delft Universityof Technology
Delft, The Netherlands

Vijay K. Naik
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

A Hybrid Linear Programming and Evolutionary Algorithm based Approach for On-line Resource Matching in Grid Environments

Paweł Garbacki
Delft University of Technology
Delft, The Netherlands
p.j.garbacki@tudelft.nl

Vijay K. Naik [*]
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA
vkn@us.ibm.com

Abstract

We describe a hybrid linear programming (LP) and evolutionary algorithm (EA) based resource matcher suitable for heterogeneous grid environments. The hybrid matcher adopts the iterative approach of the EA methods to perform a goal oriented search over the solution space and, within each iteration, uses the LP method to solve a partial resource matching problem. By judiciously controlling the partial problem size and its complexity, the hybrid matcher balances the accuracy of the solution and the execution time. We describe a grid management architecture that incorporates the hybrid resource matcher. Performance results indicate that the execution time of the hybrid matcher, under a variety of conditions, is at least as good and often significantly better than the execution time of LP and EA based matchers. The hybrid matcher is found to scale well with the complexity of the problem and to maintain sensitivity to the response time constraints of on-line environments.

1. Introduction

In this paper, we consider the resource matching problem arising in grid environments where resources tend to be heterogeneous, requests call for multiple types of resources, and system managers expect resource sharing, load balancing, high resource utilization and/or throughput. While the resource matching problem is known to be NP-complete [6], for grid environments, efficient on-line resource matching algorithms that are sensitive to response time constraints and can quickly adapt to changes in the system are highly desirable.

In our earlier work, we modeled resource matching as an optimization problem and solved it using linear programming (LP) based methods [9]. While we found the LP-based approach highly effective in providing optimal results, we observed two drawbacks of this approach: (i) it does not provide ability to externally control its response time and (ii) modeling new constraints or objectives is non-trivial. We also considered an Evolutionary Algorithm (EA) based approach called *evolutionary matching* [8]. We found this approach to be responsive to time constraints and intuitive to model a variety of problems. However, the evolutionary matcher does not guarantee solution optimality and, in some cases, the convergence rate can be too low.

In this paper, we propose a hybrid resource matcher that retains the pros of the LP and EA based approaches, while minimizing their shortcomings. The hybrid matcher adopts the iterative focused random search techniques from the EA approach and, within each iteration, adopts the direct solution technique from the LP approach.

The contributions of this paper are as follows. We describe the hybrid resource matcher in detail. To our knowledge, this is the first attempt to combine an iterative search technique such as the EA with a direct optimization method such as the LP for on-line resource matching. We describe an implementation of on-line resource matching service. We provide extensive experimental results to compare the performance of the hybrid matcher with those of LP and EA based resource matchers.

This discussion is organized as follows. In the next section, we formalize the resource matching problem considered here. In Section 3, we present a detailed description of the hybrid resource matcher. A grid management architecture incorporating the hybrid resource matcher is discussed in Section 4. The experimental setup and performance results are presented in Section 5. Section 6 describes the related work and Section 7 concludes the paper.

2. The problem of grid resource matching

Problem statement. We define the resource-matching problem as the process of systematically matching resources with requests to achieve specific global objectives while conforming to request and resource specific requirements, constraints, preferences, and policies. The problem

[*]Corresponding author.

considered here is general as a result of the models used to define resources, resource requests, and the associated policies, which we explain in the following.

Resource model. Resources in the grid environment are categorized by *resource types*. Examples of resource types are servers, file systems, databases, etc. Each resource type has one or more *static attributes* and zero or more *dynamic attributes*. For example, a resource type "server" may have the following static attributes: host name, CPU speed, number of CPUs, OS name, and so on. Examples of dynamic attributes of a resource type "server" are: current CPU load, memory usage, available disk space, etc. Dynamic attributes are associated with *capacities* that get consumed by resource requests or jobs. Capacity consumption by multiple requests results in resource sharing.

Request model. A job is a request for resources, which are expressed as a *set of dependencies* on one or more resource types. Each dependency places one or more *constraints* on the attribute values of a specific resource type. For example, a job may depend on a resource of type "server" with CPU speed of at least 600MHz and memory of at least 1GB. A job may also specify preferences. *Preferences* provide selection criteria when multiple resource sets satisfy dependencies associated with a job. A job may specify its preferences either by providing a method of ordering qualifying resources or by simply identifying specific resource instances by attribute value or by name. Finally, each job defines the expected *capacity usage* values for the dynamic attributes of the consumed resources. The dependencies, constraints, preferences, and capacity usage values together constitute the requirements of a job. Throughout the paper we use the terms resource request and job interchangeably.

Policies. The overall use of resources by jobs and arbitration among jobs competing for resources are governed by system-wide policies or *objectives* set by site administrators. These system wide policies guide the selection of resources for matching with jobs. Some examples of such policies are: (i) maximize throughput, (ii) maximize prioritized throughput, (iii) load balance, (iv) minimize the number of resources matched, and (v) match high priority jobs with high performance resources. A good resource matcher matches jobs with resources so that an optimal value of the objective function is achieved without violating any constraints.

Optimal solution. An assignment of jobs to resources is a *feasible solution* if no job or resource specific constraints are violated. In a feasible solution, not all jobs need to be matched with required resources. Obviously, there usually exist more than one feasible solution, but not all of them are optimal. To determine the optimal solution, a resource matcher needs to explicitly or implicitly compare the quality of the feasible solutions against one another. The quality of any two solutions can be compared by using their objective function values. The preferred solution is the one with the higher value of the objective function. The resource matching can be then defined as an optimization problem of finding a feasible assignment of jobs to resources that maximizes the value of an objective function.

Algorithm 1: Hybrid resource matcher pseudocode.

```
1  initialize the population of solutions
2  repeat
   begin
3      perturb the population
4      construct a partial problem
5      solve the partial problem using LP approach
6      reconstruct the complete solution
7      evaluate solution objective function values
8      select solutions for the next generation
   end
9  until the termination condition is met
10 return the best solution found so far
```

3. Hybrid resource matcher

In this section, we describe a *hybrid resource matcher* that combines two different resource matching techniques: the linear programming (LP) based approach and the Evolutionary Algorithm (EA) based approach. The hybrid matcher employs an EA approach to divide the resource matching problem into partial problems, then invokes LP to solve these problems and reconstructs the complete solution by combining selected solutions of the partial problems. In the following, we describe the EA and LP components of the hybrid matcher separately.

3.1. EA component

We build the skeleton of the hybrid matcher on an Evolutionary Algorithm (EA) [3]. EAs are optimization methods inspired by the nature. EAs perform a focused random search by simulating an evolution of a population of solutions. As described in Section 2, a solution in the context of the resource matching problem represents a feasible, but not necessarily optimal, assignment of jobs to resources.

The high-level pseudocode of the hybrid resource matcher is shown in Algorithm 1. The resource matcher maintains a constant-size population of feasible solutions that are computed taking into account job-, resource-, and policy-specific constraints. The resource matching starts with an initial population of solutions (line 1). In [8], we discuss several alternative initialization methods. The hybrid matcher follows the iterative procedure of the evolutionary resource matcher (line 2). In each iteration, the population is perturbed by removing a fixed number of job-to-resource assignments, effectively releasing the resource capacities reserved for those assignments (line 3). This can open up opportunities for new assignments that were not possible before. In the normal EA based resource matcher, after the perturbation step, a resource matching problem for all unmatched jobs is constructed and solved. In the hybrid matcher, the complete resource matching problem is divided into *partial problems* by randomly selecting a fixed number of unmatched jobs and a number of available resources (line 4). Each partial problem is then solved in line 5 using an LP approach described in Section 3.2. The partial solution is added back to construct a new feasible solution in the current population (line 6).

The rest of the hybrid matcher follows similar structure as a normal evolutionary matcher. First, the solutions in the population are evaluated and the corresponding objective function values are computed (line 7). A subset of solutions with objective function values above a cutoff value is selected to form the next generation of solutions (line 8). A small number of lower-quality solutions are also added to the population to enable exploration capability. The iterations are repeated until a termination criteria is met (line 9). Some examples of termination criteria are: (i) a predefined number of iterations, (ii) no improvement in the objective value over a number of consecutive iterations, (iii) a maximal execution time has elapsed. The matcher keeps track of the best solution which is returned as the result of the matching process (line 10).

3.2. LP component

Linear programming (LP) [12] is a popular technique of solving optimization problems. LP models the optimization problem as a set of linear expressions composed of input parameters and output variables. The LP solver starts by creating a problem instance of the model by assigning values to the input parameters. The problem instance is then subjected to an objective function, which is also required to be a linear expression. The values of the output variables, which collectively represent the optimal solution, are determined by maximizing the value of the objective function.

We apply the linear programming technique to solve the partial problems constructed in line 5 in Algorithm 1.

3.2.1. Notation. The linear program takes the following input parameters describing jobs and grid resources:

J is the set of jobs,

R is the set of resources,

T is the set of resource types,

A is the set of dynamic resource attributes. Attributes are unique across resource types — resources of different types are assigned distinct attributes,

P_j is the priority, the higher the better, of job j, $j \in J$,

$N_{(r,a)}$ is the capacity of attribute a, $a \in A$ of resource r, $r \in R$,

$U_{(j,a)}$ is the capacity of attribute a, $a \in A$ consumed by job j, $j \in J$,

$E_{(j,t)}$ is the set of resources of type t, $t \in T$, with static attribute values satisfying requirements of job j, $j \in J$.

The above set of parameters provides a formal description of the considered grid environment. In this environment the properties of grid resources are denoted by specifying capacities of their attributes ($N_{(.,.)}$). E.g., a resource representing a server may define capacities for the attributes describing available memory and CPU cycles. Jobs describe dependencies on resources by specifying the consumptions of the attribute capacities ($U_{(.,.)}$). In addition to resource capacity consumptions, a job specifies the required values of the static resource attributes. E.g., some jobs can be executed only on a server located in a certain network domain. Based on the observation that resources with required values of the static attributes can be identified in a preprocessing step, we do not include the static-attributes-related requirements in the set of input parameters. Instead, we define for each job and dependent resource type a set of resource instances of that type with required values of the static attributes ($E_{(.,.)}$).

In addition to the input parameters, we define a set of output variables that store the solution of the matching problem:

$X_{(j,r)}$ is a 0/1 variable equal to 1 if job j, $j \in J$ is matched to resource r, $r \in R$, and equal to 0 otherwise,

Z_j is a 0/1 variable equal to 1 if job j, $j \in J$ is matched to its required resources, and equal to 0 otherwise.

The value of variable $X_{(.,.)}$ indicates if a job is matched to a resource, and the value of $Z_.$ indicates if a job gets matched to all required resources.

3.2.2. Constraints. The feasibility of a particular assignment of values to the output variables in the context of the resource matching problem is determined by a set of constraints. All constraints in the linear program have to be linear expressions.

Gang match constraints ensure that we either match a job with all required resources or we do not match the job with any resource at all. We formalize the gang match constraints for each job j and dependent resource type t as a set of linear equations:

$$\sum_{r \in E_{(j,t)}} X_{(j,r)} = Z_j.$$

These equations guarantee that at most one resource is matched per dependency and none are matched unless all dependencies of a job are satisfied.

Resource capacity constraints guarantee that the consumption of the resource capacities does not exceed the total available capacity declared by the resources for their dynamic attributes. Given a dynamic resource attribute a and a job j, job j will consume $U_{(j,a)}$ of the capacity of attribute a

if this job is matched to the resource with this attribute and consume 0 otherwise. Hence, the resource capacity constraints translate to the following set of linear expressions defined for each resource r, $r \in R$ and its dynamic attribute a, $a \in A$:

$$\sum_{j \in J} U_{(j,a)} * X_{(j,r)} \leq N_{(r,a)}.$$

For a matching to be feasible, all the gang matching constraints and resource capacity constraints must be simultaneously satisfied.

3.2.3. Objective functions. The objective function defines the quality of a matching when multiple feasible solutions exist. LP uses the objective function to select the optimal matching. We model three objective functions.

Throughput objective function is defined as the number of jobs matched. Maximizing the value of this objection function leads to a matching that satisfies as many jobs as possible. We formalize the throughput as a linear expression $\sum_{j \in J} Z_j$.

Prioritized throughput takes additionally into account job priorities by assigning weights to the elements of the throughput summation $\sum_{j \in J} P_j * Z_j$.

Load balance objective function is defined as the fraction of the unused capacity on the most heavily loaded resources. Trying to maximize the value of this objective function, the matching process will move jobs from the most heavily loaded resources to less loaded resources and reach some kind of load balance. We define a new variable G_a for each dynamic attribute a to represent the fraction of used capacity of this attribute. We modify the linear inequalities for resource capacity constraints introduced in Section 3.2.2 to calculate the value of variable $G_.$:

$$\sum_{j \in J} U_{(j,a)} * X_{(j,r)} = N_{(r,a)} * G_a, \quad G_a \geq 0, \quad G_a \leq 1.$$

We then define another variable F to represent the fraction of unused capacity on the most heavily loaded resources by the following set of inequalities:

$$F \leq 1 - G_a,$$

for each dynamic attribute a, $a \in A$. The load balance objective is achieved when F is maximized.

3.3. Discussion

We have shown in our previous work [9, 8] that LP and EA can be applied to the problem of on-line grid resource matching as stand-alone matchers. By combining LP and EA in the hybrid approach, we have built a resource matcher that inherits their desired properties while reducing some of their limitations. We discuss how the properties of the hybrid matcher compare to the properties of stand-alone LP and EA matchers[†].

The deterministic nature of LP guarantees that the computed solution always represents the optimal allocation of jobs to resources. In contrast to LP, EA explores the solution space using a goal oriented random search. Asymptotically, EA can be shown to converges to the optimal solution [3]. However, it is impossible to determine if the best solution found by EA is an optimal solution.

In case of the hybrid matcher, the optimality of the job-to-resource assignment is a configuration issue. When the partial problem solved by LP is equal to the complete problem, the hybrid matcher gives the optimal solution. Smaller sizes of the partial problems may result in a suboptimal solution.

The linear program solvers operate on an abstract level of mathematical expressions. It is generally unlikely that an arbitrary intermediate state of the linear solver represents a feasible solution of the resource matching problem. The only feasible solution produced by the LP matcher is the optimal solution. Keeping in mind that the problem of resource matching is NP-hard, it may take LP an exponential amount of time to compute the optimal solution. The EA matcher explores the solution space by gradually improving the quality of the current best solution. Therefore, the execution of EA can be stopped practically at any point and still produce a feasible solution that approximates the optimal resource matching. Thus, the EA matcher has the sensitivity to the response time constraints and provides the ability to control the execution time of the matching process.

The hybrid matcher preserves the incremental solution improvement property of the evolutionary matcher. The partial problem is defined such that a solution of the partial problem is automatically a feasible solution. Two elements of the hybrid matcher design result in its scalability. First, the partial problem size can be used to control the complexity of the work delegated to the embedded LP matcher. The larger the partial problem, the higher the complexity of each invocation of the LP matcher. Second, keeping track of the best solution found so far allows the matching process to be stopped after any number of iterations and still get an approximation of the optimal solution.

A well designed resource matcher should be flexible and easy to extend. During the course of its deployment, the business goals may change resulting in different objective functions for the matcher. Similarly the workload and resource characteristics may change. Formulating a general grid resource matching problem in terms of linear expressions is not trivial. The level of abstraction of the linear model makes it difficult to modify and extend. EA is much more flexible in this respect. Alternative objective functions

[†]From now on while referring to LP or EA matcher we mean a stand-alone resource matcher, not the LP or EA component of the hybrid matcher

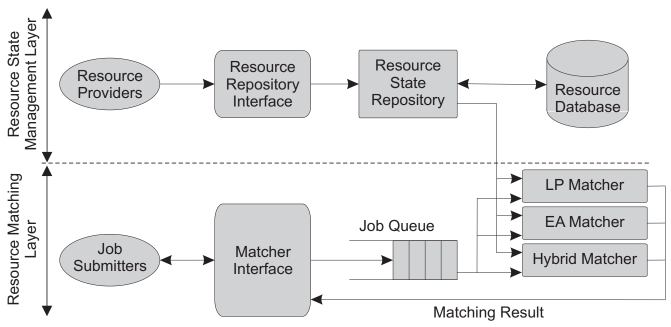

Figure 1. Architecture of the on-line matching system supporting the LP, EA, and hybrid resource matching functionality.

can be implemented in a high-level programming language or even plugged into the EA matcher in a form of external black-boxes rating solutions in a population according to some internally unknown criterion.

At first, it may seem that modifying the hybrid matcher in response to changes in the resource matching model is at least as difficult as modifying the stand-alone LP matcher. However, in many cases the change in the resource matching model can be addressed without altering the embedded LP matcher. Minor changes in the objective function can be usually handled by modifying only the solution evaluation procedure in line 7 of Algorithm 1. It is quite common that the objective function (e.g., maximize the prioritized throughput) is composed of the major objective (maximize the number of matched jobs) and a minor objective (maximize the sum of the priorities of the matched jobs). A major objective can be then hard-coded in the embedded LP matcher leaving the minor objective optimization for the solution selection procedure. Introducing an alternative minor objective (e.g., consider job preferences on resources instead of job priorities) can be done transparently to the LP matcher core.

4. Architecture of the on-line Resource Matcher

We have designed and implemented an on-line resource matching system, referred to as the *Resource Matcher*, that supports the stand-alone LP, the stand-alone EA, and the hybrid resource matching approaches. Inputs to Resource Matcher are: the resource requirements for a batch of jobs, job priorities and preferences, current resource states, resource and site specific policies. Figure 1 shows the architecture of Resource Matcher. The components of Resource Matcher can be divided into two functional layers: the Resource State Management Layer and the Resource Matching Layer.

Resource State Management Layer. This layer keeps track of the current state of each resource available for job execution. *Resource providers* specify the static resource attributes and the current values of dynamic attributes. An example of a resource provider is a site administrator, who may specify for a resource instance its resource type, static attribute values, and sharing policies for that resource. The available capacities of the dynamic attributes are updated periodically to reflect the current resource usage. Usually the update is performed by a resource usage monitor, which may also act as a resource provider.

The resource attribute information is input to the Resource State Management Layer as XML documents by making a Web service call to the *Resource Repository Interface*. Resource Repository Interface, which is a Web service deployed on an IBM WebSphere application server, translates the resource specific information from the XML documents to the internal data format of the *Resource State Repository*. Resource State Repository maintains the state and other resource specific information for all resources in the *Resource Database* running on an IBM DB2 engine. In addition, Resource State Repository performs job requirement-specific intelligent query parsing and query optimization by caching information acquired from the database.

Resource Matching Layer. The components in this layer perform the actual matching of jobs with resources. We have implemented all three resource matchers: the stand-alone LP, the stand-alone EA, and the hybrid resource matchers. The details of the stand-alone LP and EA implementations are presented in [9] and [8], respectively. The EA component of the hybrid matcher has been written in the Java programming language. The LP model is expressed in GNU MathProg language, which is a subset of AMPL [5], a well established standard among LP languages. The LP solving functionality is provided by the open source GNU Linear Programming Kit [1].

Job submitters send jobs asynchronously by invoking the *Matcher Interface*, a Web service deployed on an IBM WebSphere application server. The resource requirements, consumptions and preferences of the submitted jobs are described in XML. The Matcher Interface translates the job descriptions to the internal Resource Matcher data structures and, after the matching is finished, performs a reverse operation of creating and sending an XML document describing the matched resources to the job submitters.

Job Queue caches all arriving jobs if the resource matcher is busy processing jobs that arrived earlier. Depending on the current system configuration, the LP, EA or hybrid resource matcher is executed. The executed matcher consults Resource State Repository to obtain the current state of all resources relevant to the batch of jobs waiting in the Job Queue.

Resource Type	Static Attrs.	Dynamic Attrs.	Instances
server	CPU architecture, # CPUs, domain	utilization, memory	50
database	vendor	connections	50
network	IP, protocol	bandwidth	50
file storage	filesystem	size	50

Table 1. Resource model.

5. Performance evaluation

In this section we present results of an experimental study evaluating the performance of the hybrid resource matcher and comparing it with the performance of the stand-alone LP and EA matchers.

5.1. Experimental setup

We used real-world traces from a deployed infrastructure to represent the grid resources in our experiments. For this we used the resource statistics provided by the Server Resource Management (SRM) [2] system which reports historical and near real time trends of resources serviced by IBM. Our experimental grid environment consists of 200 resources divided into four types: server, database, network, and file storage. Each resource type is assigned one or two dynamic attributes and one, two or three static attributes. The detailed information on the resources and their attributes is presented in Table 1.

The workload used in the evaluation is generated synthetically. Job priorities are selected randomly and uniformly from the range $1, \ldots, 10$. For each job we select the dependent resource types making sure that each job depends on at least one resource type. The dependency of a job on a resource type is determined by Bernoulli distribution with the probability of success equal to the value of the *complexity parameter* that we vary throughout the experiments. After the dependent resource types have been chosen, a set of dependent attributes for each of these types is selected. Each job selects one or more dynamic attributes and one or more static attributes of the dependent resource type. Also at this stage the selection is performed according to a Bernoulli distribution with the probability of success equal to the complexity parameter. The required value of the dependent static attribute is selected randomly and uniformly from the set of available values of this attribute. The minimal required value of the dynamic attribute is selected randomly and uniformly from the interval bounded by 0 and the maximal available value of that attribute among the defined resources.

We perform experiments for all three objective functions defined in Section 3. The EA resource matcher is configured to terminate its execution if one of the following conditions is satisfied: (i) 1,000 iterations has been performed, (ii) the current best solution has not been improved during 500 consecutive iterations, (iii) the total execution time has exceeded 25,000 seconds. The hybrid matcher adopts the termination conditions of EA matcher only changing the maximum allowed number of iterations to 100 and the maximum number of iterations without improvement to 50. The size of the partial problem solved in one iteration of the hybrid matcher equals 5.

We used several machines to deploy the components of the on-line Resource Matcher described in Section 4. The Resource Repository Interface, Resource State Repository, Matcher Interface, stand-alone EA matcher, and EA component of the hybrid matcher run on a Windows 2000, dual Xeon 2.6GHz CPU, 3GB RAM machine. A machine with an analogous configuration hosts the Resource Database. The stand-alone LP matcher and LP component of the hybrid matcher run on a RedHat EL3, dual Xeon 2.6GHz CPU, 3GB RAM machine.

5.2. Results of the experiments

To compare the performance of the three resource matchers, we measured the execution time and the final solution quality using the LP, EA, and the hybrid resource matchers for a range workloads and the experimental setup described above. We discuss the results in the following.

5.2.1. Execution time. In the first set of experiments we measure the execution time of the EA, LP, and hybrid resource matchers for the three objective functions defined in Section 3. For each objective function and each resource matcher we perform four series of experiments for the values of the complexity parameter varying from 0.1 to 0.5 with step 0.1. After the value of the complexity parameter is selected, we vary the number of jobs sent to the resource matcher in a single batch. The batch size is one of 10, 20, 30, or 40.

Table 2 shows the execution time of the resource matchers for the throughput maximization, prioritized throughput maximization and load balance objective functions. In all experiments, the EA resource matcher required more time to find the matching than any of the other two matchers. Clearly, the objective most difficult to optimize is the load balance. The relative increase in the execution time of EA and LP when the throughput is replaced with the prioritized throughput and then the load balance objective is more significant than in case of the hybrid matcher. The execution time of the LP matcher is not affected much by changing the objective from the throughput to the prioritized throughput. However, setting the objective to load balance increases the execution time of LP by a factor of more than two compared to the other objective functions. This suggests that LP matcher is much more sensitive to the form of the objective function than the alternative approaches. With exception of the lowest-complexity problems, for which the LP matcher achieves best results, the hybrid matcher outperforms the remaining matchers. The hybrid resource matcher

Complexity		0.1				0.2				0.3				0.4				0.5			
Batch size		10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40
Maximize throughput	**Hybrid**	3	6	11	12	4	6	10	11	4	13	50	74	3	21	71	140	6	52	124	214
	LP	1	6	11	14	2	6	15	20	2	8	149	350	1	13	331	476	2	109	513	800
	EA	7	134	201	298	14	245	415	1822	25	374	1726	1999	661	2752	3665	4384	8020	9114	13822	25000
Maximize prioritized throughput	**Hybrid**	3	7	14	14	5	8	13	13	5	16	62	90	4	26	86	176	8	65	152	276
	LP	2	8	13	17	2	8	17	24	3	10	184	454	2	15	420	618	3	131	661	1032
	EA	10	193	285	417	20	352	615	2639	36	558	2470	2959	936	4123	5354	6401	11780	13308	20661	25000
Load balance	**Hybrid**	4	8	17	18	6	9	16	17	6	20	78	115	5	32	111	218	9	79	195	338
	LP	4	19	33	41	5	18	42	60	7	25	454	1058	4	39	1022	1455	6	337	1565	2437
	EA	12	232	361	508	25	425	720	3252	44	643	3039	3424	1143	4698	6457	7576	13759	16068	25000	25000

Table 2. Execution times in seconds of the hybrid, LP, and EA resource matchers configured with three alternative objective functions and applied to problems of different complexities and sizes.

scales much better with the problem complexity and size than any of the other two matchers. The significant improvement in the execution time achieved by the hybrid matcher is extremely important for the on-line aspect of the resource matching that imposes constraints on the maximum allowed execution time. We also note here that, compared to the other two matchers, the execution time of the hybrid matcher grows much more slowly as the complexity and/or the batch size is increased. This is an important desirable property of a resource matcher in a grid environment where changes in workload or resource availability tend to be clustered and not gradual.

5.2.2. Quality of matching. In this section, we compare the quality of hybrid matcher solution with that obtainted using the LP and EA based approaches. We use the value of the objective function as a measure of the solution quality.

Figure 2 shows the objective values corresponding to the final solutions found by the three resource matchers relative to the maximal objective value. The solution computed by the LP matcher is by definition optimal. For throughput maximization, the quality of the solution found by the EA matcher in all cases but one is better or equal to the quality of the solution computed by the hybrid matcher. Only for the complexity parameter equal to 0.5 and the batch size equal to 40, the EA matcher exceeds the allowed execution time and has to be terminated resulting in a lower quality solution. The advantages of the hybrid approach over the EA matcher become visible for more complicated objective functions. A significant difference in the quality of the matching between the hybrid and EA approaches is apparent in case of the load balance objective.

6. Related work

Condor system uses classified advertisement (ClassAd) framework for solving resource allocation problem in a distributed environment with decentralized ownership of resources [10]. The original matchmaking framework only allows a request to be matched with a single ClassAd. In [11], the matchmaking mechanism of Condor has been extended to gangmatching for co-allocation. The running example in [11] is the inclusion of a software license in a match of a job and a machine which is a simpler problem than the general grid resource matching problem introduced in Section 2.

The Redline matching system [7] proposes a matchmaking system more expressive than Condor ClassAd. In Redline, the matching problem is transformed into a constraint satisfaction problem. The constraints are checked to make sure that no conflicts occur and one of the existing constraint solving technologies (such as LP) are used to solve the transformed problem. A variant of hybrid matcher could be used to solve the constraints generated by Redline.

Globus [4] defines an architecture for resource management of autonomous distributed systems with provisions for policy extensibility and co-allocation. Users describe required resources through a resource specification language (RSL) that is based on a pre-defined schema of the resources database. Globus provides extendible APIs to perform sophisticated co-allocation which could be implemented by the hybrid resource matcher.

7. Conclusions

In this paper, we have addressed the problem of efficient resource matching in heterogenous gird environments. Based on the observation that the popular approaches to resource matching, the EA and LP approaches, are in a way complementary, we have designed a hybrid resource matcher that combines EA with LP. The hybrid resource matcher inherits the flexibility of EA allowing the matching process to be stopped practically at any point, and the high matching accuracy of LP. Those properties of the hybrid matcher, confirmed in a series of experiments on a trace-based system model, are highly desired in an on-line environment.

The novelty of the hybrid matcher is in iteratively refining the solution using the LP to solve the partial problems accurately and using the EA to guide the inter-iteration flow. This approach is generic and independent of the optimization problem itself. We believe that the hybrid optimization method can be applied in a broader scope than the particular problem of resource matching in grid environments. The

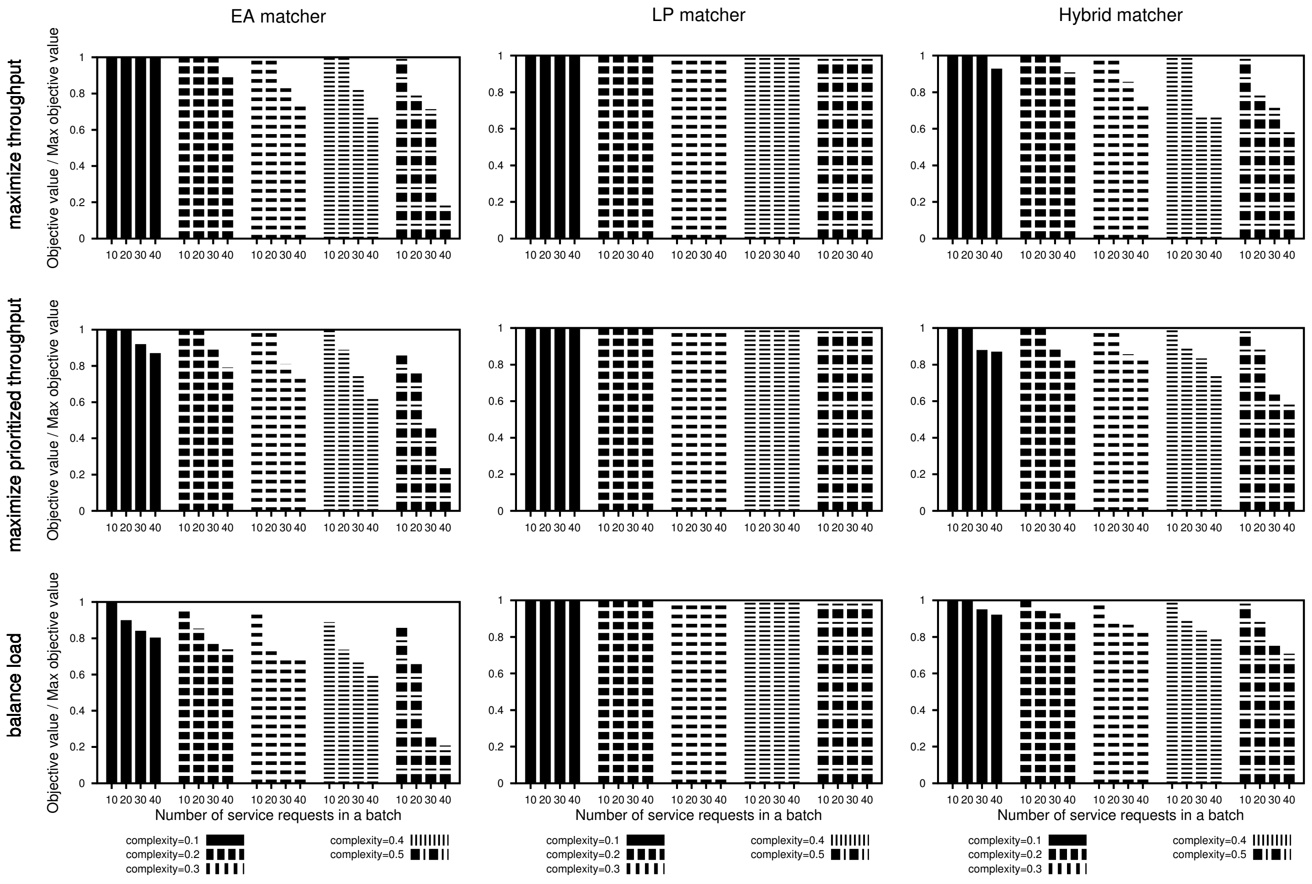

Figure 2. Objective function value of the best solution relative to the maximal objective value found by the three resource matchers maximizing the throughput, maximizing the prioritized throughput, and balancing the load.

hybrid optimization method is not superior in all the aspects to LP and EA, but it is rather an alternative. The hybrid approach is more complex to implement than EA and requires more parameter tuning than LP, e.g., the partial problem selection criterion. Those limitations open possibilities for the future work. In particular, an interesting question is whether the partial problem should be selected based on the objective function.

References

[1] GNU linear programming kit page. http://www.gnu.org/software/glpk/.

[2] IBM SRM page. https://srm.raleigh.ibm.com.

[3] T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Institute of Physics; Ringbound, April 1997.

[4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[5] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, second edition, November 2002.

[6] O. Ibarra and C. Kim. Heuristic algorithm for scheduling independent tasks on nonidentical processors. *J. of the ACM*, 24:280–289, 1977.

[7] C. Liu and I. Foster. A constraint language approach to matchmaking. In *14th IEEE International Workshop on Research Issues on Data Engineering (RIDE 2004)*, Boston, MA, March 2004.

[8] V. Naik, P. Garbacki, K. Kummamuru, and Y. Zhao. On-line evolutionary resource matching for job scheduling in heterogeneous grid environments. In *Proceedings ICPADS'06*, Minneapolis, MN, July 2006.

[9] V. K. Naik, C. Liu, L. Yang, and J. Wagner. Online resource matching for heterogeneous grid environments. In *Proceedings CCGRID'05*, Cardiff, UK, May 2005.

[10] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC-7*, Chicago, IL, July 1998.

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

[12] A. Schrijver. *Theory of Linear and Integer Programming*. June 1998.