**NOTICE:** This is the author's version of a work that was accepted for publication in Future Generation Computer Systems. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication.

A definitive version was subsequently published in Future Generation Computer Systems, Volume 27, Issue 8, October 2011, Pages 999–1010, DOI: [http://dx.doi.org/10.1016/j.future.2011.05.002](http://dx.doi.org/10.1016/j.future.2011.05.002)
Agents, Clusters and Components: 
A Synergistic Approach to the GSP

G. Kakarontzas\textsuperscript{a}, I. K. Savvas\textsuperscript{b,*}, I. Stamelos\textsuperscript{c}

\textsuperscript{a}Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece, and Department of Informatics and Telecommunications, TEI of Larissa, 41110 Larissa, Greece

\textsuperscript{b}Department of Informatics and Telecommunications, TEI of Larissa, 41110 Larissa, Greece

\textsuperscript{c}Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

\textsuperscript{*}Corresponding author, Tel.:+302410684522
Email addresses: gkakaron@teilar.gr (G. Kakarontzas), savvas@teilar.gr (I. K. Savvas), stamelos@csd.auth.gr (I. Stamelos)

Abstract

Grids provide access to vast amounts of computational resources for the execution of demanding computations. These resources are geographically distributed, owned by different organizations and are vastly heterogeneous. The aforementioned factors introduce uncertainty in all phases of a Grid Scheduling Process (GSP). This work describes a synergistic multidisciplinary approach which aims at addressing this uncertainty. It proposes a network of resource representatives (RP), which maintain the more or less static characteristics of available workers they represent. Clustering techniques are used for the efficient searching in the network of RPs by client agents. After the discovery of possibly suitable resources, client agents and resource agents negotiate directly for the selection of the best available resource set. Finally according to the characteristics of the selected resource set and its
current state, we propose a component-based application configuration approach based on component variants, that adjusts the application for the forthcoming execution phase in the selected resource set. We evaluate our approach using simulation and we show that it outperforms centralized index approaches for large computational grids.

**Keywords:** Grid Scheduling Process, Software Agents, Clustering, Software Components

### 1. Introduction

Grids provide access to vast amounts of computational resources for the execution of demanding computations. These resources are geographically distributed, owned by different organizations and are vastly heterogeneous. The aforementioned factors introduce uncertainty in all phases of grid scheduling. A Grid Scheduling Process (GSP) has been proposed in [1] comprising three phases (Fig. 1).

![Figure 1: The three phases of the Grid Scheduling Process](image)

In the *resource discovery* phase, the available resources are filtered so that only resources suitable for the application are considered for the following phases. This phase includes the sub-phases of *Authorization Filtering*, *Application Definition* and *Minimum Requirements Filtering*. Essentially the filtering of the resource discovery phase is concerned with mostly static characteristics of available resources such as the CPU type and the memory.
capacity of resources. The input to the resource discovery phase is a suitable
description of the static resource requirements for a computation such as
those described by the Info node of a ComputingElement in the GLUE In-
formation Model [2], and the output is a set of potentially suitable resources.
We use the term “potentially”, since the actual suitability of a resource is de-
determined during the next phase of the resource selection. For example if the
compute node is an opportunistic batch system, it might be the case that the
batch system is unavailable or that the actual compute nodes required are
unavailable at the moment. To give an analogy we view the phase of resource
discovery similar to that of an Internet search where a number of potentially
suitable references is returned, but these references might be unavailable due
to network failures or outdated URLs. The second phase of resource selection
determines the actual suitability of a resource at the time of the selection
process, as well as its agreement to participate in the forthcoming execution
phase. It comprises the sub-phases of Dynamic Information Gathering and
System Selection. The input in this phase is the discovered resource set of the
previous phase, and the current status of these resources is used to determine
the most appropriate resource for the execution of the computation’s tasks.
Finally the Job Execution phase is concerned with the execution of the com-
putation in the selected resource set and includes the sub-phases of Advance
Reservation, Job Submission, Preparation of tasks, Progress Monitoring, Job
Completion, and Cleanup. Essentially it includes all these activities which
are related to the successful completion of a job in a possibly remote compute
node.

In this work we consider the issue of the GSP in the general case of dy-
namically formed grids, as in peer-to-peer grids [3], in which a preexisting virtual organization of which the grid user is a member of does not exist. Since the number, type and availability of resources in this general case is a priori unknown, there are several difficulties that need to be addressed. First there is a need for a simple process that will allow users to discover suitable resources from a potentially huge number of resources from which the majority might be unsuitable and consequently the resource discovery process should be scalable to large search spaces. Second, the lack of a previously established virtual organization dictates that the formation of the execution environment might require both time and negotiation skills and therefore the underlying infrastructure should provide both pro-active and intelligent behavior especially during the resource selection phase in which the interests of both parties (users and providers) should be taken into consideration. Third, the uncertainty on the type of discovered resources requires that the application should be prepared for a number of different execution environments. This work addresses these issues by unifying under a common GSP architectural framework several different technologies and approaches that we have considered in our previous works [4, 5, 6]. More specifically on the first issue of scalability we propose the use of cluster-based techniques. The fact that the resource discovery phase filters the list of resources for the computation based on static characteristics, creates the opportunity for an efficient clustering of resources and the subsequent use of clustering algorithms [7, 8, 9] that greatly reduces the computational demand for the resource discovery phase. On the issue of proactive and intelligent behavior we propose the use of agents [10]. Software agents possess many characteris-
tics which make them ideal for the representation of both grid users and grid resource providers during resource discovery and selection, including goal orientation, autonomy, temporal continuity and communication skills. Finally for the preparation of the application for different execution environments component-based application development [11] provides a very promising approach. Since component-based applications are formed dynamically during runtime with dynamic binding of the appropriate software components, components can be replaced as required to fit the actual execution environment which is unforeseen during the initial job submission. Of course multiple variants of components will be required but in this work we provide a design approach that can be followed to achieve this.

The organization of the rest of the paper is as follows. In Sec. 2 we provide related work. In Sec. 3 we describe the general architecture of the proposed framework as well as the role of software agents in this architecture. In the following section 4 we discuss the issue of the clustering of resources and how this can be used effectively during grid resource discovery while in Sec. 5 we present the experimental results which evaluate our proposed techniques. Next in Sec. 6 we provide a design approach for grid application development that is component-based and uses the notion of quality variants of the same functional component showing how this approach can be used effectively for the configuration of a grid application prior to the execution phase of GSP. Finally in Sec. 7 we provide future research directions and conclude.
2. Related work

The idea of matchmaking independent service providers and owners is an old one and many projects have adopted a similar more or less idea to the one presented in this work.

The Condor High-Throughput Computing System [12] provides a matchmaker agent which matches classified advertisements of resource providers to those of clients [13]. The matchmaking process in Condor has two phases: in the first phase a match is produced by the matchmaker agent and the respective parties are notified and in the second phase the two parties interact directly to confirm the match based on the actual status of the resource at the time of the negotiation. Similar to condor matchmaking, our proposed system has also two phases. Differently than Condor, where dynamic information such as the current state of a machine is reported in the classad, we propose that the first phase is only concerned with static characteristics of resources since absolute information on the state of resources is hard to maintain in global scale and would result in bloating off even the most powerful matchmaker. We understand that since the absolutely latest condition of the resource is not available to grid’s resource representatives the possibility of successful matches is reduced. However the first phase in our process is not concerned with the actual matchmaking but with the generation of a pool of possible resources from which the client agent can then select the most suitable.

The Globus Monitoring and Discovery Services (MDS) [14] offer aggregator services which aggregate resource information produced by information providers. MDS aggregators can provide specialized and virtual organiza-
tion (VO) specific views on the data provided by information providers. The views can range from very simple, such as a name directory service that only records the names of the participating resources to more elaborated, such as a relational aggregate directory service that uses a relational database and can support relational database queries. Aggregators can form a hierarchy reflecting the VO structure. The aggregator framework of MDS is a very flexible framework with its support of multiple views on the same information and the hierarchical structure. We observe that our Resource Representatives could be easily implemented using the aggregator framework. We depart from the hierarchical structuring since we’re interested in supporting dynamic grids and not specifically VO-like grids.

The synergistic relationship between agent and grid computing has already been identified. In [15] leading scientists from both communities comment on the benefits and shortcomings of both distributed technologies and the ways that the one can fill the gaps of the other. In our proposed system we use the agents where autonomous intelligent behavior and negotiation capabilities are required and the Grid where robust infrastructure is required. Our client and local RMS agents carry out negotiations and should exhibit autonomous behavior representing the interest of their respective parties. On the other hand the Grid infrastructure developed by several projects over the past few years can be used for robust job execution and monitoring. We have already mentioned that aggregator services offered by the Globus toolkit [16] can be used as the infrastructure for the resource representatives implementation.

Component approaches for the grid are one of the most promising pro-
gramming paradigms that will undoubtedly have a profound effect on the way grid applications will be developed in the future. Grid component models such as GCM [17] and CCA [18] promise to bring simplicity to the development of grid applications. Yet grid component models are not the complete solution. We believe that similar to the enterprise computing component models in which design idioms and best practices are regularly published, the grid component models will also need to be complemented with additional material (e.g. specific grid component model patterns) that will further assist the developer by providing guidance. Our elastic component design process is a constructive approach that will enable the exploitation of this knowledge in a systematic way. Also it enables the development of grid component variants for the most effective use of the selected resource set in the grid scheduling process. The elastic component design process is not specific to any grid component model and provides, we believe, a useful complement in addition to the facilities provided by the selected grid component framework.

3. Architecture of the proposed GSP framework

The process for resource discovery and selection that we proposed in [4] clearly separates the Resource Discovery and Resource Selection phases, by assigning them in distinct agents participating in the grid. In Fig. 2 we can see a high level UML-like deployment diagram depicting the logical participants in our GSP framework architecture and their physical placement in the grid nodes.

In our architecture Local System RMS Agents represent local clusters of compute nodes running their own resource management system (such as Con-
They register with one or more Resource Representatives which will represent their resources in the grid. The registration includes information on the static characteristics of the resources in the local system, such as the number and the type of available processors. Local System RMS Agents will notify their representatives for modifications on their resources, although it is expected that these modifications will be seldom, since the provided information concerns static characteristics. The registration does not entail any obligation on behalf of the local systems that will actually accept any computation from the grid. It merely states their willingness to be considered for the subsequent resource selection phase. Local System
RMS Agents also participate in the selection phase, where they negotiate with the client agents in the context of a contract-net protocol whether they will accept tasks for execution or not.

*Resource Representatives* are linked in a network and collectively provide the resource discovery service to grid clients. A client knows about only a limited number of these resource representatives, similarly to a browser that knows about only a limited number of DNS servers. A client’s agent request to a resource representative triggers the resource discovery process on behalf of the client. If the request can be satisfied locally then the client’s agent gets immediately a set of suitable Local System RMS agents, otherwise its request is forwarded to a number of neighboring resource representatives that the first resource representative knows about which repeat the same process. Ultimately the client’s request will either be satisfied or it will timeout, in which case the client agent retries at a later time until the request is manually withdrawn by the grid user.

*Client Agents* represent in our architecture the grid users. They are given a set of task descriptions and attempt to find a number of resources to run these tasks. For each such task, client agents submit a request for a suitable resource to their resource representatives. Timeouts and maximum hop counts are used for the distributed garbage collection of failed requests. If and when a request succeeds, client agents will negotiate directly with Local System RMS agents in the forthcoming resource selection phase. Since resource discovery is potentially time consuming and resource selection requires negotiation abilities, client representatives are the most obvious choice for the application of agent technology on our GSP framework architecture.
3.1. Dynamic behavior of the components

Having discussed the static structure of the architectural elements of our GSP framework, we now discuss the collaborations of these elements for resource selection and discovery. We present the collaborations as UML sequence diagrams for the discovery and selection of resources.

In Fig. 3 we can see a UML sequence diagram depicting the resource discovery process.

![UML Sequence Diagram](image)

Figure 3: Resource discovery sequence diagram

A client sends a `submitTask` message to its resource representative for each of a computation’s tasks and at the same time creates a `Timer` object so that the request will time out in case it cannot be satisfied. The
ResourceRepresentative attempts to find a suitable resource in its local knowledge base (the matchTask method). If the matching succeeds the client is called back with the suitable ResourceInfo for the task. The ResourceInfo contains the information required so that the client can access the resource in the next phase of the resource selection. Otherwise the request is forwarded to a selected subset of neighbors of the resource representative who repeat the same process.

After the client agent has enough resources to run the computation it can negotiate directly with the resources, the resource information of which was obtained during the previous phase. This selection phase, depicted in Fig. 4, comprises two sub-phases.

![Diagram of resource selection sequence](image)

Figure 4: Resource selection sequence diagram

In the first sub-phase, a contract-net protocol [19] is carried out, with the client acting as the initiator of the protocol and the resource agents discovered during the discovery phase, acting as the participants. The purpose of this step is the reaching of an agreement between the client agent and the resource agents. This agreement might be based on a number of suitable
criteria of which the most obvious is the price that the client is willing to pay for the task, but the contract-net protocol is left intentionally open to this. The outcome of this step is a subset of resource agents that are willing to execute the task.

In the second sub-phase the client requests extended resource information from the so-called willing resources. This extended resource information includes up to date information on the static resources as well as additional data. For example one such additional element is the load index, which indicates the current load condition of the resource at the time. The extended resource information allows the client to proceed to a more accurate selection of the most suitable resource based on the static characteristic of the resources as well as to determine the more lightly loaded resource. This resource is then selected as the best resource to which the task is assigned for execution. The rest of the resource agents will receive a cancel message in accordance to the cancellation meta-protocol of the contract-net protocol.

In case of failure due to the inability to reach an agreement a client agent attempts re-selection at a later time and if this fails attempts re-discovery for the discovery of additional resources.

4. Clustering resources for grid discovery

In the previous section we mentioned that a Resource Representative first searches locally for suitable resources and then it may forward clients’ requests to a selected subset of neighboring Resource Representatives when it is unable to satisfy them locally (see Fig. 3). To increase the efficiency of the system we propose the use of clustering algorithms for the efficient local
search and the discovery of suitable ‘neighbors’.

In order to demonstrate the use of clustering algorithms for grid resource discovery we first need to formally define the concepts of our GSP framework. The Grid Scheduling Process (GSP) is divided into two principle phases, namely the resource discovery (RS) and the selection process (SP). The resource discovery phase is the initial stage where the client that applies for tasks \( t \) to be executed contacts its resource representative node \( r \) and submits the attributes that the computation demands in order to be executed on a worker node \( w \). The requirements of the task could be characterized by a vector:

\[
t_i = (a_{t_{i1}}, a_{t_{i2}}, \ldots, a_{t_{in}}).
\]  

These attributes represent the task’s requirements or example, in operating system, the desired CPU power, memories capacity, software and so on. Some of them are very crucial (primary) for the computation, while some other are rather common (secondary). For example, the CPU speed is a secondary characteristic since all worker nodes have a CPU (with different speed). On the contrary, there are attributes like the operating system or the existence of a specific software or a database which are very crucial for a computation. A task that requires a 3GHz CPU can survive with less speed but if the desired operating system is Linux cannot be executed in a Windows environment. Therefore, there are two main categories of attributes: 1) the primary \( (a_p) \), and 2) the secondary \( (a_s) \). So, the vector of Equation 1 can be extended as follows:

\[
t_i = (a_{t_{i1}}, a_{t_{i2}}, \ldots, a_{t_{ip}}, a_{t_{s1}}, a_{t_{s2}}, \ldots, a_{t_{sm}}).
\]
where \( n = l + m \). Thus, the vector of Equation 2 represents accurately the computations’ requirements. Subsequently, each worker node is characterized by exactly the same rather static attributes plus a dynamic one, namely its current load state, the load index, which we use in this discussion as an example of a dynamic attribute. Of course a number of other dynamic attributes could also be taken into account. Dynamic attributes, such as the load index, will be useful at the final phase, the post-selection phase, where the clients negotiate directly with the worker nodes. Therefore, the representative nodes need only to keep the static characteristics of the workers they represent. So, the vector

\[
w_i = (a_{w_{i1}}, a_{w_{i2}}, \ldots, a_{w_{il}}, \ldots, a_{w_{im}}, l_i)
\]

characterizes the workers, where \( l_i \) represents their load index which is specified by available local system facilities.

4.1. Workers’ Clustering

The resource representatives keep all the information regarding the resources they represent which are: 1) a set of available workers, and 2) a set of their neighbor representatives. The set of workers is actually a set consisting of the static attributes of the worker nodes,

\[
W^r_i = \{w_{i1}^r, w_{i2}^r, \ldots, w_{ix}^r\}
\]

where each worker is representing by a \( n \)-dimensional data point (Equation 2).
One of the main characteristics of the large scale resource environments is the existence of enormously large numbers of heterogeneous workers (resources). The resource representatives represent a set of workers which potentially could be very large and this makes the search procedure among this set a time consuming process. Initially, the resource representatives collect all the appropriate information of the set of workers that are keeping (Equation 4). In order to search among them effectively for workers which meet specific criteria for a given computation, the division of them to non-overlapping subsets (clusters) with “similar” attributes is a very promising technique [7, 8, 9]. On the contrary, the complexity of k-mean clustering algorithms on the set of workers $W^{r_i}$, is very high [20] but since the set is rather static it is affordable. Each element (point) of the set of workers is consisting of the static parameters ($n$) of the workers and since all these may be important for a given task, they all form the key for the clustering procedure. So, the worker set of any resource representative ($r_i$), is divided into $k_i$ clusters ($K^{r_i}$), and each one of these clusters is represented by its centroid ($c$), and radius ($d$). So, the set of workers can be denoted as a set of couples as follows:

$$W^{r_i} = \{(c_1^{r_i}, d_1^{r_i}), (c_2^{r_i}, d_2^{r_i}), \ldots, (c_{k_i}^{r_i}, d_{k_i}^{r_i})\},$$

where:

$$c_{x}^{r_i} = \frac{1}{n} \left( \sum_{k=1}^{l} a_{p_k}^{w_x} + \sum_{k=1}^{m} a_{s_k}^{w_x} \right), n = l + m$$

and
\[ d^r_i = \max_{l=1}^y(||c^r_i - w^r_i||), w^r_i \in K^r_i \]

The notation \(||c^r_i - w^r_i||\) stands for the distance between \(c^r_i\) and \(w^r_i\).

In addition to the \(k_i\) clusters, the resource representative calculates the centroid (\(C^r_i\)) and radius (\(D^r_i\)) of all clusters, considering now that the centroid of the clusters form the data points. This produces a new centroid and radius of all data points of the worker set:

\[ \bar{W}^r_i = (C^r_i, D^r_i). \]  

This new centroid is useful for the initial search of a computation’s requirements. At the beginning, the vector of the computation’s requirements tries to see if the primary characteristics of the task exist within the set of workers of its resource representative comparing them with the coordinates of the centroid of \(r_i\). If the corresponding coordinates of \(r_i\) are non zero, the task’s requirements could be satisfied of its \(r_i\) and then searches its clusters, otherwise there is no need to search among them since the primary attributes cannot be satisfied by this specific resource representative. This increases the efficiency of the searching technique since if the primary attributes are not satisfied there is no reason to search among the clusters of \(r_i\).

In addition of its worker group, each resource representative keeps a set of the neighbor resource representatives. This set is consisting of resource representatives that are “close enough” to \(r_i\). The notion of neighborhood can be extended to any distance measure (norm) by mapping the space of system factors (parameters) to a vector space [21, 22, 4]. For example, two
representative nodes can be characterized as neighbors if their Euclidean distance is less than a predefined value (i.e. limited within a geographical area) or if they can reach each other on a predefined time interval and so on. Neighborhood of a resource representative node will comprise all nodes whose norm meet the “neighbor” criteria. So, the triple:

\[ r_i = (W^{r_i}, \bar{W}^{r_i}, N(r_i)). \]  

(7)

classifies accurately each resource representative. Each node \( r_j \in N(r_i) \) does not need to contain all the information that \( r_j \) keeps, but the centroid and the radius of the worker group that \( r_j \) represents is more than efficient, \( \bar{W}^{r_j} = (C^{r_j}, D^{r_j}) \).

The resource discovery process with clustering can now be formalized as depicted in Algorithm 1.

In Algorithm 1 a client sends a computation request \( t_i \) to its representative node \( r_i \), \( t_i \rightarrow r_i \). \( r_i \) searches its workers set in order to find out if there are such elements that satisfy the task’s criteria. Precisely, firstly the proposed technique compares the global centroid of the resource representative to the task’s primary attributes. If these attributes are satisfied it continues with the clusters of \( r_i \) using the same method. First searches the corresponding centroids and radiuses and then the individual worker nodes. If there are nodes that could satisfy \( t_i \), \( r_i \) returns to the clients these workers and its job stops here. On the contrary, if no convenient workers for the submitted task are found, \( r_i \) searches among \( N(r_i) \) in order to find other (neighbor) resource representatives which may meet the task’s needs and if there are such then it transfers \( t_i \) to them. If the Resource Discovery algorithm fails to find the
Algorithm 1 Resource Discovery

1: Compare the primary attributes of $t_i$ to the global centroid and radius 
   $(C^{r_i}, D^{r_i})$ of $r_i$
2: if (the corresponding values of the primary attributes of $r_i$ are non zero) then
3:     $q \leftarrow 0$, $End \leftarrow FALSE$
4:     repeat
5:         $q \leftarrow q + 1$
6:         Compare the primary attributes of $t_i$ to the centroid and radius of 
         the $q$-th-cluster, $(c^{r_i}_q, d^{r_i}_q)$
7:         if (the corresponding values are non zero) then
8:             Determine the candidate subset of $W^C_{r_i} \subseteq W^{r_i}$ that satisfies $t_i,$
             $w^{r_i}_y : ||t_i - w^{r_i}_y|| \leq \sigma \land w^{r_i}_y \in K^{r_i}_y$ ($\sigma$ is a predefined constant)
9:             $End \leftarrow TRUE$
10:        end if
11:     until $(END)$ or $(q = k_i)$
12: if $(W^C_{r_i} \neq \emptyset)$ then
13:     Return $W^C_{r_i}$ to the client
14: end if
15: else
16: if (hop count of $t_i < MAXHOPS$) then
17:     Perform “Neighbor Resource Discovery”
18: else
19:     the request is withdrawn
20: end if
21: end if
22: End.
appropriate workers then we have an implied failure of the process, since the client will timeout the request.

At line 17 of algorithm 1 a neighbor resource discovery is initiated. For this reason, the resource representative has to keep a set of its neighbors, $N(r_i)$ in order to explore their solution space which in turn may meet the computation’s needs. The resource representatives nodes that meet the criteria are kept in the candidate set ($N^C(r_i)$). Again, the searching method is based on the primary characteristics of the neighbor resource nodes and if these satisfy the computation’s requirements the resource representative examines if the distance of the corresponding centroid from the computation’s vector is less than the radius of the neighbor resource representative. If the specific node meets both the previous criteria, it is chosen for inspection and the computation’s requirements are transferred to it. If the Neighbor Resource Discovery algorithm fails to find the appropriate workers then we have an implied failure of the process, since the client will timeout the request. Algorithm 2 describes in detail the Neighbor Resource Discovery procedure.

4.2. Concrete Example

In order to clarify the searching technique a simplified concrete example is given. Let the vector of the attributes consist of three attributes ($x, y, z$), from which one of them is secondary ($x$), i.e. CPU speed, and the other two are primary ($y, z$), i.e. $y$ for operating systems and $z$ for databases. The set of workers is consisting of the attributes as they are shown in Table 1 and in Fig. 5.

Considering that the number of clusters is 4, the application of the $k$-
Algorithm 2 Neighbor Resource Discovery

1: $N'(r_i) \leftarrow N(r_i)$
2: $N^C(r_i) = \emptyset$
3: repeat
4: Select $r_j : r_j \in N(r_i) - N'(r_i)$
5: $N'(r_i) \leftarrow N'(r_i) + \{r_j\}$
6: if ($r_j$ satisfies the primary attributes of $t_i$) then
7: if $||C^{r_j} - t_i|| \leq D^{r_j}$ then
8: $N^C(r_i) \leftarrow N^C(r_i) + \{r_j\}$
9: end if
10: end if
11: until ($N'(r_i) = \emptyset$)
12: if ($N^C(r_i) \neq \emptyset$) then
13: MAXHOPS $\leftarrow$ MAXHOPS +1
14: Transfer $t_i \rightarrow r_j : \forall r_j \in N^C(r_i)$
15: else
16: the request is withdrawn
17: end if
18: End.

Figure 5: Clusters of the workers’ set
mean algorithm produces four clusters as in Fig. 5. The four centroids with their corresponding radiuses are given in Table 2.

In addition, the resource representative calculates its global centroid and radius, which are:

\[ C = (5.75, 0.17, 0) \]
\[ D = 6.25 \]

In such a resource’s representative environment, a client submits a computation. The attributes of the computation could be one of the following three general categories:
Table 2: Centroids and radiuses.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Centroid</th>
<th>Radius</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_1$</td>
<td>(2, 0, 0)</td>
<td>1</td>
</tr>
<tr>
<td>$K_2$</td>
<td>(5, 0.33, 0)</td>
<td>1.05</td>
</tr>
<tr>
<td>$K_3$</td>
<td>(8, 0.33, 0)</td>
<td>1.05</td>
</tr>
<tr>
<td>$K_4$</td>
<td>(11, 0, 0)</td>
<td>1</td>
</tr>
</tbody>
</table>

1. The computation requires only the secondary attribute, i.e. $t_1 = (13, 0, 0)$
2. The computation requires one secondary and one primary attribute, i.e. $t_2 = (8, 1, 0)$
3. The computation requires one secondary and two primary attributes, i.e. $t_3 = (8, 1, 1)$

First of all, the proposed technique checks the global centroid and radius with the computation’s requirements. If it succeeds (cases 1 and 2), the algorithm continues with searching the clusters. Again, firstly it tries to find the most convenient cluster and secondly, if there is such a cluster, it tries to search within the individual elements of the specific cluster in order to find out the data point (node) which is as close as possible to the computation’s requirements. That means that for $t_1$, the algorithm will skip without searching the individual data points of the first three clusters, and it will collect the last data point which is the “closest” point to the task’s attributes. In the second case, $t_2$, again the algorithm will skip the first cluster since its $y$ coordinate (which is primary) is zero hence there is no possibility of any matching within the specific cluster, and will search only
the second and third clusters because their corresponding \( y \) attributes are non zero. Finally, in the third case, \( t_3 \), once the algorithm realizes that the global centroid of the resource representative has zero its third coordinate (which is a primary one) will stop searching immediately because there is no case to find any node which will satisfy the computation, \( t_3 \).

The quick results of the search of task \( t_3 \), leads to the observation that even if only one primary coordinate is missing or its value is zero from the global centroid of the resource representative, the searching procedure has to stop immediately. Consequently, if a cluster has a centroid with zero value to one primary coordinate that the task demands, again there is no need to search its individual data points but to proceed to the next cluster. These two facts increase substantially the efficiency of the searching process.

4.3. Performance Model

Let \( N \) be the number of tasks which are sent to the resource representatives. Each task according to equation 2 is consisting of \( n \) attributes (requirements) where \( l \) of them are primary and \( m \) are secondary \((n = l + m)\). In addition, let \( y \) represents the total number of resource representatives while each one of them represents \( W^{r_i} \) workers. So, the total number of the participating workers is \( W = W^{r_1} + W^{r_2} + \cdots + W^{r_y} \). Finally, and in a worst case analysis, let assume that the number of \( MAXHOPS \leftarrow y \), so the search space is extended to the global solution space. Therefore, the worst case analysis leads to the search of all the resource representatives of the GRID system and in the last one of them, the search of all its clusters and if we assume that at the end cluster at its end worker the task meets its criteria, then this is the worst scenario. Thus, in terms of complexity it can
be expressed as follows:

\[ f^r(N) = N(k_1a_l + k_2a_l + \cdots + k_ya_l) + Nk_ya_m = Na_l \sum_{i=1}^{y} k_i + Nk_ya_m \] (8)

In addition, if C represents the communication overhead to transfer the task’s requirements from a resource representative to its neighbor, since \( y \) is the total number of the resource representatives, the communication overhead produced by the \( N \) tasks can be expressed as \( N(y - 1)C \). Therefore, the total cost for both Resource Discovery and Neighbor Resource Discovery phases can be expressed as follows:

\[ f^r(N) = NKa_l + Nk_ya_m + N(y - 1)C \] (9)

where \( K = k_1 + k_2 + \cdots + k_y \). It is obvious that the time complexity increases linearly.

On the other hand, in a centralized environment, as in virtual organization grids [14] and local high throughput computing systems [13] where an Index Table keeps all the information regarding the workers (both primary and secondary attributes), the corresponding worst case scenario is to search the whole index table. The cost in this case can be expressed as follows:

\[ f^{IT}(N) = N(a_l + a_m)W = Na_lW + Na_mW \] (10)

Comparing the equations 9 and 10 it is more than obvious that:

\[ f^r(N) << f^{IT}(N) \] (11)

since \( K << W \) and \( k_y << W \) and the distance between \( f^r(N) \) and \( f^{IT}(N) \) is increasing when a) \( N \) is increasing, b) \( W \) is increasing and this
means that in a real computational GRID where the number of tasks which submit their requirements is extremely large and the number of the participating workers could be even millions, the proposed technique seems highly promising as it can be seen in the next section.

5. Experimental Results Analysis

In this section, we will evaluate the performance of our proposed techniques and validate the theoretical model previously presented, namely the Resource Discovery and the Neighbor Resource Discovery algorithms. At the system level, the computational GRID is represented as an undirected network graph where each node represents a Resource Representative (RR). Each worker is assigned to exactly one RR. At the application level, each task contacts its RR node and submits the attributes that the specific computation demands.

5.1. Simulation Parameters and Settings

The system of the Resource Representatives used in this simulations is consisting of 10 RRs connected through an irregular topology. On each RR there is a set of workers assigned on it as it can be seen in Fig. 6. The total number of the participating workers is 100 and they were distributed randomly to the RRs.

At the application level $10,000 \rightarrow 100,000$ tasks used to contact their resource representatives and to submit their requirements. Two different types of distribution used in order to simulate which tasks contact which RRs and the tasks’ arrival time, namely the uniform and Poisson respectively. In addition, uniform distribution used for the tasks’ requirements. The reason
is that if the system works properly under random conditions (uniform distribution), then it must work properly under any condition. The Poisson distribution is very close to real behavior so it had to be tested to simulate the tasks’ arrival time.

5.2. Simulation Results

Firstly, we would like to measure the time needed when a rather large number of tasks (100,000) contact the Resource Representatives (RRs) to find the most convenient workers for their requirements. The number of available workers were constant (100) and the attributes used in this simulation varied from 1 to 20. These attributes were considered a) all of them primary, b) half of them primary and half secondary. In contrary, we tested a system where a Centralized Index Table (CIT) exists and all the tasks contact it. As it can be seen in Fig. 7 when the number of tasks’ attributes is small the CIT approach behaves better than the RRs approach but increasing the
number of attributes, the RRs technique operates much faster than the CIT and when there are secondary attributes the RR technique responds even better. Note, that the time needed by both RR approaches, includes the time for the direct negotiations between the clients and the set of workers that the Resource Representative returns to the client plus the communication overhead (equation 9). Another very important issue here is that the failure of a resource representative does not affect the system since the client can contact another one while the failure of the CIT may be fatal or even if a CIT mirror exists, the transfer of the contacts will take additional time which will affect the performance of the system.

Figure 7: Different Number of Attributes

Another point of interest was how the number of workers affect the system’s behavior. In Fig. 8 two different kind of tasks in terms of the required number of attributes tested when applying to an increasing number of workers (100 → 1000) while the number of the required attributes were 4 (Fig. 8
- (a)) and 6 (Fig. 8 - (b)) respectively, and all of them were considered as primary. If some of the attributes were secondary then the speedup of the system would be better since they are tested only if there is a match to the required primary attributes. As it can be seen the RR approach presented much more better response time than the CIT’s one and its efficiency is increasing when the number of attributes is increasing.

In Fig. 9 the results obtained are presented when an increasing number of tasks ($10,000 \rightarrow 100,000$) are applying to a constant network of 100 workers and 10 RRs. Again, we tested two different numbers of attributes: 4 and 6 and once again all of them were considered as primary.

In addition to the previous results, we increased the number of participating worker to 1,000 while we kept the same number of RRs and we applied again an increasing number of tasks in order to observe if increasing the number of participating workers operates in a beneficial way for the system or not. Again two different numbers of attributes were used (all of them considered as primary). The results obtained show that the RR approach operates more than two times faster than the CIT approach when 100,000
Figure 9: Different Number of tasks, workers=100, (a) Attributes=4, (b) Attributes=6 tasks were applied and this is linearly increasing.

Figure 10: Different Number of tasks, workers=1000, (a) Attributes=3, (b) Attributes=6

Finally, in order to evaluate our techniques when there are both primary and secondary attributes, we tested a system where the number of participating workers were constant (1,000), the number of tasks were ranging from 10,000 → 100,000 and the requirements of the tasks were 10 from which 5 attributes were primary. Notice that the number of attributes if we include both dynamic and static attributes can be large as in condor classified advertisements [23, 13]. From the results obtained (Fig. 11) it is easy to see that when the number of primary attributes is small the performance of the system is increasing. The RR technique offers a 5 times speedup over the
CIT approach when the number of tasks is 100,000.

The overall conclusion derived from the simulations, is that the performance of the system increases when a) the number of tasks increase, b) the number of participating workers increase and finally, c) when the number of primary attributes decrease. These observations are very important since a computational GRID is consisting of thousands or even millions of workers distributed along the Globe and the Resource Representative can include workers that exist and operate in concrete Geographical spaces. The heterogeneity of the participating workers in terms of both software and hardware availability is expressed by the number of the attributes. Our approach seem to operate much more better than the CIT approach when the numbers of attributes is increasing and this is very promising since this reflects the real GRID where the available resources is extremely large.

Figure 11: Different Number of tasks, Workers=100, Attributes=10 (primary=5, secondary=5)
The CIT could be indexed of course using traditional database indexes such as the B-Tree index. However since CIT keeps both static and dynamic attributes the overhead for updating the B-Tree index would be tremendous in the case of a global table due to the frequency of change of the dynamic attributes. Dynamic attributes (such as the load of a worker) change very frequently and with the indexed CIT approach, each such change would require an update of the index file. This observation reinforces again the two essential proposals of our approach: (a) dynamic attributes should be excluded from the resource selection phase and (b) clustering approaches with networked RRs are essential for efficient resource selection as the number of workers increases.

6. Pre-execution configuration with elastic components

So far we concentrated on the variability and the size of the resources in the grid and consequent need to organize the searching process among these resources. There is however another important aspect in relation both to searching for resources and the eventual execution of a computation in the grid environment. The problem is that some computations may require a very specific set of resources which will be almost impossible to satisfy with the available resources. The grid computations need to be flexible enough to enable the execution in a variety of resources. This will increase the search space for suitable resources and the probability of successful matching. To allow this flexibility we are developing a process for engineering grid application component variants that are suitable for different execution environments.

There are many definitions of what a component is. The most widely
used is the one provided by Clemens Szyperski in [11]:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

Components are implemented using a component model. There are several popular component models for enterprise distributed systems, such as Microsoft COM (Component Object Model) and .NET, Sun Java Beans and Enterprise Java Beans and OMG CORBA. Component models prescribe the exact requirements of what a component should look like in order to be deployed in a respective component framework. Using component models and frameworks, difficult architectural mismatches [24], such as which component controls the sequencing of operations or incompatible patterns of interactions, are avoided and this is a very important first step for achieving reuse. Recently the interest in component-based approaches for both the development of grid applications and environments has increased [18]. A most notable example of component model specifically designed for the grid is the Grid Component Model (GCM) [17] in which an existing component model, Fractal [25], has been extended for the development of grid applications. The extensions of GCM to Fractal include mechanisms specific to the grid, as for example asynchronous communication in addition to RPC-like client-server communications, and the provision of common communication patterns such as multicast (one-to-many) and gathercast (many-to-one).

In grids applications might be executed in a number of different environments and under a number of different runtime conditions. The configura-
tion and runtime adaptation of applications to these different environments and conditions respectively is therefore essential for grids. A component model and the respective framework can provide the appropriate low-level mechanisms that are required for the configuration and adaptation of grid applications and therefore enable the efficient development of applications that will run successfully in a number of environments and under a number of different conditions. For example the generic communication patterns provided by GCM are useful to almost every grid application and can be used to configure an application prior to the execution phase of the GSP. In general however most configurations will be application specific since it is not possible to embody every conceivable optimization to the component model and framework itself. The role of the component framework is to provide useful generic mechanisms that can be used to build more application specific ones. To this end what will generally be required is a design process for grid applications that will enable the development of different component variants suitable for different environments and conditions. We are currently developing such a process for the design of so-called elastic components [6]. An elastic component is a family of components in which a base component called pure is refined by a number of variant components. Variants are produced through quality-driven transformations of the pure component. Variants provide the same essential functionality with the pure component but differ on a number of quality properties which they improve. The basic idea is that we apply to the component a number of different architectural design patterns depending on the actual quality improvement that we are interested in achieving. Architectural design patterns [26, 27, 28] are
multi-party collaborations in which each component assumes one role. The remaining roles become architectural obligations (i.e. other components in the architecture should assume these roles). Each pattern improves one or more quality properties and might impair others. Applying a design pattern assumes therefore certain tradeoffs. These tradeoffs in our case are related to the unforeseen environment in which the application will actually execute. In the general case there might be some interesting environment or contextual scenarios (e.g. slow network connection vs. fast network connection). The task of the grid application designer is then to provide suitable component variants for these application scenarios by modifying a pure component with the application of the appropriate patterns. The configuration of the application prior to the execution phase of the GSP will then include the client agent applying staging rules according to the environment which is now known, downloading the appropriate component variants as described in its policy.

To give a simple motivating example assume that a task needs to access a resource. This resource might be scarce to a site or abundant. The problem is that the grid application developer has no way of knowing which will be the case prior to resource selection, but this information might be available during the contract-net protocol that we described earlier in Sec. 3 as part of the extended information acquired from the client agent. Although the application can function both in the case of scarce as well as in the case of abundant resource availability, if the resource is abundant it would be preferable to acquire the resource during application startup to increase predictability and improve performance during the execution of the task. On
the other hand if the resource is scarce acquiring the resource at the latest possible moment (lazily) might be preferable to increase stability and provide optimal task startup, thus allowing the task to proceed until the resource is actually needed. These are the liabilities and benefits of the Lazy Acquisition pattern described in [28]. In this pattern the acquisition of the resource is carried out using a Resource Proxy. The Resource Proxy when first used actually acquires the resource lazily but prior to that it merely acts as being the resource. To achieve this the resource proxy has an identical interface to the resource and therefore the rest of the application remains unchanged and the difference is never noticed. In this application scenario then, the grid application developer would simply provide two different variants of the same pure component: one that embodies the lazy acquisition pattern by using a resource proxy instead of the actual resource, and another using the actual resource. Since this pattern does not require any additional roles to be present the rest of the application remains unchanged (i.e. there are no remaining architectural obligations). The client agent during the contract-net protocol would decide by following a suitable predetermined policy when to use the lazy version or the non-lazy version based on a threshold value for the resource availability.

6.1. The elastic components design process

The process for developing elastic components has the following steps:

1. The developer first develops a pure component. A pure component is a component that provides a cohesive functionality modulo any quality transforming techniques. The developer will of course follow the guidelines of a specific component model to which the component under
development belongs, such as GCM [17], and can use any optimizations provided but the component framework that seem important for the component.

2. The second step of the process involves the identification of important usage contexts for the component and the implications of these contexts to the Quality of Service for the application that will use (or reuse) the component. For example in the short motivating example presented earlier the different usage contexts were a resource constrained usage context and a usage context in which the resource was abundant.

3. After the identification of the possible interesting usage contexts and the respective quality concerns, the developer will identify from the design patterns literature, the patterns which seem more promising at achieving the quality goals. In our example the identified pattern was the Lazy Acquisition pattern.

4. Finally the developer will apply the identified design patterns to the component to produce a quality transforming variant component of the pure component. Both the pure and the refined (to the quality dimension of interest) components, will be stored in a repository at which all elastic components are stored, for future reuse.

In order to provide a constructive approach for the development of elastic components we are developing a toolset that will assist the developer in applying the elastic component design process. In this process the component developer will first develop a pure functional component and then will apply a number of design patterns to satisfy different quality concerns. The toolset is based in the metamodel depicted in Fig. 12.
In our metamodel each pure component provides a number of provided interfaces and might require a number of required interfaces. Interfaces are sets of operations. In addition each pure component provides some quality contracts that include the dimensions of interest. Dimensions are quality properties, such as availability, performance etc. Dimensions of the same type are comparable to each other. Also each dimension that is mentioned in a quality contract can be verified by using a verification model. The verification model describes the method that the component developer used to determine the value of the dimension. For example the model might describe that the dimension value was determined by simulating the component and its environmental assumptions and provide details for the simulation. The verification models are also necessary at a later stage to assist the component reuser (which might be a different person than the component provider) to assess the suitability of a component for a new system. A variant component is obtained by applying a number of patterns in the pure component. Patterns are also associated with the dimensions that they improve (ben-
efits) and the dimensions that they impair (liabilities). This results in a variant component that has improved quality contracts in some dimensions, but might have impaired quality contracts in other dimensions. In any case the component developer after applying the architectural pattern can use the same verification model that was used for the pure component’s contracts, to produce a new set of contracts for the variant component in which at least one dimension will be improved and zero or more dimensions might be impaired. Notice that verification models and formal contracts are optional, since it might be hard work to verify the quality properties of a component. This hard work though is justified in the cases that the component is candidate for reuse in many future applications and/or the component plays an important role in a safety critical system.

7. Conclusions and future research directions

In this work we have described a GSP process based on software agents, clustering algorithms and software components. We believe that collectively these technologies provide suitable solutions to the problems of resource discovery and selection since they address the issues of (a) efficient search to large search spaces (clustering), (b) negotiation skills, autonomy and temporal continuity (software agents), and (c) pre-execution configuration of applications to address the variations to the possible execution environments (software components). All these are important issues in relation to grid environments. Specifically for software components we believe, as others [18, 17], that in addition to allowing configuration and adaptation of applications with component variants, are also generally a very promising approach to
the development of grid applications, since component-based development will eventually hide the complexities of hand-crafted programming of grid applications, by exploiting the much simpler component framework services.

We are currently evaluating a number of different clustering approaches to determine the conditions under which each one is more appropriate. As for elastic components we plan on developing a toolset that will provide support to the application developer in choosing the appropriate patterns for each quality concern and generally implement more efficiently the process that we described in Sec. 6.1. We do not think that this process can be fully automated, however our research goal is to determine the exact conditions under which full automation is possible and provide adequate guidance to the developer for the general case.

References


