An Ontological-Fuzzy Approach to Advance Reservation in Multi-Cluster Grids

D.J. Ferreira¹, M.A.R. Dantas¹, Michael A. Bauer²

¹ Federal University of Santa Catarina (UFSC), Department of Informatics and Statistic (INE), 88040-900, Florianopolis - SC, Brazil
² University of Western Ontario, Department of Computer Science, London, Ontario, N6A 5B7, Canada
E-mail: ¹ {ded, mario}@inf.ufsc.br, ² {bauer}@csd.uwo.ca

Abstract. Advance reservation is an important mechanism for a successful utilization of available resources in distributed multi-cluster environments. This mechanism allows, for example, a user to provide parameters aiming to satisfy requirements related to applications’ execution time and temporal dependence. This predictability can lead the system to reach higher levels of QoS. However, the support for advance reservation has been restricted due to the complexity of large scale configurations and also dynamic changes verified in these systems. In this research work it is proposed an advance reservation method, based on a ontology-fuzzy approach. It allows a user to reserve a wide variety of resources and enable large jobs to be reserved among different nodes. In addition, it dynamically verifies the possibility of reservation with the local RMS, avoiding future allocation conflicts. Experimental results of the proposal, through simulation, indicate that the proposed mechanism reached a successful level of flexibility for large jobs and more appropriated distribution of resources in a distributed multi-cluster configuration.

1. Introduction
Multiple compute clusters, if well orchestrated as in a grid environment, can represent significant computational power for the execution of several classes of applications, including parallel jobs. One significant challenge in such grid environments is how to efficiently allocate resources on both local and remote clusters to applications. Within a cluster, a local resource management system (RMS) will naturally provide high priority to local applications. Some RMSs also support advanced reservation (AR), such as PBS Professional [1] and LSF [2].

RMS packages that have a built-in AR capability are broadly called meta-schedulers. Meta-schedulers are characterized by receiving users’ requests and then scheduling an application to one (or perhaps more) of the clusters to run. Thus, meta-schedulers are intrusive since they can change both local and remote queues. This AR functionality requires efficient design of co-reservation and co-allocation functions. A number of meta-scheduler projects for grid environments have already been proposed with different characteristics, such as Gridway [3] and GridARS [4].

Advance reservation is an important mechanism for ensuring effective utilization of available resources in distributed multi-cluster environments in order to meet a variety of user requirements. AR allows, for example, a user to specify requirements related to an applications
execution time, temporal dependencies among processes, types of hardware and software, etc. These specifications can, in turn, enable the system to attain higher levels of service. However, the support for advance reservation has been limited in multi-cluster environments due to the differences in how resources and capabilities are defined among different systems by different administrators within different organizations. This makes matching the user specifications, which can also be done in a variety of ways, to the actual resources difficult.

In this work we introduce an advance reservation method that can allow a user to reserve a wide variety of resources among different clusters (matching). It also dynamically verifies the possibility of any reservation with the local RMS, thus avoiding future allocation conflicts. The approach makes use of a fuzzy-ontological approach for matching user requirements to descriptions of system configurations. An architecture for doing this and interacting with local RMSs is introduced. Experimental results of the proposed approach through simulation suggest that the proposed mechanism can achieve a reasonable level of flexibility for handling large jobs and a more appropriate distribution of resources in a distributed multi-cluster configuration.

2. Related Research Work
Co-allocation can be referred as the distribution of processes of a job among several clusters. This mechanism can increase the utilization level of computer resources, reduce turn-around time and computations using numbers of processes larger than processes on any single cluster. The research presented in [5] introduced a scalable co-allocation strategy called the Maximum Bandwidth Adjacent cluster Set (MBAS) strategy. This strategy focuses on improving the usage of networks in high performance computing clusters.

Advance Reservation means to dedicate a set of resources to a user at a specific future time for a specific duration. The work of [6] presents an extension of the path selection process for advance reservations, introducing the notion of time to dynamic routing. It shows and classifies different temporal link metrics.

Co-Reservation, in the scope of multi-clusters, means the coordination of resources in sequence, as a grid configuration. In [7], a mechanism for advanced reservation and co-allocation of grid resources is presented. It takes into account users’ interaction to negotiate with a resource provider for a suitable option between application requirements and resource capabilities. The paper focuses on enhanced utilization of resources, maintaining the Quality of Service (QoS) requested and a specific time for performance.

3. Approach
A shared coordinated multi-cluster environment that could be able to solve problems in dynamic fashion, crossing different enterprises, can be referred as a computing grid configuration. An application is composed of a group of dependent, or independent, subtasks. These subtasks can be scheduled and distributed through a parallel multi-cluster grid system therefore meeting various objective functions. This resources sharing can improve resource utilization across virtual organization (VOs) and enhance the response time for the application [8].

In this work, we assume that when an application requires the execution of a job, this job is composed of processes, each one with different subtasks. The focus of this paper is to introduce a dynamic advance reservation method for multi-cluster configurations. The method developed supports jobs comprised of a number of processes that could not be executed in a single cluster. A global selection and co-reservation tool interacts directly with a clusters local reservation system. The design and implementation of our approach takes into account the research in [9] and [10].

On the client side, we assume that there is an Interactive Interface that receives a request from a user detailing the type and quantity of resources required to execute a job. A semantic query is created and then sent to a server. The server has static semantic information about the resources
of each cluster of the environment. The Selector matches the query with the information about the clusters resources, except the information about the number of processors. Then it finds all possible scenarios, based on that semantic stored data, and sends it to the AR Manager component.

In the server, the AR Manager accesses information about the reservations already done. Based on the information from the Selector and the availability of each selected cluster, it determines how to make the reservation of resources for the job. However, depending on the quantity of processors required, sometimes no single cluster is sufficient for the request. In this case, distributed reservations among clusters (i.e. multi-clusters) are done for the application tasks. This distribution is controlled by a fuzzy method based on research presented in [10], which takes into account bandwidth workload between clusters.

In order to avoid goal conflicts, the AR Manager dynamically requests the reservations with the RMS of each local cluster. Only when the reservation is locally accepted, is its information stored by the AR Manager. Our experiments were done with the Gridsim simulator [11], and assumed that all local RMS involved support advanced reservations.

### 3.1. Server Architecture

The server side represents the main contribution of this paper. It is the server that decides on the reservation scheduling for users requests. Figure 1 shows the architecture of the server.

![Figure 1. Server Architecture](image)

The Selector module uses a resource matching approach based on multiple ontologies, an extension of the research in [9]. A reference and query ontology was conceived in order to integrate different ontologies from clusters in different VOs. It treats the diversity and integration of multi-clusters environments statically, keeping semantic information of some cluster parameters, such as operating system version, processor architecture, processor clock speed and amount of memory.

The FuzzyControl module is an extension of work in [10], that uses fuzzy logic algorithms to control and to determine the matching of a user’s request to that of available resources and communication links in a multi-clusters configuration.

The AR Control is responsible for the co-reservation of users’ requests. It takes into account parameters of timing and uses the fuzzy control method to calculate availability of the clusters in
the environment. Therefore, it decides which clusters are capable, and preferable, in providing the requested resources.

Finally, the advanced reservation layer has information about global reservations already done and the possible links among clusters. It also has a module for contacting the environment, in order to test local reservations.

4. AR_Manager Implementation

4.1. Main Concepts

In this section, we provide details about the implementation of the AR_Manager module. In order to understand its design, it is important to clarify some aspects and assumptions considered in the development of the system components.

The Selector component allows a user to decide what kinds of resources are required in a cluster to execute a specific job. It offers various features (e.g. operational system, minimum disk space, etc.) that are used as filters to the semantic selection process. It is important to observe that the number of processors is not considered in this phase. When the selection is done, just the clusters that fit the user’s requirements remain in the set of clusters available to answer the user’s requests. Then, these clusters pass through the final filter, the AR_Manager module, which decides the final scenario.

In the advance reservation process, a few more parameters are analyzed in order to get the appropriate configuration. These parameters include the number of processors, the availability of the cluster and, if it is the case that multiple clusters must be used, the workload bandwidth required between clusters.

We assume that the query for one job is a set of resource features required for the execution of that job. Among these features there are the inter-process workload bandwidth required and the number of processors. The latter parameter it is referred as the size of the job. In addition, a query for a job also has three types of time parameters: a start time, which is the time a job is ready to run; the deadline for the execution; and the duration of time the resource should be reserved for the job.

When the Selector returns results satisfying the query, it generates a ReservationJobQuery, composed of the job query itself, and the set of clusters remaining after the semantic selection phase. In the second phase, a ReservationJobResult is created. It has the identifier of the final clusters selected to be reserved, the job size, and an interval. This interval consists of the start time for the reservation and the how long it will spend.

A time-slot based resource management approach is used to keep the available and reserved intervals of each cluster. The idle periods are in duration order so as to return from a search the interval with minimum duration that can answer the job. This behavior reduces fragmentations and leaves major intervals free for bigger jobs.

Another structure maps connections between clusters. A connection is composed of the identifiers from of two connected clusters, the link maximum bandwidth and the workload. These workloads are updated when a reservation is completed. If a reservation requires inter-cluster communication, it also carries a link connection information.

4.2. Advance Reservation Activities

This subsection describes the main activities of the AR_Manager, which are illustrated in Figure 2. On the left, the semantic Selector gets an answer for a query from the ontology base specifications previously stored. It returns a set of clusters that can probably meet the query individually, but does not take into account information about the quantity of processors required for the job. This set of clusters is arranged in ascending order of cluster size. The job and the set of selected clusters comprise the ReservationJobQuery, which is sent to the AR_Controller.
The **AR_Controller** provides a feature that allows each selected single cluster to verify if it is able to answer the request alone. First it determines the number of processors available within the cluster. If this number is greater or equal to the required number of processors for the job, the cluster is allocated to the **ReservationJobResult**, and the **Reservations_DB_Manager** component statically analyses the availability of this specific cluster. The **Reservations_DB_Manager** has access to information about global reservations already done and idle intervals of each cluster. An interval fits the job when $max(s_I, s_J) - min(d_I, d_J) \geq dur_J$, where $s_I$ is the cluster interval start time, $s_J$ is the job start time, $d_I$ is the cluster interval deadline, $d_J$ is the job deadline and $dur_J$ is the job duration. The algorithm passes through the cluster intervals in ascending order of duration. Therefore, the interval with minimum duration that fits the job is the one which is selected as result. This behavior leaves major intervals for incoming greater jobs, giving to theses jobs more chance of being allocated in a single cluster.

The interval found is then allocated to the **ReservationJobResult**, and it is sent to the **Connector** component. The **Connector** sends this result to the specific remote local cluster, asking it to try the reservation. If the cluster can reserve the job at any interval, it will do the local reservation and return the interval reserved. The interval result of the **Reservations_DB_Manager** is not always equal to the interval returned by the **Connector**. However, it is important to make the static search before trying it locally, in order to reach higher probability of finding an interval. This avoids unnecessary communication with the local cluster and spending time for nothing.

Finally, the result of the reservation is stored by the **Reservations_DB_Manager**, and the intervals are updated.

Once the clusters are searched in ascending order of amount of processors, the answer is the one with minimum size that has an interval fitting the job. The algorithm lets majors clusters...
free for incoming jobs of larger sizes. An important feature is that it tries to avoid putting dependent tasks of the job to be executed among different clusters, since that would increase communication time. If at the end of this phase there was no single answer found at all, then the AR_Controller tries to distribute the job among the clusters, as is shown next.

4.3. Co-Reservation
In the co-reservation process, job process reservations are distributed among clusters received by the Selector. The control of this distribution is done by a fuzzy logic method based on the work presented in [10]. It generates two thresholds:

δ - Calculated from the job size and the type of communication among clusters within the VO. It controls the job splitting, trying to avoid excess inter-cluster communication, which could increase the time to execute the job;

λ - Calculated from the same parameters as δ, plus the job bandwidth workload required for inter-process communication, and the maximum bandwidth between each cluster selected to answer the query. It controls the saturation of the links between clusters. If the link is saturated, it cannot be used by the job.

Each cluster of the ReservationJobQuery that passes the test of the FuzzyControl module is summed to answer the job until reaches the job size, as illustrated in Figure 3. In order to control the job splitting, at least one of the clusters to be reserved must follow the rule \( c1Size \geq \delta \times jobSize \), where \( c1Size \) is the cluster. Therefore, it is the first requirement calculated. If the cluster meets this first constraint, the Reservations_DB_Manager finds an idle interval \( c1i \) for the job in the cluster \( c1 \).

In a second phase, all clusters that have direct connections to cluster \( c1 \) are examined. This is done by the Connections_DB_Manager, that has previous information about all connections.
Table 1. The ontology set utilized for the experiments

| Request's Concepts | VO.B's Concepts | VO.C's Concepts |
|--------------------|-----------------|-----------------|
| min.number_of cpus | min.number_of cpus | number_of_cpu | |
| min.virtual_memory_available | min.virtual_memory_available | swap_free_MB | free_virtual_memory |
| min.clock_speed | min.clock_speed | max_clock_speed | |
| owner | owner | login | |
| number_of_cpus | number_of_cpus | number_of_cpu | |
| min.physical_memory_available | min.physical_memory_available | mem_free_MB | free_physical_memory |
| min.free_disk_space | min.free_disk_space | free_space_MB | available_space |
| os.type | os.type | [operating system] | |
| max_load_cpu_15min | max_load_cpu_15min | percentage_load_15min | |
| max_load_cpu_5min | max_load_cpu_5min | percentage_load_5min | |
| max_load_cpu_1min | max_load_cpu_1min | percentage_load_1min | |

among clusters in the environment. These clusters do not have to pass on the share constraint, but only the link saturation constraint. This last constraint defines that the link between \( c_1 \) and \( c_2 \) must follow the rule \( \text{workload}/\text{maxBW} \geq \lambda \), where \( \text{workload} \) is the inter-process workload required and \( \text{maxBW} \) is the maximum bandwidth between \( c_1 \) and \( c_2 \). As a result, the link is saturated and \( c_2 \) cannot be in the final list of clusters.

If there is an interval in \( c_2 \) that fits \( c_1i \), the cluster \( c_2 \) is added to a list in UPBW (Unit Process Bandwidth) order [10]. The UPBW is a value that measures the \( \text{linkW}/c_1\text{Size} \), where \( \text{linkW} \) is the current link workload between \( c_1 \) and \( c_2 \) given by the \textit{Connections_DB_Manager}. When the list size reaches the job size, it creates a reservation for each cluster in the list and then tries to do all the cluster reservations locally, through the \textit{Connector}. If one of the reservations fails, the \textit{AR_Controller} tries a new cluster to be the \( c_1 \).

5. Experimental Environment and Results

In order to test the approach, a simulation environment was built with the GridSim simulator [11]. This section describes the environment created under GridSim and describes a case study that illustrates the general behavior of the methods.

5.1. Simulation Aspects

This work is based on a configuration with a central server, remote clients and remote clusters inside a multi-cluster grid environment, as seen in Figure 1. The central server stores cluster information and has the global co-reservation module, which was described above. This module decides in which cluster a job or subtask must run. We assume that all clusters in the multi-cluster system support advanced reservation. In other words, the local schedulers on each cluster are capable of making advanced reservations for local and global jobs.

The GridSim toolkit was used to simulate this multi-cluster configuration. It provides a discrete event framework for simulation of different classes of core Grid entities such as heterogeneous resources, users, jobs, applications, resource brokers, and schedulers. It provides schedulers for single or multiple administrative domains systems configurations such as clusters and Grids.

One important feature of GridSim is that it allows advance reservations. It uses FCFS (first-come-first-served) approach for global scheduling. However, we used the co-reservation module described, which offers the possibility to analyze not only the availability of the local cluster, but also take into account features of the job and resources. Then, the local advanced reservation provided by GridSim is used to check if there is any conflict with local reservations.

In the following experiments, each process is considered as a GridSim gridlet, and a job is composed of various processes.
5.2. Environment
The environment created in the GridSim for experiments is composed of two VOs, and their characteristics are described in Table 2. VO-C has three clusters which are all inter-connected. VO-B has two inter-connected clusters. However, the clusters of VO-C do not have direct connections to the clusters of VO-B.

After creating the multi-cluster configuration, we identified a number of basic concepts to be used in both virtual organizations for requesting resources. The matching of these semantic concepts is done by the tool described in this paper. Table 1 presents how these request concepts are understood in virtual organizations C and B. VO-C represents a Canadian environment, whereas VO-B is a Brazilian configuration.

| Table 2. Multi-cluster configuration |
|-------------------------------------|
| VO-C processors | VO-B processors |
|-----------------|-----------------|
| Cluster_01      | 7               | Cluster_1 | 4           |
| Cluster_02      | 4               | Cluster_2 | 4           |
| Cluster_03      | 3               |           |             |

5.3. Case Study
Several experiments are presented and analyzed in order to clarify the sequence and main aspects of a selection and co-reservation in this work. It considers two users, user_0 and user_1, making a request for a reservation. Both users require almost the same clusters characteristics, presented in Figure 4 and 6. The requests differ with respect to the number of processors needed and time characteristics for each user job, as seen in Table 3.

| Table 3. Required parameters for reservations |
|-----------------------------------------------|
| Parameters | user_0 | user_1 |
|-----------------|--------|--------|
| CPUs            | 2      | 6      |
| start time (min)| 10     | 20     |
| duration (min)  | 60     | 60     |
| deadline (min)  | 80     | 90     |

User_0 requires 2 CPUs. Cluster_03 is the one with minimum size in the environment that can answer this request. Since there were no goal conflicts during the reservation process, it can be answered by a single cluster, Cluster_03, which is locally and globally reserved. The result received by user_0 is shown in Figure 4. On the left part of the figure, there are the parameters required by the user. On the right side, there are a set of clusters that could possible answer the request. These clusters are results from the first phase of the search, when only the resource features are considered. On the bottom of the figure, there is the final result of the reservation, in which the availability of the clusters is analyzed.

On the other hand, user_1 requires 6 CPUs. In the environment, Cluster_01 is a single cluster that has enough CPUs for this request. However, it does not have the minimum disk space required, thus it cannot be an answer to the request. Therefore, it does not appear in the first phase of results for user_0 (Figure 4), or user_1 (Figure 6).

User_1’s job has to be distributed among clusters. The algorithm first tests the cluster with the largest size, namely, Cluster_02. But, the other cluster selected that it has connection to is
Cluster_03, which has two CPUs occupied by user_0. As a result, the sum of free CPUs available does not match the requirement. Afterwards, the clusters of the second VO are tested, reaching the answer shown in Figure 6.

Figure 4. Results for user_0

Figure 5 presents the usage of the cluster processors along the timeline.

Figure 5. Reservations timeline

5.4. Conclusion and Evaluation
In this paper, we presented an approach for advance reservation and co-reservation of cluster resources in a multi-cluster environment system. It adopted a semantic integration of multiple ontologies to provide a more appropriated mechanism to gather information on available grid resources. In addition, it integrated a logic fuzzy algorithm to control distribution of job processes among clusters in co-reservation phase.
Figure 6. Results for user_1

The proposed approach was characterized by the description of the grid resource reservation mechanism and a prototype. This prototype was designed and implemented to make dynamic reservations for large jobs. It also allows the process of matching available resources from different virtual organizations, under different resource descriptions, integrating different ontologies.

The proposed approach also considers the possibility of dividing large jobs that cannot be executed in a single cluster among multiple clusters. These jobs are divided into subtasks among clusters in a controlled way, in order to try to avoid extra fragmentation that increases the cost of communication. It also controls the saturation level of links between clusters at the time the reservations are done. Finally, it avoids conflicts with local reservations, by checking dynamically with local RMSs using a co-reservation approach.

The prototype was tested through simulations done with GridSim, and appropriately achieved the desired goals of selection and the reservation of clusters to attend large jobs in an effective way. The approach employed a human interaction interface as part of the solution, in order to make the interaction more accessible for ordinary users.

As a future research direction, we intend to extend the grid resource reservation system to guarantee different levels of QoS can be met for users. It is also important to experiment with job workflows using the prototype, and to consider ways to optimize performance issues. Additionally, it would be interesting to dynamically show reservations and jobs status during their activities.

References
[1] Pbs. [online], available: http://www.pbsgridworks.com/
[2] Platform computing inc.: Lsf product suite. [online], available: http://www.platform.com/products
[3] Gridway metascheduler. [online], available: http://www.gridway.org
[4] Takefusa A, Nakada H, Kudoh T, Tanaka Y and Sekiguchi S 2008 Job Scheduling Strategies for Parallel Processing, *(JSSPP’08)* LNCS 4942 152–168

[5] Qin J and Bauer M A 2009 Cluster Computing 12 323–340

[6] Barz C, Pilz M and Wichmann A 2008 8th IEEE International Symposium on Cluster Computing and the Grid *(CCGRID’08)*, IEEE Computer Society, 710–715

[7] Siddiqui M, Villazon A, Prodan R and Fahringer T 2005 9th International Multitopic Conference, IEEE Computer Society 1–6

[8] Foster I 2002 Grid Today 1 22–25

[9] Silva A P C and Dantas M A R 2007 19th International Symposium on Computer Architecture and High Performance Computing, *(SBAC-PAD’07)* 143–150

[10] Qin J and Bauer M A 2007 The 20th Annual Symposium on High Performance Computing Systems and Applications, *(HPCS’07)* 18–24

[11] Buyya R and Murshed M 2002 Concurrency and Computation: Practice and Experience *(CCPE’02)* 14(13-15) 1175–1220 URL www.gridbus.org/gridsim/