Middleware-Level Collaborative Resource Discovery for Large Clusters

Jameela Al-Jaroodi and Nader Mohamed
Dependable Systems Middleware (DeSyM) Research Lab.
Electrical and Computer Engineering
Stevens Institute of Technology, Hoboken, NJ 07030
{jaljaroo, nmohamed}@stevens.edu

ABSTRACT

In this paper, we introduce a collaborative resource discovery mechanism to enhance system resource allocation and utilization by the distributed applications. The mechanism involves the collaboration of the software agents to collect, organize and efficiently exchange resources information. The main goal is to provide an accurate single system image for all the resources. The concept of virtual clusters is used to enhance localization of information and minimize exchanged messages. This mechanism is introduced as a part of the run-time support layer in Delmon. Delmon is a three-layer middleware framework for distributed systems, which includes the run-time support layer. This layer utilizes agents that reside on the participating machines and communicate with one another to perform the required functions. The agents hide the details of scheduling, controlling, monitoring, and executing user jobs, while the management of system resources is made transparent to the user. The proposed mechanism for collaborative resource discovery is based on the hierarchical structure of the agents and the organization and update protocols.

1 INTRODUCTION

A new choice of architecture for high performance computing has emerged to be the cluster of PCs/workstations and networked heterogeneous systems. Such systems provide processing power comparable to special purpose high-end multi-processor systems for a fraction of the cost. However, these systems require system and software services (middleware) that can support the distributed architectures and provide transparent and efficient utilization of the multiple machines. Furthermore, heterogeneous environments pose great challenges to the distributed applications developers. As a result middleware is becoming a very important component in distributed systems design.

Delmon [1] is a three-layer distributed systems middleware framework, where the run-time support layer relies on agents to provide the required functions to the system. A hierarchical structure for the agents to optimize their functions [2] was also introduced by the authors. Agent startup, configuration, and update protocols were discussed. Delmon, using distributed software agents, is developed to provide services for high performance programming environments and applications on clusters and networked heterogeneous systems. A layered approach is utilized to enhance reusability and stability in addition to enhancing functionality and resource utilization. Delmon also allows for easy modifications and updates of the different functions and services at the different layers and provides flexible component-based plug-ins. The agents enhance expandability, allowing the number of machines involved to grow easily by providing services that include job distribution, monitoring, and controlling for the system. This provides flexibility and ease of managing the different resources available. In addition, the distributed agents provide the required run-time support for the parallel and distributed applications developed at the application level.

Resource discovery is an important aspect of efficient utilization of computing resources; therefore, many research efforts have been invested in the issue. In general, most resource discovery approaches address the issues of information resources on the Web, peer-to-peer networks and large distributed databases. Furthermore, very few of these approaches address the issues related to physical and computing resources that support distributed environments such as heterogeneous clusters. More importantly, very few discuss the issues of resource discovery as a middleware-level service.

In this paper, we approach resource discovery as an integral part of the run-time support layer of our middleware framework. At this layer, agents organized in a hierarchical structure can utilize their access to resources and collaborate to perform resource discovery tasks efficiently. The main objective of the paper is to describe the hierarchical agents’ organization and introduce a mechanism for collaborative resource discovery. We first start by identifying the types and attributes of the resources and the necessary mechanisms needed to find them and keep track of their status. Moreover, we devise a mechanism to make such information about resources up-to-date and available to the entire system in an efficient
and timely manner. The hierarchical infrastructure of the agents [2] will be the bases for agent collaboration.

In the rest of this paper, Section 2 provides an overview of some related work and Section 3 discusses the resource discovery concepts. Section 4 provides an overview of the Distributed systems middleware framework, Delmon, and discusses the run-time support layer in addition to the hierarchical agents’ organization. Then, Section 5 introduces the collaborative resource discovery mechanism. Finally, Section 6 concludes the paper.

2 RELATED WORK

Many researchers explored the issues in resource discovery for large scale systems. However, many address issues related to file searches and information discovery rather than system resources. In a few occasions some researchers addressed these issues such as in the Grid environment, peer-to-peer networks, and mobile networks. The following is a brief summary of some of these efforts.

Warfield et al. [19] approach the distributed systems issues and identify some architectural problems such as failure resolution, resource management and administration. In addition they also identify aditional problems such as physical resource discovery and naming, security and privacy, and system evolution. Furthermore, Foster [6] identify the various stages of resource management which include discovery, selection and allocation. They define resource management as “Operations used to control how the capabilities provided by the resources and services are made available to other entities” in addition to highlighting scalability as one of the challenges facing efficient resource management.

Resource discovery has been explored intensively in various domains such as peer-to-peer computing, mobile and wireless networks, information searches and retrieval in addition to systems resources discovery. Schwartz et al. [18] provide an interesting overview and comparison of some information resource discovery approaches. Foster [6] also describes several resource management tools in addition to discovery mechanisms for the Grid. The Globus architecture [7] includes some of these mechanisms to enhance resource discovery. In other research [17] the authors discuss a flexible information discovery mechanism for large distributed systems such as the Grid and peer-to-peer networks. Kuten and Peleg [11] also address peer-to-peer networks and provide an asynchronous resource discovery mechanism and introduce a deterministic algorithm for the environment of asynchronous networks. Another example of resource discovery is the one used in Trilogy [9], a collaborative project to develop a virtual laboratory. It utilizes a multi-agent system to facilitate resource discovery through the resource agents and mediator agents. The article [20] provides an analytical overview of some of the existing decentralized information resource discovery techniques and their shortcomings. Furthermore, it discusses an enhanced discovery technique using adaptive social discovery.

Another approach for resource discovery is in relation to mobile hosts and wireless networks. A statistical framework [3] is described for resource discovery in distributed computing environments with mobile hosts. Denny et al. provide another example [5], where the authors describe Mobiscope, a service for discovering moving resources and identifying their locations.

Finally, there is also some limited research in systems resource discovery as part of a larger set of management and allocation services to support distributed computing in various environments such as clusters, the Grid, sensor networks, and networked systems. A mechanism for rapid resource discovery using metadata trading [13] is discussed. SWORD [14][15] is a scalable resource discovery service for wide-area distributed systems using a query mechanism to probe resources. Kong and Berry [10] use the architecture model proposed by DSTC’s architecture unit to specify a general resource discovery system. NEVRLATE [4] is another approach to provide resource discovery mechanisms in peer-to-peer networks using a two-dimensional grid for resource registration and lookup. The classified advertisement (classad) matchmaking framework [16] provides a flexible and general approach to resource management in distributed environment with decentralized ownership of resources. Based on the examples introduced here and many others explored, we arrive at some interesting observations:

1. Most resource discovery approaches address the issues of information resources on the Web, peer-to-peer networks and large distributed databases.
2. Although some techniques address physical resources management, very few address the resource discovery issues of these resources to support distributed environments such as heterogeneous clusters.
3. Very few approach the issues of resource discovery as a middleware-level service.

In this paper, we approach resource discovery as an integral part of the run-time support layer of our middleware framework. At this layer, agents organized in a hierarchical structure can utilize their access to resources and collaborate to perform resource discovery tasks efficiently.

3 RESOURCE DISCOVERY

Resource discovery as defined earlier, deals with issues of finding resources that match the applications demands. In addition, most approaches deal with information as the resources to be discovered. In this domain the approaches become very complex and require various mechanisms to search for and allocate the requested information. From the distributed system’s view point, it is the physical
resources and the computing services that need to be discovered. As a result the search domains and allocation techniques are relatively smaller and require different approaches from those used for information discovery. Generally, resource discovery is part of a larger set of services called resource management, which include:

1. Resource discovery, where the required resources are found and registered
2. Resource Selection, where the most suitable resources for the application are selected from the set of discovered resources
3. Resource allocation, where the selected resources are bonded to the requesting application
4. Resource monitoring, where allocated resources are monitored
5. Resource deallocation, where resources are returned to the pool of un-used resources for later use

Discovery is an essential step in the process of allocating and managing the systems resources and providing applications with the optimal set of resources for their requests. Furthermore, the discovery process requires continuous access to all available resources in the system. As a result, the larger the system, the more complex the discovery process becomes. There are many challenges facing resource discovery services:

1. Distributed resources. In a distributed system many of the resources are physically located in different places. This poses communications limitations and delays on the process. In addition, the distributed environment usually does not have a centralized control entity, which makes it difficult to keep the available information in one place. Furthermore, a centralized approach is usually inefficient for a large distributed environment since frequent updates and monitoring causes increased loads on the system.
2. Heterogeneity. In addition to being distributed, resources are also heterogeneous in hardware and in software. Many resources have different methods of access and utilization and even similar resources could become heterogeneous during their utilization because of the differences in capacity, speed, and services using them.
3. Scalability. Resource discovery mechanisms increase in complexity with the increase of the size of the distributed environment. Many approaches work well with small scale systems, but fail quickly when the systems expand. The more resources available, the more work needed to keep track of them all and update the interested components of their status.
4. Timeliness. A resource discovery service must be able to provide its information in a timely manner. In many cases, any delay in relaying the information to the interested application may result in inaccurate information, thus inefficient resource utilization. The dynamic changes in the environment need to be reflected accurately to the requesting applications to ensure efficient utilization of these resources.

To have an accurate resource discovery service, we also need to accurately define resources. Any resource in a distributed environment may have two types of attributes:

1. Quantitative attributes such as speed, capacity, delay, bandwidth, memory capacity, latency, etc. These represent measurable components that the discovery service can easily measure and compare to others. These attributes are also dynamic depending on the levels of utilization. For example, if the total memory on a machine is 512MB, and a process is executing on this machine is using 200MB then its current capacity is only 312MB. However, when another process starts and uses 150MB, then the memory capacity will reduce to 162MB.
2. Non-functional attributes such as accessibility, ownership, security and reliability. These are important descriptors of a resource however; they are not easily measurable and cannot be compared to one another directly.

4 THE RUN-TIME SUPPORT LAYER IN DELMON

The middleware infrastructure is designed to satisfy the requirements of distributed applications and tools on heterogeneous systems. The framework provides a three layer middleware that provides services in a modular manner. At the middle layer, the run-time support, system services such as scheduling, resource management and others are provided. (See Figure 1). Software agent technology has been used in many systems to enhance the performance and quality of their services [8]. In Delmon, the main functions of the agents in the run-time support layer are to deploy, schedule, and support the execution of the distributed applications, in addition to managing, controlling, monitoring, and scheduling the available resources on a single cluster or on a collection of heterogeneous systems.

The agents have a framework for an automated startup and configuration mechanism in a hierarchical structure. The startup stage is essential to guarantee the accurate and efficient operation of the middleware infrastructure and the applications using it. The main goal is to provide system startup and configuration with minimum user involvement. The mechanism for adding and removing agents from the system and their effect on the configuration is provided.

4.1 Hierarchical Structure of Agents

Within a single cluster or a limited number of machines participating in the system, a linear structure is sufficient for the agents to communicate and achieve their functionality. However, this requires agents to be fully connected, which may not always be possible. In addition,
the linear structure causes considerable delays for some operations that need to be performed on all participating machines. To overcome these limitations, we designed a hierarchical structure where agents have multi-level connections in the system. Generally, a networked heterogeneous system composed of clusters and multiprocessor machines forms the top level of the hierarchy. Within each of these machines or clusters, one or more levels may be formed, depending on the type of machine and number of nodes/processors in it. In this structure agents are in one of two modes of operation:

1. **Leader agent (called leader hereafter):** an agent that manages and controls a set of other agents under its control. Leaders at the same level communicate with one another directly.

2. **Regular agent (called agent hereafter):** an agent that performs the regular agent operations. Agents under the control of the same leader should be able to communicate with each other and their leader directly. In addition, agents in different layers, but in the same physical cluster communicate with one another directly.

Shown in Figure 2 is an example of a hierarchical structure of a networked heterogeneous system, which we will refer to in the rest of this section. The squares denote leader agents, while the circles represent agents. A leader along with its agents forms a virtual cluster (e.g. leader 4 and the nodes under it). Some machines such as SMP (symmetric multiprocessing) or MPP (Massively Parallel Processing) machines need a single agent to handle the resources (e.g. leader 1), while others such as clusters need an agent for each node. The connecting lines represent bi-directional communication links between nodes. However, at the top level of the hierarchy (where leaders reside on headnodes), the links between leaders (e.g. 1, 2 and 3) may be a multi-hop path. The details of the hierarchical structure and startup protocols are described by Al-Jaroodi et al. [2].

### 4.2 Agent Operations

For the agents to operate efficiently, they need a startup protocol to automatically identify and communicate with one another. The initial stage requires manual installation of the first leader agents on the head nodes. The leaders then start the startup and automatic configuration phase.

1. Each leader is responsible for performing the following tasks:
   a. Execute the startup protocol to automatically acquire connectivity and operational information in the system.
   b. Periodically perform availability checks of the leaders and descendant agents. If a leader or agent does not respond, activate leader recovery or agent update protocols.
   c. Perform information routing for other agents to ensure full connectivity with other clusters and machines in the system. Many routing protocols can be adapted for this system, but the discussion of the routing details is beyond the scope of this paper. One suitable example is the content-based object routing technique called Java Object Router (JOR) [12].

2. Each agent should
   a. on activation (by receiving an AAM) find and register local node resources information. Resources include available CPUs, CPU power, storage and memory capacity, etc,
   b. respond to the leader with an AMA message containing the agent’s ID, address and resources information, and
   c. receive and locally update the neighbors’ addresses from the leader for future inter-process communication.
4.3 Protocols

4.3.1 Leader Startup Protocol
This protocol is designed to assist in automating the startup and configuration of leader agents. In this protocol the leaders in each headnode will seek to communicate with other active leaders and establish connectivity. Then each leader will start activating its descendant agents and collecting their information. The outcome of this protocol is to have leaders acquire resource information about their descendant agents (including virtual clusters) and routing information about other leaders. In addition, all agents within the same cluster (or virtual cluster) need to have address information of their leader and that of one another. Another important aspect of this protocol is that it allows leaders and agents to be easily added to the system with minimum user intervention. In general, the overhead incurred in constructing a hierarchical structure is relatively high, thus it may not benefit a system with a small number of nodes. However, it is essential in two environments:

1. The system is composed of multiple smaller systems (clusters, NOW, multiprocessor machines, etc.) that do not have full connectivity to all their nodes. Thus, the head node in each subsystem is assigned a leader that is responsible of connecting it to other subsystems.

2. The system includes very large clusters comprising tens/hundreds of nodes, thus accessing all nodes in a linear fashion is very time consuming. Here, the threshold needs to be selected to optimize the utilization of the suitable structure. Analytical models (as discussed in more details in section 5.6) or experimental evaluations can be used to select that value.

4.3.2 Leader Recovery Protocol
This protocol is used in case a leader fails to respond to an agent monitor message sent by another leader. If a leader \( L_x \) at one level times out before receiving an acknowledgement message from another leader, say \( L_y \), then \( L_x \) pings the node/machine where \( L_y \) resides to see if it is connected and up. If the node is still up, then \( L_x \) initiates a remote agent activation command using an agent activation message to reactivate the agent and when the new agent is up, \( L_x \) activates it as a leader and sends it all relevant leader information. The new leader, \( L_y \), uses the startup protocol to restore its information. If the agent does not reinitialize (e.g. has been deleted from the node) or the node does not respond (e.g. has been powered off), then, if a connection exists to another node, \( L_x \) activates it as a new leader. Otherwise, \( L_x \) reports the problem to the administrator and excludes all routing information to the cluster led by \( L_x \) from the routing tables and informs other leaders of the changes.

4.3.3 Agent Update Protocol
This protocol is used to report changes in the available resources within a cluster or virtual cluster. The protocol is triggered if one or more nodes (other than the headnode) in the cluster fail. When a leader \( L_x \) does not receive a response from a descendant agent, then \( L_x \) pings the node of that agent. If it is up and running and still connected to the network, then \( L_x \) remotely reactivates the agent on that node using the agent activation message. If the node does not respond, then \( L_x \) reports the problem to the administrator, excludes the node from the cluster or virtual cluster, updates the leader’s resources information, and informs all other nodes on the cluster or virtual cluster of the changes. If the node is restored later, the agent on that node informs the leader of its recovery and updates the cluster and itself with the local routing information.

5 COLLABORATIVE RESOURCE DISCOVERY

The agents in the run-time support layer take advantage of the hierarchical structure to simplify many of their operations. The agents in this structure also utilize automatic startup and configuration mechanisms and dynamic agent allocations that reduce user involvement. The hierarchical structure also provides other advantages such as: (1) providing scalable mechanisms to easily expand the system. (2) Providing the update and recovery mechanisms for automatic detection of agent failures or change of status and techniques to report errors and adapt to changes. (3) Providing routing capabilities in the leaders to facilitate process communications across multiple platforms over multi-hop links. (4) Making the agents management and monitoring operations more efficient and less dependant on the full connectivity of the system. In addition, the hierarchical structure and the introduced protocols provide a robust basis for providing resource discovery services. In this section will explore the details of our proposed collaborative resource discovery service based on the agents’ hierarchical organization. The main goal is to have, on request, an updated view of all system resources and their utilization and accessibility status for the entire environment.

5.1 System Resources
Many resources are available in a distributed environment including physical resources such as the CPU, storage and network components and services such as communications protocols, schedulers and I/O and storage controllers. To manage these resources, some information about them need to be collected and kept up-to-date. Any resource would have two types of attributes to describe it; quantitative and descriptive attributes. The quantitative attributes include measurable parameters such as processor speed and utilization, storage type and capacity, storage access time, memory capacity, network
bandwidth, latency, and delays, and service execution time. These attributes are usually easily obtainable and can be compared to each other for selection and management purposes. In addition, some of these attributes are relatively static and can be collected once in the environment. However, many resources can change some of these attributes dynamically depending on the systems utilization and loads. For example, storage capacities change with time by storing or removing components, network bandwidth changes dynamically with its utilization, memory capacity also changes with load changes, and processor speeds vary depending on the number of processes it is executing concurrently. As a result, in an ideal situation continuous measurement of these attributes is necessary to keep the system up-to-date at all times. Resources also have many characteristics and descriptive attributes that are very helpful for system management. Information regarding the resource access policies, security levels, ownership, authentication rules, and reliability levels. These attributes define many of the resource characteristics and suitability of utilization by different applications. Furthermore, many of these are not directly comparable to each other and could lead to difficulties deciding on the utilization of resources when needed. Both types of attributes are necessary for any resource management service in a distributed environment. Although some of these attributes are static, many others dynamically change depending on the current state of the system and the applications utilizing it. As a result, continuous monitoring and collection of the status of these attributes is necessary to optimize utilization. However, this process is lengthy and time consuming in a distributed environment.

5.2 Discovery Mechanism

We base this mechanism on the hierarchical structure of the agents. As the agents begin the startup protocol the process of resource discovery also starts. We will modify the agent protocols introduced in Section 3 to facilitate their collaboration to collect the necessary information. Every leader will have to carry part of the overall responsibility of discovering and keeping track of all the resources under their control. In addition, leaders need to effectively communicate this information to other leaders as necessary. We will describe the mechanism in three different stages. The first stage is the startup phase where initial systems resources information are gathered and shared among leaders. At this phase static attributes are identified, measured, and exchanged and the initial status of the remaining attributes are also collected. The second phase is a periodic discovery task that monitors the dynamic attributes and keeps track of their status. In addition, leaders need to exchange some of this information, while maintaining low overhead on the system. Finally, on request of resources leaders will need to communicate and collectively find the resources requested by the application.

5.2.1 Startup Phase

When a leader is activated, one of its initial duties is to activate other agents to form its virtual cluster and establish communications with other leaders. The leader startup protocol handles this stage. However, to include resource discovery we modify the protocols by adding the following steps.

1. Initialize the resources database and identify local resources (within the machine running the leader).
2. When agents in the virtual clusters are activated, the leader requests resources information from these agents.
3. The leader sorts the information and maintains the two types of information (static and dynamic) separately.
4. The leader then aggregates the information to provide a single system image of the resources in the virtual cluster. This information will allow external entities to view this virtual cluster as a single machine having the collective pool of resources.
5. The leader then sends the collective resource information to other leaders at the same level. And to its own Leader, if one exists. For example Leader 4 in Figure 2 will send its information to Leader 3 only, while Leader 2 will need to send its information to both Leaders 1 and 3.

When information becomes available, agents and leaders communicate through the hierarchical structure where agents collaborate to satisfy user job requirements efficiently. As discussed earlier, the leaders will keep the information about the different resources within its own machine and all the agents within its virtual cluster. However, the dynamic attributes about these resources will change over time. This brings us to the second phase of the discovery mechanism.

5.2.2 Periodic Updates

In this phase, which continues throughout the agents and leaders lifetime, is responsible for keeping the resources information updated and ready for use. To achieve this goal the agents will exchange the necessary information such that communication is minimized. This can be achieved by following thoses steps for each period. These steps are performed in conjunction with the agent update protocol.

1. The leader probes its descendant agents for changes in their status. Each agent will respond to its leader with only the changes that occurred during the time from the last check or with a no-change message.
2. The leader will compile the information received to update its virtual cluster resource information and keep track of the changes made.
3. The leader then sends the updated information to its own leader if it exists.
4. The leader also communicates with the other leaders at the same level to determine if it is necessary to pass the changes to them.

The periodic availability checks and updates can be fine-tuned to the system properties to minimize the number of checks performed. This mainly relies on the stability of the system used and the frequency of applications executions. If the system is stable and has low probability of failures, then the period between checks can be set to be long, thus reducing the total messages exchanged. However, if the system includes unreliable components or is connected through unreliable communications links, the period should be short enough to discover failures and recover quickly to minimize job failures. In addition, with the increase of the number of jobs executing, the agents will need to keep track of the processor, memory, and storage utilization more frequently.

Although the need for an updated view of the system is mainly needed when a new application is being submitted, we recommend the periodic updates for three reasons. One is the need to keep track of active agents and of any failure in the system to enhance system stability and reliability. The second is that the periodic checks will help minimize the startup time for the new application since leaders can use the latest information available from the last update. However, a mechanism to determine when to use this information and when to collect more updated information is necessary to ensure latest resources information. Finally, the periodic check can discover recently released resources when applications terminate.

5.2.3 Selection phase
The first two phases prepare the system to serve any application requiring resources on the system. The periodic checks and updates maintain a complete view of all available resources and their current status. However, at the time when the application is submitted for execution, it is necessary to ensure complete and accurate resources information. Thus it may be necessary to repeat the update process to acquire the latest resources information from the other leaders. At the same time, performing the full discovery process is lengthy and may delay the application’s execution. Therefore, we rely on the local leader to coordinate this process and provide the required information quickly. The following steps will be followed by the local agent (where the job is submitted).

1. If the local agent is not a leader, then it transfers the job request to its leader to find the required resources.
2. The local agent’s leader first checks the time of the last update. If it was done within an acceptable time frame and no jobs were scheduled during that time, then it uses the information to allocate resources to the application.
3. If the update was done before the specified time frame or a new application has just started executing on the virtual cluster, then the agent will need to coordinate a collaborative process to find the changes to the resources.
4. To do that, the agent performs two checks concurrently; probe its own resources (descendant agents) and contact its leader to get more resource information about other virtual clusters in the system. The leader send the information currently available (does not perform a check at this time). In case the requester will require resources; the leader will perform a check of its current resources status to be ready for the request with the latest information.
5. If the resources available to the local agent satisfy the application’s demands, then the leader executes the application locally on the virtual cluster.
6. Otherwise, the leader uses the information received from its leader to decide where to schedule the application and transfer the request to the leader.
7. At this point, the leader should have compiled the most current information and can schedule the application immediately.

One important issue to be considered is deciding weather to perform resource checks or not when a new application is submitted. The periodic check usually provides an almost accurate view of the systems resources. However, if the application is submitted near the end of the check period, some of the information may have changed either due to some applications ending their executions or by applications that just started execution. As a result, a threshold value is needed to trigger a new check for the new application. This value can be agreed upon by the agents based on the history of the system utilization and the stability of the resources status.

The combined efforts of the agents in the system will provide a complete view of the systems resources while minimizing disruption to the applications. The exchange of information is kept to a minimum due to the hierarchical organization and the use of the virtual clusters. Most of the information is localized and exchange and update is limited to the significant changes in the systems resources. In addition, the changes propagate through the leaders in the hierarchy as needed and do not flood the whole system.

6 CONCLUSION
In this paper we introduced a mechanism for a collaborative resource discovery in large homogeneous and heterogeneous clusters. The resource discovery process is a costly operation and costs increase drastically for large distributed systems. In addition, distributed applications usually require multiple resources and need to be scheduled efficiently and in a timely manner. The mechanism introduced is part of the distributed systems
middleware framework, Delmon, which provides services to support the development of distributed applications on clusters and heterogeneous systems. The distributed agents in the run-time support layer support distributed programming models, tools and applications. The agents use a number of protocols to self-organize in a hierarchical structure. Some of the main advantages of using distributed agents and the hierarchical structure are portability, Expandability, Flexibility, Security, and Resources Management. The resource discovery aspect of distributed systems resource management schemes is introduced as one of the services of the run-time support layer. Using the hierarchical structure we provided an efficient mechanism to discover and update resources information. Agents collectively have information about all the resources, which provides a distributed information base of system resources. Thus they can collaborate to provide efficient and comprehensive resource discovery and management tools. The collaboration provides a single system image within each virtual cluster and collectively for the complete cluster, while minimizing the message exchange among agents system-wide. As a result new applications are provided with the necessary resources in an efficient and timely manner.

References
