An Agent-based Service Discovery Algorithm Using Agent Directors for Grid Computing

Leila Khatibzadeh[1], Hossein Deldari[2]
[1]Computer Department, Azad University, Mashhad, Iran
[2]Computer Department, Ferdowsi University, Mashhad, Iran

Abstract - *Grid computing has emerged as a viable method to solve computational and data-intensive problems which are executable in various domains from business computing to scientific research. However, grid environments are largely heterogeneous, distributed and dynamic, all of which increase the complexities involved in developing grid applications. Several software has been developed to provide programming environments hiding these complexities and also simplifying grid application development. Since agent technologies are more than a ten-year study and also because of the flexibility and the complexity of the grid software infrastructure, multi-agent systems are one of the ways to overcome the challenges in the grid development. In this paper, we have considered the needs for programs running in grid, so a three-layer agent-based parallel programming model for grid computing is presented. This model is based on interactions among the agents and we have also implemented a service discovery algorithm for the application layer. To support agent-based programs we have extended GridSim toolkit and implemented our model in it.*

Keywords: Agents, Grid, Java, Parallel Programming, Service Discovery Algorithm

1 Introduction

Grid applications are the next-generation network applications for solving the world's computational and data-intensive problems. Grid applications support the integrated and secure use of a variety of shared and distributed resources, such as high-performance computers, workstations, data repositories, instruments, etc. The heterogeneous and dynamic nature of the grid requires its applications' high performance and also needs to be robust and fault-tolerant [1]. Grid applications run on different types of resources whose configurations may change during run-time. These dynamic configurations could be motivated by changes in the environment, e.g., performance changes, hardware failures, or the need to flexibly compose *virtual organizations* from available grid resources [2]. Grids are also used for large-scale, high-performance computing. High performance requires a balance of computation and communication among all resources involved. Currently this is achieved through managing computation, communication and data locality by using message-passing or remote method invocation [3]. Designing and implementing applications that possess such features is often difficult from the ground up. As such, several programming models have been presented.

In this paper, we have proposed a new programming model for Grid. Numerous research projects have already introduced class libraries or language extensions for Java to enable parallel and distributed high-level programming. There are many advantages of using Java for Grid Computing including portability, easy deployment of Java's bytecode, component architecture provided through JavaBeans, a wide variety of class libraries that include additional functionality such as secure socket communication or complex message passing. Because of these, Java has been chosen for this model. As agent technology will be of great help in pervasive computing, Multi-Agent systems in aforementioned environments pose important challenges, thus, the use of agents has become a necessity. In agent-based software engineering, programs are written as software agents communicating with each other by exchanging messages through a communication language [4]. How the agents communicate with each other differs in each method. In this paper, a three-layer agent-based model is presented based on interactions between the agents and a service discovery algorithm is implemented in this model.

The rest of the paper is organized as follows: In Section 2, a brief review of related works on programming models in Grid and a review of service discovery algorithms are given. Section 3 presents the proposed three-layer agent-based parallel programming model in details. The model in which the service discovery algorithm has been presented is shown. Simulation results are presented in Section 4. Finally, the paper is concluded in Section 5.

2 Related works

There are different kinds of programming models, each of which have been implemented in different environments. The summary of these programming models are briefly explained below.

- Superscalar is a common concept in parallel computing [5]. Sequential applications composed of the tasks of a certain granularity are automatically converted into a parallel application. There the tasks are executed in different servers of a computational Grid.
- MPICH-G2 is a Grid-enabled implementation of the Message Passing Interface (MPI) [6]. MPI defines standard functions for communication between processes and groups of processes. Using the Globus Toolkit, MPICH-G2 provides extensions to MPICH. This gives users familiar with MPI an easy way of Grid enabling their MPI applications. The following services are provided by the MPICH-G2 system: co-allocation, security, executable staging and results collection, communication and monitoring [5].
- Grid–enabled RPC is a Remote Procedure Call (RPC) model and an API for Grids [5]. Besides providing standard RPC semantics, it offers a convenient, high-level abstraction whereby many interactions with a Grid environment can be hidden. GridRPC seeks to combine the standard RPC programming model with asynchronous course-grained parallel tasking.
- Gridbus Broker is a software resource that transparently permits users access to heterogeneous Grid resources [5]. The Gridbus Broker Application Program Interface (API) provides a straightforward means to users to Grid-enable their applications with minimal extra programming. Implemented in Java, the Gridbus Broker provides a variety of services including resource discovery, transparent access to computational resources, job scheduling and job monitoring. The Gridbus broker transforms user requirements into a set of jobs that are scheduled on the appropriate resources. It manages them and collects the results.
- ProActive is a Java-based library that provides an API for the creation, execution and management of distributed active objects [7]. Proactive is composed of only standard Java classes and requires no changes to the Java Virtual Machine (JVM). This allows Grid applications to be developed using standard Java code. In addition, ProActive features group communication, object-oriented Single Program Multiple Data (OO SPMD), distributed and hierarchical components, security, fault tolerance, a peer-to-peer infrastructure, a graphical user interface and a powerful XML-based deployment model.
- Alchemi is a Microsoft .NET Grid computing framework, consisting of service-oriented middleware and an application program interface (API) [8,9]. Alchemi features a simple and familiar multithreaded programming model. Alchemi is based on the master-worker parallel programming paradigm and implements the concept of Grid threads.
- Grid Thread Programming Environment (GTPE) is a programming environment, implemented in Java, utilizing the Gridbus Broker API [1]. GTPE further abstracts the task of Grid application development and automates Grid management while providing a finer level of logical program control through the use of distributed threads. The GTPE is architected with the following primary design objectives: usability and portability, flexibility, performance, fault tolerance and security. GTPE provides additional functionality to minimize the effort necessary to work with grid threads [1].
- Open Grid Services Architecture (OGSA) is an ongoing project that aims to enable interoperability among heterogeneous resources by aligning Grid technologies with established Web services technology [5]. The concept of a *Grid service* is likened to a Web service that provides a set of well-defined interfaces that follow specific conventions. These Grid services can be composed into more sophisticated services to meet the needs of users. The OGSA is an architecture specification defining the semantics and mechanisms governing the creation, access, use, maintenance and destruction of Grid services. The following specifications are provided: Grid service instances, upgradability and communication, service discovery, notification, service lifetime management and higher level capabilities.

Dynamic service discovery is not a new issue. There are several solutions proposed for fixed networks, all with different levels of acceptance [10]. We will now briefly review some of them: SLP, Jini, Salutation and UPnP's SSDP.

The Service Location Protocol (SLP) is an Internet Engineering Task Force standard for enabling IP network-based applications to automatically discover the location of a required service [11]. The SLP defines three "agents", User Agents (UA) that perform service discovery on behalf of client software, Service Agents (SA) that advertise the location and attributes on behalf of services, and Directory Agents (DA) that store information about the services announced in the network. SLP has two different modes of operation. When a DA is present, it collects all service information advertised by SAs. The UAs unicast their requests to the DA, and when there is no DA, the UAs repeatedly multicast these requests. SAs listen to these multicast requests and unicast their responses to the UAs [10].

Jini is a technology developed by Sun Microsystems [12]. Its goal is to enable truly distributed computing by representing hardware and software as Java objects that can adapt themselves to communities and allow objects to access services on a network in a flexible way. Similar to the Directory Agent in SLP, service discovery in Jini is

based on a directory service, named the Jini Lookup Service (JLS). As Jini allows clients to always discover services, JLS is necessary for its functioning.

Salutation is an architecture for searching, discovering, and accessing services and information [13]. Its goal is to solve the problems of service discovery and utilization among a broad set of applications and equipment in an environment of widespread connectivity and mobility. Salutation architecture defines an entity called the Salutation Manager (SLM). This functions as a directory of applications, services and devices, generically called Networked Entities. The SLM allows networked entities to discover and use the capabilities of other networked entities [10].

Simple Service Discovery Protocol (SSDP) was created as a lightweight discovery protocol for the Universal Plug-and-Play (UPnP) initiative [14]. It defines a minimal protocol for multicast-based discovery. SSDP can work with or without its central directory service, called the Service Directory. When a service intends to join the network, it first sends an announcement message to notify its presence to the rest of the devices. This announcement may be sent by multicast, so that all other devices will see it, and the Service Directory, if present, will record the announcement. Alternatively, the announcement may be sent by unicast directly to the Service Directory. When a client wishes to discover a service, it may ask the Service Directory about it or it may send a multicast message asking for it [10].

In this paper, according to the models presented, we have integrated the benefits of message passing through explicit communication. In addition, we have made use of the Java-based model because of the similarity of portability to that of the distributed object method. Agents that act like distributed threads were also utilized. Playing the role of the third layer of this model, the service discovery algorithm was also implemented.

3 The three-layer agent-based parallel programming model

In this section, a three-layer agent-based parallel programming model for grid is presented and the communications among agents classified into three layers:

- *The lower level* is the transfer layer which defines the way of passing messages among agents. In this layer, the sending and receiving of messages among agents are based on the UDP protocol. A port number is assigned to each agent which, in turn, sends and receives messages.
- *The middle level* is the communication layer which defines the manner of communicating among agents. There are different communication methods.

Direct communication, such as contract-net and specification sharing, has some disadvantages [4]. One of the problems with this form of communication is the cost. If the agent community is as large as grid, then the overhead in broadcasting messages among agents is quite high. Another problem is the complexity of implementation. Therefore, an indirect communication method using Agent Directors (ADs) has been considered. An AD is a manager agent which directs inter-communication among its own agents as well as communication between these agents and other agents through other ADs. Figure 1 shows the scheme of this communication. Message passing and transfer rate are reduced in this model. Depending on the agent's behavior (requester/replier), two AMS (Agent Management System) are considered in each AD. In each AMS, there is information about other agents and their ADs which directs the request to the desired AD as necessary. As for the two AMS, time spent seeking in the agent platform decreases.

Based on the AD's needs and the type of request, the AD's table of information on other agents in other ADs is updated in order to produce a proper decision. This update reduces the amount of transactions among ADs.

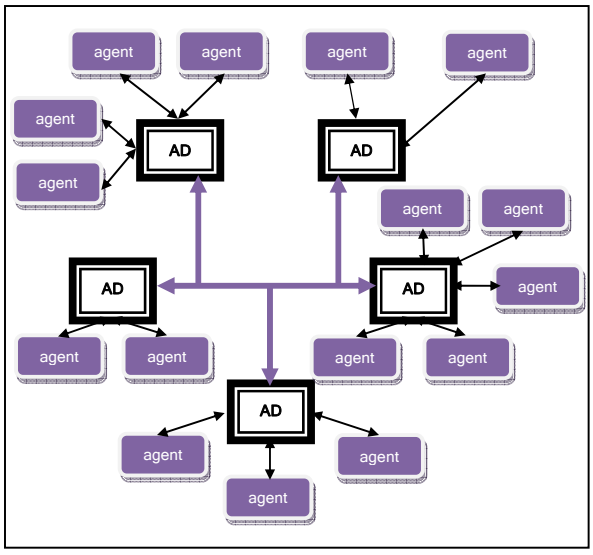

Figure 1. The scheme of communication among agents in the presented model

The following parameters are considered for each AMS when being stored in the AMS table:

- AID: Is a unique name for each agent which is composed of an ID of the name of the machine where the agent was created.

- State: Shows the state of the agent. Three states are possible in our model:
  - I. Active State: When the agent is fully initiated and ready to use.
  - II. Waiting State: When the agent is blocked and waiting for a reply from other agents.
  - III. Dead State: When the agent has completed its job. In this state, information about this agent is removed from AMS.
- Address: Shows the location of an agent. If an agent has migrated to other machines, this is announced to the AD.
- AD name: Shows the name of the AD associated with the agent. This parameter is essential as communication among agents is performed through ADs.
- Message: Shows the message sent from an agent to the AD. According to the agent's behavior, different kinds of messages are generated. The content of the message shows the interaction that is formed between the AD and the agent. Based on FIPA [15], we have considered ACL (Agent Communication Language) with some revisions. Below is the list of the different message types:
  - If the agent acts as a requester, then the content of the message, based on the program running on grid, explains the content of the request, e.g., a required service in the service discovery algorithm.
  - As the agent is created, the message content includes the agent's specifications for registering in AMS.
  - If the agent acts as a replier, the message content, based on the program which is running on grid, explains the content of the reply for that request.
  - When the job that is related to the agent is completed, the message content informs AD to remove this record from AMS.
  - While a program is running, if an error occurs, the message content informs AD.

We have considered a history table which keeps information about agents which have completed their jobs. The jobs done through AD are reported to the user.

In this model, AD decides to send the request to its own agents or to other agents in other platforms. This decision is made according to the information that is stored in the replier AMS and the condition of agents.

- *The higher level* is the user application layer, which defines the application running through software agents. The application used in this model for testing is a service discovery algorithm. The algorithm implemented is based on SLP and Jini. We have considered three kinds of agents similar to those of SLP:
  1- User Agents (UA) that run service discovery on behalf of client software.
  2- Service Agents (SA) that announce location and attributes on behalf of services. In the implemented algorithm, each SA may have different kinds of services. The creation and termination of services are dynamic and the list of services that each SA owns dynamically changes. The root of action of SAs in our model is similar to that of process groups in MPI.

Directory Agents (DA) that store information about services informed in the network are similar to JLS in Jini. In this model, each AD contains one DA. Because the structure of ADs includes histories of the processes that each agent performs, this model is effective for algorithms such as the service discovery. With two different histories implemented for each AD, the way of searching services in grid has been facilitated.

4 Simulation Results

This model has been implemented in Java by using the GridSim toolkit [16]. We extended the GridSim toolkit to implement agent-based programs with this simulator. The *gridsim.agent* package has also been added, and three different models have been implemented. The three-layer agent-based parallel programming model presented in this paper has been compared with these models through a service discovery algorithm.

The first model involves message passing among nodes that form the service discovery algorithm. The second, called Blackboard, is a simple model for communication among agents [4]. In this communication model, information is made available to all agents in the system through an agent named "Blackboard". The Blackboard agent acts like a central server. The third model is our own and was fully explained in Section 3.

In order to evaluate our model, we measured the number of messages which were sent by agents until the time all user agents obtained their services. Three different methods were considered for this algorithm:

1- Message passing: In this method, there is no difference between the types of agents. All agents act like nodes in the message passing method.

2- Blackboard: In this method, the Blackboard agent acts like a Directory Agent.

3- Agent Director: In this method, each AD acts like a DA.

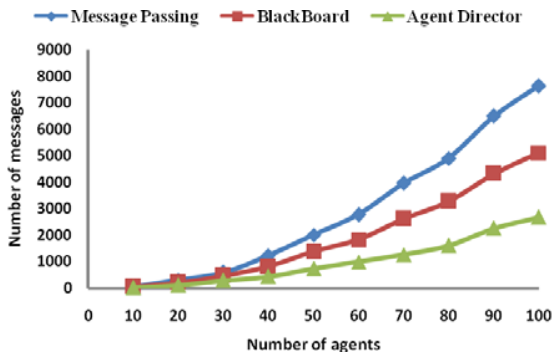

Figure 2. The comparison of the average number of messages received by agents

In Figure 2, the vertical axis represents the average number of messages and the horizontal axis represents the number of agents. In the first method, the number of agents means the number of nodes. In the other methods, the number of agents is the sum of the three types of agents earlier explained.

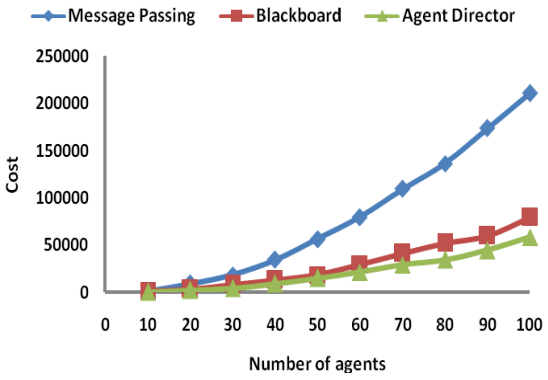

Figure 3. Cost Vs. Number of Agents

The results show that the average number of messages increase as the number of agents increase. This is quite obvious, because as the number of agents increase, the communication among agents also increases.

It is observed from Figure 3 that if the number of agents (nodes in message passing) exceeds 40, the cost of the communication in message passing implementation rises suddenly. As the number of nodes increases, the number of messages performing service discovery grows, along with the cost of communication. In other words, due to the lack of a database which could store the specifications of nodes having services, the cost of message passing rises suddenly when the number of agents exceeds 40. However, AD performs better than Blackboard. We estimate the cost of each method as follows:

- In message passing, the cost is estimated to be between 10 and 100.
- In Blackboard, the cost between agents and the Blackboard agent is estimated to be between 10 and 50.
- In AD, because of the two existing types of communications, two different costs were calculated. One is for ADs and is estimated to be between 50 and 100. The other is between AD and agents, which is estimated to be between 10 and 20.

These estimations are derived according to the method's actions.

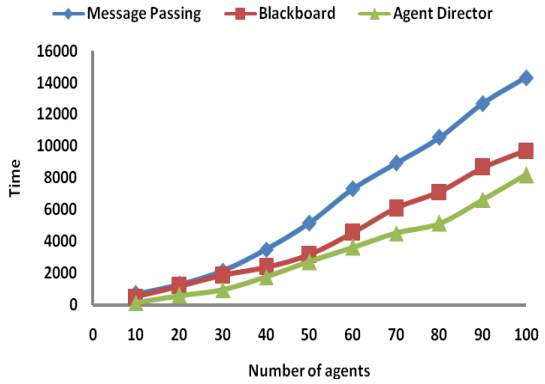

Figure 4. Time for different service discovery algorithms Vs. number of agents

It is obvious from Figure 4 that with an increase in the number of agents, execution time also increases. Due to the agent-based nature of Blackboard and AD, the time in which all user agents reach their services is shorter.

5 Conclusion and Future Work

In this paper, we have studied different programming models that had been previously presented for grid. Because of the Java advantages for Grid Computing, among others, portability, easy deployment of Java's bytecode, component architecture and a wide variety of class libraries, it has been chosen in this research. As agent technology is very effective in pervasive computing, the use of Multi-Agent systems in pervasive environments poses great challenges, thus, making the use of agents a necessity. In agent-based software engineering, programs are written as software agents that communicate with each other by exchanging messages through a communication language. How the agents communicate with each other differs in each method. In this paper a three layer agent-based programming model has been presented that is based on

interactions among agents. We have integrated the benefits of message passing through explicit communication and made use of a Java-based model because its portability is similar to that of the distributed object method. In addition, agents acting like distributed threads were utilized and so formed a three-layered programming model for Grid based on agents. We have extended the GridSim toolkit simulator and added the *gridsim.agent* package for agent-based parallel programming. A service discovery algorithm has been implemented as the third layer of the model. Using our model, measured parameters, such as the number of messages, cost and execution time, resulted in a better operation compared to that of the other methods.

6 References

[1] H. Soh, S. Haque,W. Liao, K. Nadiminti, and R. Buyya, "GTPE: A Thread Programming Environment for the Grid", Proceedings of the 13th International Conference on Advanced Computing and Communications, Coimbatore, India, 2005

[2] I. Foster, C. Kesselman, and S. Tuecke. "The anatomy of the grid: Enabling scalable virtual organizations". Intl. J. Supercomputer Applications, 2001.

[3] D. Talia, C. Lee, "Grid Programming Models: Current Tools, Issues and Directions", *Grid Computing*, G. F. Fran Berman, Tony Hey, Ed., pp. 555–578, Wiley Press, USA, 2003.

[4] C. F. Ngolah, "A tutorial on agent communication and knowledge sharing", University of Calgary, SENG609.22 Agent-based software engineering, 2003.

[5] H. Soh, S. Haque, W. Liao and R. Buyya, "Grid programming models and environments", In: Advanced Parallel and Distributed Computing ISBN 1-60021-202-6

[6] N. T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface", Journal of Parallel and Distrbuted Computing (JPDC), vol. 63, pp. 551-563, 2002.

[7] ProactiveTeam, "Proactive Manual REVISED 2.2", Proactive, INRIA April 2005.

[8] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A .NET-Based Enterprise Grid Computing System", Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), June 27-30, 2005, Las Vegas, USA.

[9] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework", High Performance Computing: Paradigm and Infrastructure, L. Y. a. M. Guo, Ed.: Wiley Press, 2005.

[10] Celeste Campo, "Service Discovery in Pervasive MultiAgent Systems", pdp_aamas2002, Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices 2002 Bolonia, Italy.

[11] IETF Network Working Group. "Service Location Protocol", 1997.

[12] S. Microsystems. "Jini architectural overview". White paper. Technical report, 1999.

[13] I. Salutation Consortium. "Salutation architecture overview". Technical report, 1998.

[14] Y. Y. Goland, T. Cai, P. Leach, and Y. Gu. "Simple service discovery protocol/1.0". Technical report, 1999.

[15] http://www.fipa.org/

[16] R. Buyya, and M. Murshed, *GridS*im: "A Toolkit for the Modeling, and Simulation of Distributed Resource Management, and Scheduling for Grid Computing", The Journal of Concurrency, and Computation: Practice, and Experience (CCPE), Volume 14, Issue 13-15, Pages: 1175-1220, Wiley Press, USA, November - December 2002.