GSF: A Problems Solving Environment Supporting Multi-models Parallel Grid Programming

Qian-ni Deng and Xin-da Lu

Dept. of Computer Sci. & Eng., Shanghai Jiaotong University, Shanghai 200030, P.R .China
{deng-qn, lu-xd}@cs.sjtu.edu.cn

Abstract. Web service is a grid computing technology that promises greater ease-of-use and interoperability than previous distributed computing technologies. In this paper we propose Group Service Framework(GSF), a grid problems solving environment based on Microsoft .NET, to: (1) locate and harness volunteer computing resources, and (2) support multi parallel programming paradigms such as Master/Slave, Divide and Conquer, Phase Parallel and so forth in Grid environment, (3) allocate tasks and realize load balancing dynamically and transparently for different grid application.

Keywords. Web Service; Volunteer computing; Grid computing; parallel programming paradigm

1 Introduction

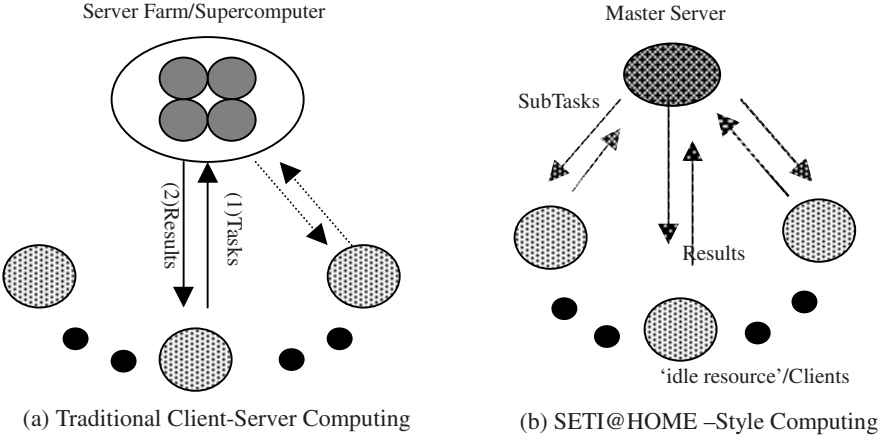
Volunteer computing has the idea that home PCs are mostly idle and thus could be harnessed for solving complex computational problems. This idea is to decompose the problem into many chunks that can run concurrently with very little interaction, referred by some as "embarrassingly parallel" computation. The best known example of this idea are SETI@HOME. A comparison of traditional high performance server versus volunteer computing is shown in Figure 1. Though volunteer computing model has been successfully used to handle some complex computation problems, there are many inherent shortcomings in this model:

(1) Volunteer computing can not be used to solve the computation problems that have to be divided into several chunks which run concurrently with interactions.
(2) Volunteer computing only supports Master/Slave programming paradigm.
(3) Generic volunteer computing system is application-specific and only can be used for one application.
(4) Lack of security and dynamic management mechanisms.

Aiming at above problems，we brought forward a grid computing framework and built a generic problem solving environment based on Master/Group/Worker model and Web Service[3] technology. Unlike those application-specific volunteer computing systems, this problem solving environment not only can locate and harness volunteer computing resource for different applications, but also support multi-programming models such as Master/Slave, Divide and Conquer, Phase Parallel [5] and so forth parallel programming paradigms. The main contribution of this paper is

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 413–419, 2003.

that we attempt to explore a grid-oriented generic programming model well combined with web service technology.



(a) Traditional Client-Server Computing (b) SETI@HOME –Style Computing

Fig. 1. A comparison of traditional computation servers versus resource sharing approach

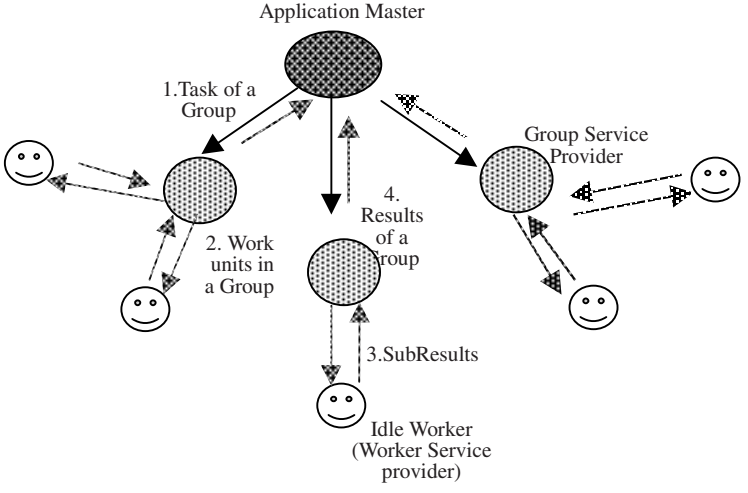


Fig. 2. Master/Group/Worker Style computing

2 Master/Group/Worker Model

Fig 2 shows the Master/Group/Worker model. There are three main roles:

(1) Worker Service Provider: A worker would be an idle PC who can provide volunteer computing service intermittently.

(2) *Group Service Provider:* Group service provider has the abilities to locate worker service providers and bring together the discovered workers into a group to solve an SPMD problem. In the lifetime of a group service provider, it locates workers, decomposes the task of a group into several work units, sends work units to located workers who perform computing. From the point of view of an application master, one group service provider can be looked as an SPMD computational web service. In our problem solving environment it can exists one or more group service providers.

(3) *Application Master:* In our problem solving enviroment an application can be of SPMD or MPMD programming model,when the application is SPMD model,only one group service can satisfy its computing request, when the application is MPMD model, serveral group services are needed. For example, to calculate the formula:

$$(u_1^2+u_2^2+\ldots+u_n^2\ ) / (A[1]*B[1] + A[2]*B[2] + \ldots + A[m]*B[m]) \quad (1)$$

The Application Master just need to divide the task into two groups, one calculates:

$$x=u_1^2+u_2^2+\ldots+u_n^2 \quad (2)$$

the other calculates:

$$y=(A[1]*B[1] + A[2]*B[2] + \ldots + A[m]*B[m]) \quad (3)$$

Firstly the two group service providers decompose the task in each group into work units respectively and parallely according to the partition strategy specified by application master, then the two groups work respectively and independently to compute the value of x and y, finally the application master callbacks the value of x and y from the two group respectively and compute the final result: *x/y*.

In above mentioned model a programmer need not to specify the number of workers in advance, it just need to specify the data partition strategies when the programmer define the application master. The benefits of this model are: easy to programming, dynamic load distribution and load balance. Apparently, besides Master/slave model, the architecture in fig.2 can support " divide and conquer" programming paradigm. If the process shown in fig.2 repeats several times, it can support "phase parallel" programming paradigm.

3 Data Partition and Merging Strategy

3.1 Automatic Data Partition

To partition the task of an SPMD program in a group automatically，it needs to specify the data partition strategies of a group when the programmer define the application master. By defining an object of C# class: DataStructure, the programmer can specify the dataset and partition strategie of one SPMD task.

```
public class DataStructure
{  const int BROADCAST=0;
```

```
const int PER_ELEMENT=1;
const int SAME_SIZE=2;
const int DIFFERENT_SIZE=3;
int dds;    //data dispatching strategy, range from 0~3
int num_of_worker;
String datatype; //it can be any C# data type
byte[] data; // the date set
int[] data_size;   }
```

Fig. 3. C# class : DataStructure

This structure supports following partition strategies：

- BROADCAST, broadcast the whole dataset to all workers in a group.
- SAME_SIZE, averagely partition the dataset to each workers according to the amount of works in a group.
- PER_ELEMENT：send first element of data-set to first worker, second element to second worker，the rest may be deduced by analogy until last worker receive the element. If the number of elements is greater than the number of workers, the residual elements may be allocated in terms of above rule until all elements in data-set been allocated.
- DIFFERENT_SIZE：each worker in a group sees after different size computing tasks. By defining the value of num_of_worker and data_size[] array in the class DataStructure，the application programmer can specify the allocation scheme and the number of elements allocated to each worker in a same group.

3.2 Automatic Computation Results Merging Strategy

To automatically merge sub-results collected from group service providers, it needs to specify the results merging strategies when the application programmer defines the *application master*. By defining an object of a C# class: Recv_DataStructure, the programmer can specify the result sets and merging strategy of an application. The merging strategies supported by our problems solving environment include: calculating maximum, minimum, summation and product.

```
public class Recv_DataStructure {
String recvtype;
int recvcounts;
byte[] recvbuf;
const int NOTHING=4;
const int MAX=0;  static final int MIN=1;
const int SUM=2;  static final int PROD=3;
int  merge_strategy; // range from 0~4
String result_type; // type of merged results
int result_counts;  //number of merged results
byte[] result;  }
```

Fig. 4. C# class: Recv_DataStructure

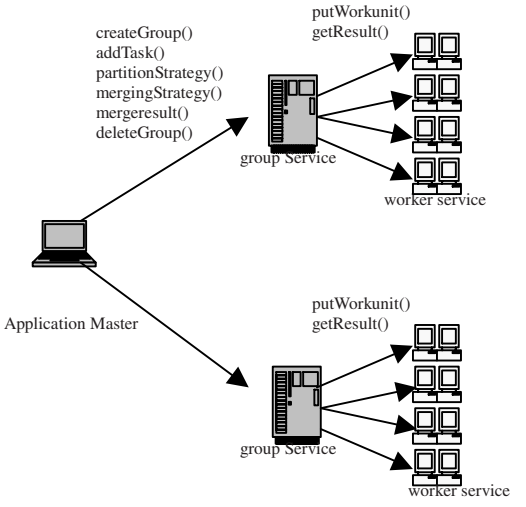


Fig. 5. Group Service Framework

4 Group Service Framework

We have already implemented the group service framework (GSF) and developed a genetic parallel problems solving environment based on Microsoft .NET. Microsoft .NET is a set of software technologies designed to connect information, people, systems, and devices. The foundation of .NET is XML Web services: reusable applications written in Extensible Markup Language that allows users to connect with applications and data on the Internet. Based on the platform of .NET, all the roles (Workers, Groups, Masters) in our problems solving environment are defined by WSDL language and communicate with each other by XML language. Fig. 5 shows an example of how group services hide volunteer computing for an MPMD application.

Recently, the Globus group has started integrating web service with globus toolkits, and defined a standard architecture OGSA[1,2]. OGSA gives a standard of how to define the grid service and doesn't limit the implementation of grid service. We think that our group service is one kind of grid services who can server for many parallel computing applications. To accord with OGSA(Open Grid Service Architecture), we plan to re-define and implement all the service interface following the standards of OGSA specification. Fig. 6 shows the future Group Service Framework coincident with OGSA.

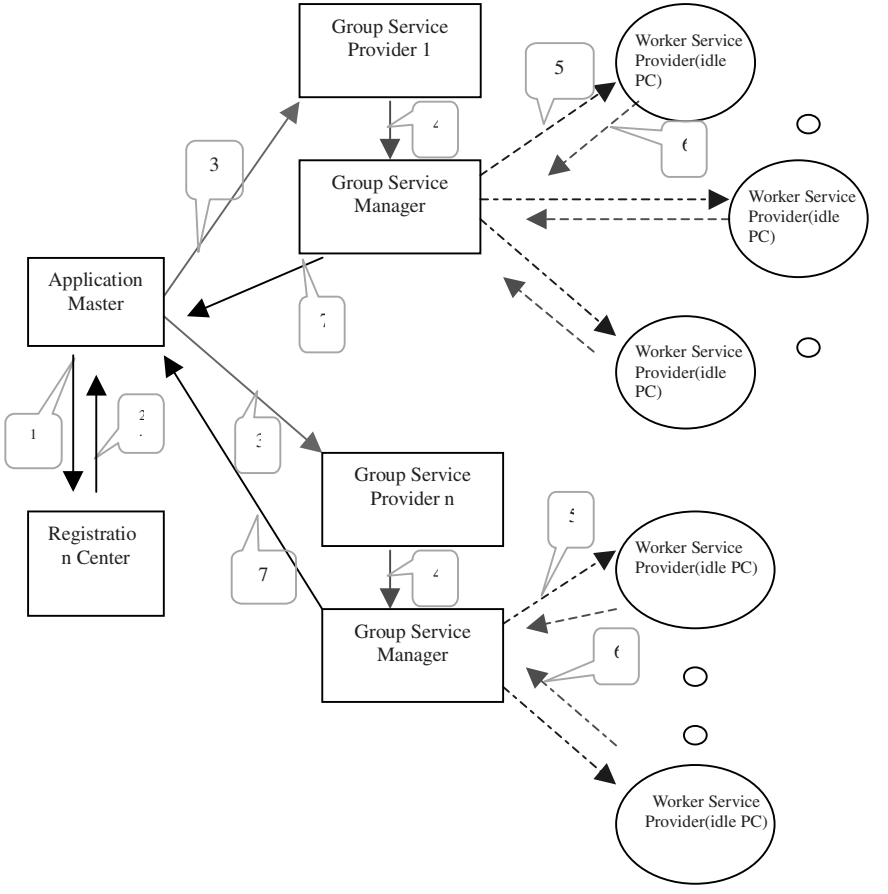


Fig. 6. Group Service Framework according with OGSA

According with OGSA, our future Group Service Framework should work as following processes:

(1) *Application master* submits one or more group service requests to *Registration center;*
(2) *Registration center* returns one or more *group service handlers* to *Application master;*
(3) *Application master* defines the lifetime of group service, asks *group service provider* for creating the grid service instance;
(4) *group service provider* creates an instance;
(5) *group service instance* discovers *worker services* , then dispatchs work units;
(6) after completing computing, *worker* returns the sub-results;
(7) *group service instance* keeps lifetime, returns the group results to *Application Master.*

5 Results, Related Work, and Conclusion

Currently, we have used Grid Service Framework based on Microsoft .NET and Java platform to implement several simple parallel computing applications, including π calculating parallel program, Mandelbrot set application, and parallel string matching, and have been able to use these with different kinds of computational worker clients including a .NET application running on PC and a Java applet running on non-PC. The related experiment results have already been presented on [4]. From the experiments we get the conclusion that Group service Framework proposed by this paper is suitable for generic parallel numerical computing. These results are just the beginning of our research in which grid computing and web services can be used together. We plan to redefine the Group service Framework according to OGSA specification and continue developing Group Service Framework as well by exploring issues such as authentication, authorization, and resource accounting.

References

[1] Ian Foster, Carl Kesselman , et al. Open Grid Service Infrastructure WG, Global Grid Forum. The Physiology of the Grid, An open Grid Service Architecture for Distributed Systems Integration. http://www.globus.org

[2] S.Tuecke, K. Czajkowski, I.Foster, J.Frey, S.Graham, C.Kesselman; Open Grid Service Infrastructure WG, Global Grid Forum. Grid Service Specifications. Draft 2, 7/17/2002. http://www.globus.org

[3] Microsoft Corporation.XML and .NET White Papers. http://www.microsoft.com/serviceproviders/whitepapers/xml.asp

[4] QianNi Deng, XinDa Lu. JOGR-Utilizing Java Object Group Relationship to Support Multi-model Parallel Programming, ACTA ELECTRONICA, Nov. 2002.

[5] Kai Wang, ZhiWei Xu. Scalable Parallel Computing: Technology, Architecture, Programming. McGraw-Hill. 1998.