Towards a Service-based Collaborative Framework for Data-intensive Grid Applications

Hsi-Min Chen[1], Chao-Chin Chang[2], Jan-Jan Wu[2], Chien-Min Wang[2], Chun-Chen Hsu[2]
[1]*Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan*
[2]*Institute of Information Science, Academia Sinica, Taipei, Taiwan*
{seeme, preisner, wuj, cmwang, tk}@iis.sinica.edu.tw

Abstract

As data-intensive applications increase continuously in various domains, scientists nowadays need to save, retrieve and analyze rapidly increasing large datasets. The long latency of data transfer over Internet results in a serious challenge on ensuring high-performance access to large quantities of data. Furthermore, the vision of Grid researches is to provide a collaborative space where scientists can share their data, resources and experiences with others. These issues motivate us to develop a service-based collaborative framework for data-intensive Grid applications. In this paper, we shall describe our initial work towards this goal. We plan to provide an integrated and virtual infrastructure, which manages distributed and diverse data resources and services. By means of Web Services technologies, we can realize the interoperability among various types of resources and integrate computing services with storage services to serve data-intensive applications. By making use of portals and data-intensive applications built on the top of the framework, users will be able to manage and analyze large quantities of data with high performance and share useful data with others in a way of forming virtual groups to complete cooperative work.

1. Introduction

As data-intensive applications increase continuously in various domains, such as high-energy physics, astrophysics, genomic computing and digital library, scientists now need to save, retrieve and analyze rapidly increasing large datasets. These data are gathered from scientific instruments, sensors, sensor networks and other data-collection devices. Data Grid [1] is a distributed storage infrastructure that integrates distributed, independently managed data resources. It addresses the problems of storage and data management, data transfers and data access optimization, while maintaining high reliability and availability of the data. In recent years, a number of Data Grid projects emerge in various disciplines, for instance, EU DataGrid [2], PPDG [3], etc. Moreover, as we see the evolution of Gird infrastructures from native Girds, OGSI [4] to WSRF [5], Web Services technologies [6] play a more and more important role for new generation Grids. By means of Web Services, we can realize the interoperability among various types of resources and improve the scalability of a Grid system.

The long latency of data transfer over Internet results in a serious challenge on ensuring high-performance access to large quantities of data. Furthermore, the vision of Grid researches is to provide a collaborative space where scientists can share their data, resources and experiences with others. The high-performance and collaboration issues motivate us to develop a service-based collaborative framework for data-intensive Grid applications. Our framework provides an integrated and virtual infrastructure, which manages and integrates distributed and diverse resources and services.

The rest of the paper is organized as follows. In the section 2, we describe the architecture of the proposed framework in detail. Section 3 explains the current implementation of our framework. In the section 4, we illustrate an application example built on the framework. Finally, we present some concluding remarks in the last section.

2. Service-based Collaborative Framework

Figure 1 shows the architecture of the proposed framework. All that users can use are front-end applications and web portals in the top layer. The middle layer consists of web services, including data management, workspace, registry, compilation, etc. These web services furnish the above layer with storage and computing supports and connect to underlying data and computing resources. In the following, we will describe each layer of the architecture in detail.

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

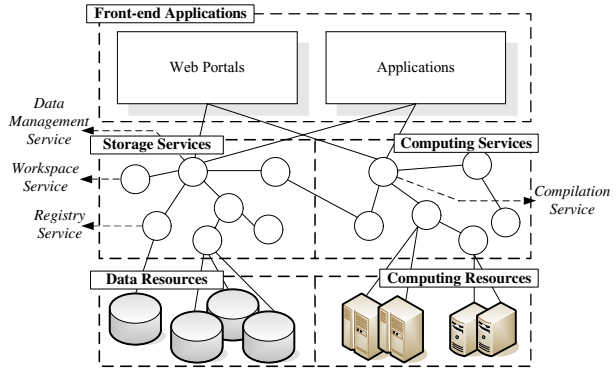

Figure 1. The layered architecture.

2.1. Portal and Applications

The top layer of our framework includes web portals and data-intensive applications. Grid portal is a web site that allows scientists, researchers and application users to access remote resources via a familiar web interface. The portal of our framework authenticates users to remote resources and services, and facilitates data manipulations by interacting with the services resided in the middle layer.

To overcome the representative limitations of web interfaces, our framework also supports data-intensive Grid applications that are developed as desktop ones for rich rendering.

2.2. Web Services

We adopt Web services technologies that can interoperate across heterogeneous resources and components by means of standard transport protocols and XML-based messages. In our framework, the middle layer comprises a number of web services providing various features. Moreover, it enables scalability so that we can add new services with ease to meet new requirements. Because services are composable, we can assemble a new service by reusing existing ones and have no need to create from scratch. Web Services technologies reduce the coupling and complexity of implementations and improve the interoperability and scalability of our framework.

2.3. Resources

There are three categories of data resources that our framework plans to support initially: parallel file systems, i.e. PVFS (Parallel Virtual File System) [7], Relational Database, and WWW. To alleviate I/O bottleneck of data Grid, we adopt PVFS as the primary data storage of our framework in which applications can access experimental data in a way of parallel I/O. PVFS also supports several features, including file striping across nodes, multiple access interfaces, utilizing commodity network and storage hardware, etc. By using PVFS, high-performance data accesses are improved.

3. Implementation

The implementation of proposed framework supports functionalities for web portal and applications. The fundamental infrastructure consists of data management, data registry, workspace, compilation, and computing services.

3.1 Data Management Service

Data management service is a high-level one that provides users of web portal or applications to manage their data. It supports basic data operations such as listing, cutting, copying, deleting, and renaming data. Furthermore, it integrates with other services to furnish advanced functionalities. Data management service provides a uniform interface by which web portal and applications can access the underlying storage services in the proposed framework.

3.2 Data Registry Service

The data registry, as shown in figure 2, keeps records associated with storage data. When users create a piece of data, such as saving a PVFS file, a directory and a link, the service will register it in the registry and ask users to annotate it for searching. A registry entry includes a unique identifier, a name, a set of metadata that describe the data, an access control list, and a link pointing to the physical storage data. Users don't care about which physical storage stores the data, and they simply need a logical identifier to look for it.

Since the registry keeps a large mount of entries pointing to the physical data, it is difficult for users to get a desired one with identifiers. The service provides searching function by which users can look storage data up with ease.

3.3 Personal and Group Workspace

Collaboration is one of crucial visions for Gird that allows users to share their data, resources, and experiences. In our framework, personal and group workspaces enable collaboration of users. Personal workspace is one that users can construct their favorite data links in a structure way. Figure 2 shows the relationship among data registry, storage data, and personal and group workspaces. All of data are saved by owners in the virtual storage and

registered in the registry. Data owners can share their data with others by writing access control lists that allows group members or others to access the data.

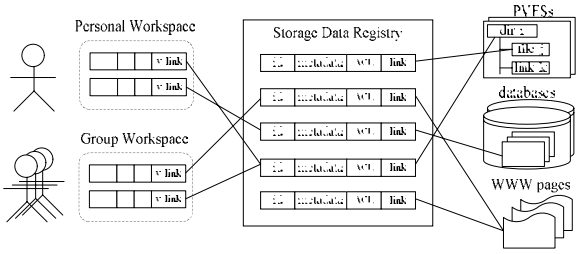

Figure 2. Data registry and workspaces.

3.4 Compilation Service

A uniform global name space over dispersed physical data storages helps users manipulate their data with transparency. However, it is incompatible with existing programs that access data directly through native I/O calls. Therefore, we provide a new library of I/O calls that can access globally named files. By replacing native I/O library with the new one, existing programs can access files in the dispersed data storages as if these files are in local disks. Another advantage of this approach is that it has the flexibility to implement more performance optimizations in the future.

A similar problem arises when users compile their programs on the proposed framework. The physical locations of source files in the proposed framework are unknown to the compilers. To overcome this problem, we build a compilation service so that users can compile their programs as if these files are in local disks. The compilation service and the new I/O library make user data accessible not only to the users themselves but also to their programs. With these supports, users are able to analyze large quantities of data stored in the dispersed data storages with programs.

4. Application Example

In this section, we will demonstrate the proposed framework with a data-intensive application on collaborative filtering recommendation [10]. A high-performance computing service for collaborative filtering is implemented. Users can combine the storage management service and the collaborative filtering service to find items in which they might be interested.

4.1 Collaborative Filtering

Collaborative filtering (CF) attempts to alleviate information overload by identifying which items a user will find worthwhile. It focuses on identification of other users with similar tastes and the use of their options to recommend items. In the past year, a wide range of web sites have begun to use CF recommendations in a diverse set of domains including books, grocery products, art, entertainment and information.

There are a variety of CF recommendation algorithms. In the following, we adopt the one used in the GroupLens system [8]. The input dataset can be modeled as a two-dimensional sparse matrix, where the columns are users, the rows are items, and the cells contain the ratings. A rating is a number from 1 to 5, with 5 highest and 1 lowest. Prediction can be modeled as matrix filling. A CF system predicts scores for missing cells before the users examine the corresponding items.

4.2 The Scheduling Algorithm

The prediction of scores in a CF system can be computed independently for each user. Let the prediction of scores for a single user is a task. A practical CF system will compute a lot of independent tasks. To speedup the computation, we can distribute these independent tasks to processors in computing resources. Such a process is called task scheduling. Each processor can read the necessary input data directly from data resources, compute its own tasks, and then store the result back to data resources.

Task scheduling can be done statically or dynamically. Since Grids are dynamic and heterogeneous in nature, it is more appropriate to schedule tasks dynamically. Self-scheduling is a dynamic scheduling method, in which each processor autonomously acquires a task for execution whenever it becomes idle. It is a good technique for balancing the load when task execution time is unpredictable. Furthermore, it offers the user a simple programming model. The main disadvantage is its large number of scheduling operations. The number of scheduling operations can be reduced if a block of tasks is allocated simultaneously. To get the best performance, the block size must be properly chosen to balance the scheduling overhead against the load imbalance. The determination of the best block size desires a better understanding of the scheduling overhead and the load imbalance. In the following, we define parameters used in modeling the performance of the Grid system and propose an adaptive algorithm to determine the block size.

- p: the number of processors.
- n: the number of tasks.
- r: the number of unallocated tasks.
- k: the number of tasks to be allocated and computed.
- t_s: the measured scheduling overhead of previous allocation on the same processor.

- t_p: the measured execution time of tasks in previous allocation on the same processor.
- t_1: the estimated execution time of a task on the same processor.
- μ_r: the expected execution time of unallocated r tasks.
- μ_n: the expected execution time of all n tasks.
- α : an adjustable parameter of load balancing.
- β : an adjustable parameter of efficiency.

Intuitively, the block size has an upper bound n/p and a lower bound 1. We shall set the block size to $k = \sqrt{n/p}$ in the first allocation of each processor so that the block size is neither too large nor too small for the subsequent allocations. In our algorithm, we use the scheduling overhead and execution time of previous allocation to predict the future performance on the same processor. Without knowing what will happen in the future, we simply assume the scheduling overhead is unchanged and the execution time of a task is a negative exponential distribution with t_1 as mean. Hence, the scheduling overhead of the new allocation will be t_s and the expected execution time will be $k \cdot t_1$. The expected imbalance time is approximated as $k \cdot t_1 \cdot e^{-\mu_r / kt_1}$. Although this model may not be very accurate, we can use the two adjustable parameters to control the inefficiency factor and the imbalance factor.

- Let $k \geq \beta \frac{t_s}{t_1}$, then the inefficiency factor = $\frac{t_s}{k \cdot t_1} \leq \frac{1}{\beta}$.
- Let $k \leq \frac{\mu_r}{\alpha \cdot t_1}$, then the imbalance factor $\leq \frac{r}{\alpha \cdot n} e^{-\alpha}$.

As indicated in the above, a larger block size can reduce the number of scheduling operations and the inefficiency factor. On the other hand, a smaller block size can make the distribution more even and reduce the imbalance factor. This implies the two adjustable parameters are related. If the inefficiency factor and the imbalance factor are treated as equal importance, we can combine these inequalities and derive the best block size as follows:

$$\beta \frac{t_s}{t_1} = k = \frac{\mu_r}{\alpha \cdot t_1} \text{ and } \frac{1}{\beta} = \frac{r}{\alpha \cdot n} e^{-\alpha}$$

$$\Rightarrow \beta = \frac{\mu_r}{\alpha \cdot t_s} \text{ and } \beta = \frac{\alpha \cdot n}{r} e^{\alpha}$$

$$\Rightarrow \alpha^2 e^{\alpha} = X \text{ where } X = \frac{r \cdot \mu_r}{n \cdot t_s}$$

Since it is difficult to get a closed-form solution for α, here we make another approximation. If $X \geq e$, let $\alpha = \ln(X)$. Otherwise, let $\alpha = X^{1/2}$. This choice of the block size looks reasonable. When there are a large number of unallocated tasks, a larger block size is preferred. On the contrary, a smaller block size will be better if only a small number of tasks remain unallocated. In addition, a higher scheduling overhead will tend to increase the optimal block size.

4.3 Performance Comparison

We have implemented the CF recommendation system in MPI over Grids. To explore and compare the performance of the proposed adaptive-block self-scheduling, fixed-block self-scheduling is implemented as a reference. There are also two versions of I/O schemes implemented: one uses sequential server I/O and data communication and the other uses PVFS I/O directly. Since PVFS provides a Unix I/O interface for programmers, it is much easier to use PVFS I/O than sequential server I/O and data communication.

The dataset for this demonstration is drawn from the EachMovie collaborative filtering dataset [9]. In this dataset, some 72916 users entered a total of 2811983 numeric ratings for 1628 different movies (films and videos). It has been used in numerous collaborative filtering publications. To explore the effect of different numbers and execution times of independent tasks, the demonstration is performed with three datasets of 2,500, 10,000, and 40,000 users, respectively.

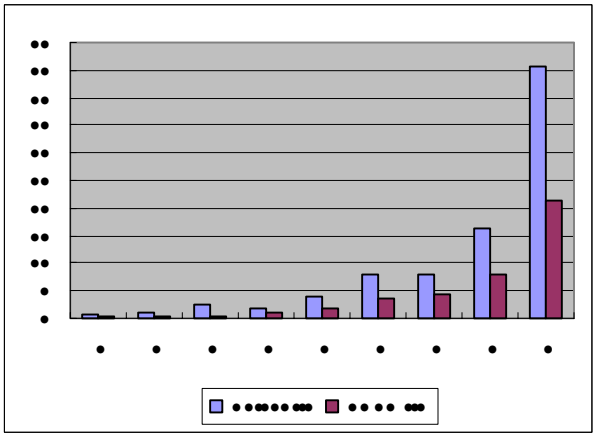

Figure 3. I/O Performance.

The experiment is conducted on the Taiwan UniGrid [11] using 4, 8, and 16 processors, respectively. First, we compare the performance of PVFS I/O and server I/O. As shown in Figure 3, in all the experiments, the performance of direct PVFS I/O is superior to the server I/O and data communication. This indicates that direct PVFS I/O not only simplifies the programming model but also improves the performance of MPI programs on Grid systems.

Next, we compare the performance of the two scheduling functions. To explore the best block size, 7

different block sizes are tested in the fixed-block self-scheduling. Since the block size varies in adaptive-block self-scheduling, we count the number of scheduling operations to compare it with fixed-block self-scheduling. The experimental results on 16 processors are shown in Tables I, II, and III. Similar results can be obtained from experiments of 4 and 8 processors and are omitted here due to page limitation. These results clearly show the impact of the block sizes on the performance. As we can see in these tables, a smaller block size results in a large number of scheduling operations that will slow down the performance. On the contrary, a larger block size also slows down the performance due to load-imbalance. It can be observed that the performance of adaptive-block self-scheduling is superior to any fixed block size in all the experiments. Although the proposed adaptive algorithm is not optimal in either scheduling operations or imbalance time, it provides the best trade-off between these criterions. This indicates that it is useful in real environments.

Table I. Experimental result for 2,500 users.

Table II. Experimental result for 10,000 users.

Table III. Experimental result for 40,000 users.

5. Concluding Remarks

In this paper, we present our initial work towards a service-based collaborative framework for data-intensive Grid applications. The fundamental infrastructure consists of data management, data registry, computing, compilation and workspace services. As a demonstration, we also implement a collaborative filtering computing service. The preliminary result looks promising. By means of Web Services technologies, we can integrate computing services with storage management services to serve data-intensive applications. Adaptive task scheduling algorithms and direct I/O to parallel file systems help the realization of high-performance computing and data access on Grid systems. There still remain many research issues to explore and services to implement. In the future, we will continue working towards this goal to make it a reality.

Acknowledgement

This work is sponsored by the National Center for High-performance Computing, Taiwan, under the grant NCHC-KING_010200.

Reference

[1] Ian Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, 15(3), 2001.

[2] EU DataGrid, available at http://www.edg.org.

[3] PPDG: Particle Physics Data Grid, available at http://www.ppdg.net/.

[4] S. Tuecke, et. al, "OGSI Specification Version 1.0," Global Grid Forum, available at http://www.gridforum.org/ogsi-wg/.

[5] Karl Czajkowski, et. al, "The WS-Resource Framework version 1.0," Global Grid Forum, available at http://www.globus.org/wsrf/.

[6] Web Services Activity, W3C, available at http://www.w3.org/2002/ws/.

[7] Philip H. Carns and Walter B. Ligon III, "PVFS: A Parallel File System for Linux Clusters," *Proc. of the Extreme Linux Track: 4th Annual Linux Showcase and Conference*, October 2000.

[8] P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl, "GroupLens: An open architecture for collaborative filtering of netnews," *Proceedings of the 1994 Computer Supported Collaborative Work Conference*, 1994.

[9] The EachMovie collaborative filtering data set, http://research.compaq.com/SRC/eachmovie/.

[10] J. Herlocker, J. Konstan, and J. Riedl, "Explaining Collaborative Filtering Recommendations," *Proceedings of ACM 2000 Conference on Computer Supported Cooperative Work*, pp. 241-250, December 2000.

[11] Taiwan UniGrid, http://unigrid.nchc.org.tw.