THCORE: A Parallel Computation Services Model and Runtime System*

Qingxuan Yin[1], Xiaoge Wang[1]
1 Department of Computer Science and Technology, Tsinghua University
Beijing 100084, China wangxg@tsinghua.edu.cn,
WWW home page: http://os.riit.tsinghua.edu.cn

Abstract. Wrapping parallel programs or parallel numerical library functions into software components and using them as computation services in service-oriented programming presents a method of delivering powerful computation capabilities of multi-processor supercomputers to the application developers who may only familiar with their desk-top or hand-held computing environment. These parallel computation services on computer clusters are used as ordinary software components on the desktop programming environment with their internal parallel or distributed characteristics hidden from the users. In order to use the parallel scientific computation applications and libraries as the software components conveniently in the development of new applications, a parallel computation service model and the runtime system that support this model on computer clusters are presented and some design and implementation issues are discussed in this article.

1 Introduction

Parallel and distributed computing on computer clusters is an effective way to speed up large-scale scientific computations. However, it is more difficult to build up applications on such environment than that on a sequential machine. A traditional way of developing a large application on such environment usually requires a great deal of tight collaboration between experts in computer architecture, algorithm design and application area and leads to a monolithic program. Although there are many successful sophisticated parallel scientific libraries and packages available such as PETSC [1, 2], SCALAPACK [3] and BLACS [4] for application developers, it still requires the application developers to have certain degree of expertise in programming on such parallel/distributed computing environment. When using these

* This work is supported by the National High-Tech Research and Development Plan of China (No.2003AA1Z2090), and by Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology.

libraries, one usually needs to know how to initiate such environment in the code before using the library functions and know how to compile his/her own code with the libraries on the parallel computer and to start the application using some job and resource management tools. The debugging of parallel program on a computer cluster is even more difficult. For those scientists and engineers who are not familiar with parallel/distributed programming on computer clusters, these difficulties could be a serious obstacle for development of applications requiring high performance computation.

Service-oriented programming is a way to ease the difficulties mentioned. It separates the development of computation services components from the development of applications composed from the services. The development of computation services put the efforts on the design and implementation of algorithms of the services to make them efficient and reliable on their runtime environment, while the development of applications using selected services would focus the efforts on the business logic and the workflow of the applications.

In this paper, THCORE, a component model and its runtime systems are introduced. It unifies the interface of components running on sequential and parallel/distributed computers. The implementation differences between the components on sequential and parallel/distributed machines are encapsulated inside the component and managed by the corresponding component runtime systems. In this model, application developers do not need to have knowledge of parallel programming but still could use the computation power of parallel computers. The parallel computation services implemented as the THCORE components are used to compose the application in the same way as other components. The well developed and frequently used mathematics libraries, legacy packages, and even applications could be wrapped into components to provide computation services for other applications. The detail of the parallel computational service model of THCORE and its runtime system on computer clusters are presented.

The rest of the paper is organized as the following. An overview of THCORE and other related works are presented in section 2. The parallel computing service component model will be introduced in Section 3. The component runtime system for computer clusters is introduced and discussed in Section 4. In Section 5 the performance issues is discussed. Some experiment results that evaluate the overhead of componentization is shown. Related work will be mentioned in Section 6. The last section gives the conclusion and direction of future work.

2 Overview of THCORE and Related Work

THCORE is designed as a lightweight, efficient and reflective component platform for pervasive computing. It is written in C for the best performance and minimum memory footprint. Its component model adopts the component object model of Microsoft's COM/DCOM with some new extensions to suit the pervasive computing environment that includes as well as embedded systems and high performance computer systems. THCORE supports the binary level interoperability protocol, transparent local/remote invocations as in COM/DCOM. It deploys a standard

runtime substrate that manages the execution context of components. For example, a component can be instantiated in different running spaces: it can be created in the same process of application for efficiency, or be created outside the application process for system isolation, or it could even be created on remote machine connected via network.

Multi-level reflection is one of the new features brought into THCORE component model. It provides the access to both interface-level and component-level Meta data. Interface-level Meta data contains the definition of interfaces, functions and parameters. It provides the finest grain of self-description information of the system, and it is obtained by using the IMetaInterface interface of THCORE component. With the help of this, middleware can dynamically load components and invoke the method without generating accessing code. Component-level Meta data is used to describe a component's requirement of execution context (such as hardware and OS requirement). The Meta data in this level also contains the information of the reliance of dependency that one component lies on others. Component-level Meta data can be accessed by ICompMetaInterface interface of THCORE component. This reflection feature is very useful in supporting adaptive programming.

Some system services are provided for programming with THCORE model and platform, including event service, cache service and parallel computing service. Some THCORE related research projects have been reported. PURPLE [12, 13] is a component-based reflective middleware for pervasive computing which is built upon THCORE platform. It provides support for adaptive context-aware programming. The structural and functional modules that compose the middleware platform are THCORE components. THAOP [14] is a lightweight and flexible Aspect Oriented Programming Framework based on THCORE component platform. It provides support for component-level AOP. FT_THCORE [15] is a fault tolerant extension of THCORE specification and platform. It implements the easy component replication and voting strategy so that it supports N-Version Programming.

As the applications in pervasive computing usually involve several different computation environments ranging from resource limited mobile/handheld devices to powerful multi-processor supercomputers, THCORE is design to provide the interoperability between computation components (services) on different environments. As it is lightweight, THCORE can be installed on the resource limited devices. The discussion of the extension on COM/DCOM for embedded devices will not be discussed in this paper. But, the model adopted from COM/DCOM may not be viable for the parallel program directly. There are two design issues need to be considered. First, we wish to hide parallel programming from the programming with THCORE. Second, we allow the services components to be implemented by parallel program and executed on parallel/distributed computer systems. The details of the design and implementation for the extension of component model for parallel computation services and the correspondent runtime system on the computer clusters are discussed in section 3 and 4.

It is not a new topic to hide the parallel programming from the development of applications while the computation power from parallel programming is used for the execution of the application. Take Matlab for example. It is well known that Matlab is a convenient tool for engineering computation. It does not introduce explicit parallel computation concept into its programming. But it may need large amount of

computation power so much that the parallel computing may become necessary. Researchers have presented many methods to make parallel Matlab. A survey was performed [9] and 27 parallel MATLAB projects, such as MatPar, MatLab*P and so on are found through extensive web searching. The approaches to make MATLAB parallel are different: some compile MATLAB scripts into parallel native code; some provide a parallel backend to MATLAB, using MATLAB as a graphical front-end; and some others coordinate multiple MATLAB processes to work in parallel [10]. Take Matpar [5] for example. Matpar is a software program in C/S model that allows MATLAB users to take advantage of a parallel computer. Some calls to certain built-in MATLAB functions are replaced with calls to Matpar functions on the client side. The code of Matpar function calls in turn initiates a session on a parallel computer. The parallel code uses parallel mathematical libraries to produce a solution that is sent back to the calling program. Because of the limitation of the model, Matpar is difficult to be reused to build other applications. And it is difficult for Matpar to deal with different software and hardware environment. As the compiler is special designed for Matpar and Matpar is based on some parallel mathematics libraries, it is quite difficult to expand the functions that Matpar is supporting.

As a component model designed specifically for high performance computing, CCA (Common Component Architecture) [11] is better known in the HPC community. In CCA, components interact with each other and with a specific framework implementation through standard application programming interfaces (APIs). Each component can define its inputs and outputs by using a scientific interface definition language (SIDL); these definitions can be deposited in and retrieved from a repository by using a CCA Repository API. The goal of CCA is to gain abstractions that capture high-performance concepts in component architectures, which can enable more efficient interactions between SPMD programs. There are also tools associated with CCA to help with decomposition of legacy code into CCA components for reuse. Although the goal of CCA is also to foster the component-oriented programming, there are two main differences from THCORE: the first is the user knowledge requirement. To use CCA, one needs to have certain knowledge in parallel programming. The second is in the way of component composition. CCA allows parallel component to be more tightly connected because the interface contains the information of “parallelism” while THCORE hide parallelism completely from the interface. Therefore, CCA is more suitable for the development of component based parallel applications while THCORE is better for the application deployed on the heterogeneous computing environment such as in pervasive computing scenarios.

3 The Model of the Parallel Computation Services

One of the characteristics of service-oriented programming is the separation of service interface from its implementation. A client requests a computation service by invoking its interface. The implementation of the computation service, whether in sequential or parallel program, is transparent to the user. To the client of the service, interface will take the input from the client and return the results of the computation

to the client all through one "port". This matches with the way of human thinking and it is easy for implementation of business logic. On the other hand, single processor may not provide enough power for the computation so that multi-processor computers are introduced into the application. It is a common solution for the fast computation. In such case, we wish a service could be running on multi-processor machine. However, as the execution of a parallel program are quite different from that of a sequential program, the structure of a parallel service on a computer cluster is different from its counterpart on desktop computers and other hand-held devices.

The design of the model for the parallel computation services on computer clusters takes two factors into consideration. First, a parallel program is most likely to have "sequential" entrance though more parameters may be required to start a parallel execution. By "sequential" entrance, we mean that the program has single starting point, and its input and output data are in a whole, rather than in the form of partitioned pieces. Therefore, it is viable to keep the interface of parallel computation services the same as the interface of other sequential computation services with a few extra parameters. In addition, all parallel computation service should have a main process/thread to act as a "driver" and entrance of the parallel program. It is responsible for partition, distribution and aggregation of data structures for parallel computation if necessary. Second, as the services are executed on multiprocessor computer systems, in general, the number of processors to be used should be specified in advance by someone in someway, and the mapping from program's logic process to the physical processors needs to be performed. To run a parallel program, one needs to submit the job via job management tool such as PBS [16]. The number of processors and other execution parameters are submitted to the job manager system and the manager will arrange the resources for the execution.

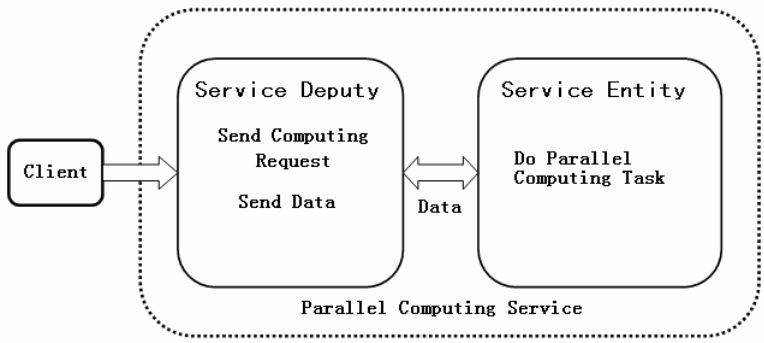

Fig. 1. Parallel Computation Service Model

The model of parallel computation service is shown in Figure 1. A service is made of two parts: service deputy and service entity. The service deputy is a sequential code. It acts as if it is the implementation of the service to the client, but in fact, it is only a "driver" and a "wrapper" of the parallel computation program. The service entity is a parallel code that implements the computation function and will execute on parallel computers or computer clusters in our case. The service deputy

receives the request from the clients and analyses the request to generate the information for the creation of the parallel tasks of the computation function including the task partition, data partition and distribution, synchronization mechanism of the algorithm, etc. The service deputy itself does not implement any parallel computing. The service entity receives the information from the service deputy and implements the parallel computation function. The input data sent from the client is received by the service deputy and then forwarded to the service entity accordingly. On return, the output data of the service moves on the same path from opposite direction to the client.

The course of making a parallel computation service is similar to that of making a normal component in THCORE except that the code of the parallel computation function has to be separated from the interface. It starts with the description of interface using IDL (interface definition language). The compiler of IDL will then generate the code of the interface. The interface provides an abstraction of its implementation and serves as the connection point between its client and the service it provides. In developing normal component, one could insert the code of the component function into the interface code. But in the parallel computation service case, the parallel computation function has to be separated. As the service deputy is actually only a "driver" and a "wrapper" of the parallel program, what contained in it is the information regarding the function identifiers, the code of creating multiple tasks and transferring data. Therefore, it can be generated by IDL compiler automatically as part of the interface code. Some sequential part of the parallel computation function, such as the pre/post processing steps, can be inserted into the interface code as the part of the service deputy if it is desired. The development of the service entity is similar to the development of a parallel program. The only difference is that the implementation of the parallel computation function has to be registered with the component runtime system as the interface. This is the result of the separation of sequential and parallel part of implementation.

The limitation of this model is that it does not provide the parallel interface for the composition of parallel computation services. It can be seen that the data in and out from the service component are packed into one single stream while the internal presentation is distributed for the parallel/distributed computation. If two parallel computation services are requested consecutively, the output data of the first service will be redistributed when it is used as the input data of the second services even though both services have the same internal distribution of the data. The time spent in the data movement would cause a serious problem in performance. The performance issues will be discussed in section 5.

4 The Service Runtime on Computer Clusters

For supporting the execution of the parallel computation service on computer clusters described in the previous section, a service runtime system is designed and implemented. This runtime system should have the following functions:

- **Service activation:** As the same as the function of THCORE runtimes on desktop computers and handheld devices, it should provide runtime

support for the service. When a service interface is invoked by a client, the runtime system should be able to activate the code, including both service deputy and service entity, to run on the cluster. It serves as a service container and connects the deputy and entity parts of the service.

- **Resource management:** As the amount of computation in a service may vary with the input parameters, such as the problem size and accuracy of the solution, the resource of the cluster, such as the number of processors, may vary from execution to execution. As an advanced function, the system should have the capability to allocate and manage the resource efficiently. It should dispatch the computation task to the processors appropriately in order to make good use of all the processors in the cluster for load-balance and better efficiency. In some sense, the runtime system performs the function of job manager like PBS and bears more responsibility for the success of execution of a parallel computation service than the operating system of the machine.

4.1 Architecture Overview

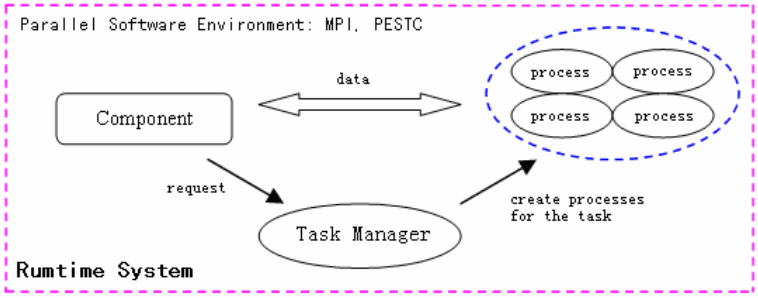

Fig. 2. Architecture of the service runtime system on computer clusters

The architecture of the runtime system for computer clusters is shown in Figure 2. In addition to the function of service activation, as THCORE runtime on desktops and handheld devices, it has a task manager module that performs the function of resource manager. It takes the information regarding the parallel computation characteristics of the service entity, such as the number of tasks to be created, from the service deputy, and then creates tasks and allocates the processors for the parallel tasks accordingly. When a client requests a parallel computation service by invokes its interface, the runtime system activated the interface code and the service deputy, a message of parallel computation task request is sent by the deputy to the task manager module of the runtime. The task manager parses and analyses the request, determines whether to accept this request or not, determines the number of processor to run the parallel tasks when the request is accepted, assigns processors to the tasks, and create a set of MPI [6, 7] processes on the processors for the execution of the parallel program of the service entity. The task manager is implemented as a MPI

process. It calls MPI library functions to create the MPI processes. The processes of the task receive the input data from the service entity directly, and send the computation result data back to the service deputy after the completion of the task. The processes of the parallel tasks will end themselves when the tasks are finished.

Service runtime system is running on a computer cluster. All the communication and data transfer on it are implemented through socket provided by the operating system. The location of the service deputy, the parallel processes of the service entity and the task manager of the runtime on cluster nodes is flexible. They are not necessarily located on the same node of the cluster. In general, each node of the cluster would host only one process of the tasks because running more than one processes on a node may reduce the efficiency of the execution.

4.2 Task Manager

The task manager is the core of the runtime system. It is consisted of five parts: the message queue of parallel task request/completion, the interface manager, the task dispatcher, the task scheduler and a queue of the tasks to be schedules,. The interface manager takes the message of task request from the message queue, analyzes the message, and creates the task for the request accordingly. The task dispatcher determines the number of processes that should be created for the task and dispatches the task to appropriately selected processors. The task scheduler schedules of the tasks in the waiting task queue. The flow chart of the task manager is shown in Figure 3.

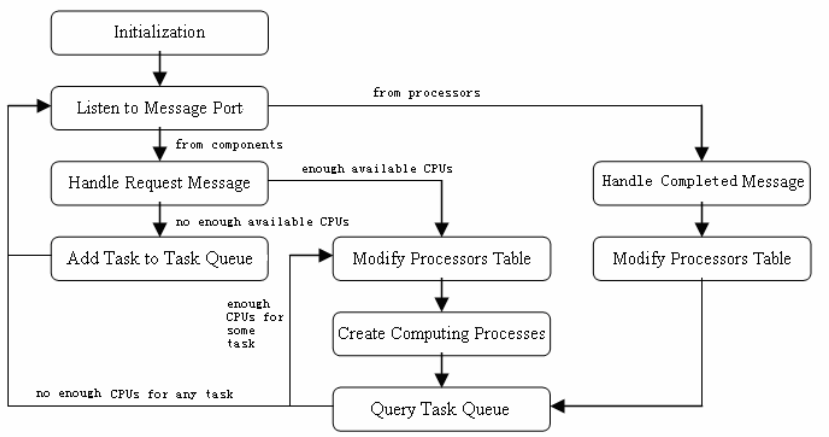

Fig.3. Flow Chart of the Task Manager

During the course of initialization, the message queue is set and a socket port associated with it is established to receive the message of task request from the service deputy and the message of task completion from task processes, a processor

table is created for the task dispatcher to record the status of the processors, and the task queue is created for the task scheduler.

After initialization, the task manager takes a message from the queue if there is any. If the message is a task request from the service deputy, the interface manager is called to analyze the request. Then the task dispatcher is called to determines how many processors are wanted for this task according to the available information such as the computational complexity of the task, the total number of the processors in the cluster, the number of the available processors in the cluster, the length of the task queue and possibly some other information. If the number of the processors wanted for the task is less than the number of the currently available processors in the cluster, the execution of the task could start. The task dispatcher is called to start the task. Otherwise, the task will be inserted into the task queue. In the situation that the number of the processors wanted by is greater than the total number of processors in the cluster, such as when the problem size is too large, the task request would be refused and the rejection message will be send to the service deputy.

The task dispatcher updates the processor table before it starts a computation task. Then it creates a set of processes to execute the parallel computation task. After the creation, the processes will communicate with the corresponding service deputy directly for the input data and start the computation. The task queue is handled by the task scheduler. A task in the task queue could be scheduled if there is enough number of processors available according to the processor table. If a task is scheduled, the task dispatcher is called to start it. The task will then be deleted from the queue. When a computation task is completed, a task completion message is sent to the task manager. On receiving the message, the task dispatcher updates the processor table. The task is finally accomplished at this point.

4.3 The Processor Management Strategy

The management of the computation and resources is the main concern of the runtime system design. The processors are the most important resource in the cluster. How to make the best use of the available processors is a question that designers of the runtime system must answer. Improving the efficiency of the processors and increasing the task throughput are the goal of the design of the resource management policy of the system.

The design of the processor management strategy is different for the traditional batch job management tools on clusters. For the transparency of parallel implementation of a parallel computation service, the number of physical/logical processors wanted to run the service is determined by the runtime system at runtime, while in most batch job manager tools, the number of processors needed to run a program is usually set by the owner of the program at the time when the job is submitted. The number of the processors assigned for the task is determined by the system at runtime according to the size of the task and the capacity of the cluster. The size of a task is measured by the computation complexity provided by the service provider in the interface description and the size of the computation input/output data available when the service is requested. The capacity of the cluster is measured by rate of the number of the total processors and the number of the available processors in the cluster. Generally the number of processors wanted by a

task is determined at the time when the task request is generated. We allow it to be changed later when necessary. For example, when a node of the cluster is down, the previously determined number of processors wanted for some tasks has to be reduced. Meanwhile, when a computer node is added into the cluster, the number of processors determined for some tasks may be increased appropriately.

When the number of available processors is large enough for more than one task in the queue, the selection of the task to be scheduled may influence the efficiency of the processors in the cluster. The policy for the scheduling is designed as the following:

A. Priority scheduling is applied. The task priority is the most important factor for determining the execution order of tasks. No tasks with lower priority could be executed unless there is no tasks with higher priority could be scheduled. Each task has its priority set when created. The priority could be determined according to several factors, such as the client's user ID, size of the task, even the time of task creation.
B. When several tasks are of the same priority and number of available processors is large enough for any of them, the task that needs more processors should be scheduled prior to those tasks that need fewer processors. The goal of this policy is to make more processors busy and schedule the large sized task as early as possible. In this way, the possibility of large sized tasks staying in the queue for a long time waiting for available processors could be reduced.
C. If several tasks have the same priority and need the same number of processors, the order them are scheduled to run is the same as the order they enter the queue. The task that is inserted into the queue earliest should be scheduled to run first.
D. Dynamic priority is introduced into task scheduling. A task, which has been waiting for certain period of time, should have its priority increased. Otherwise, a large sized task with lower priority may be waiting for too long to be acceptable.

The implementation of the design is currently undergoing. A prototype of the runtime for clusters is realized. On this prototype, some parallel computation services are developed for experiments. As the parallel libraries/packages are valuable legacies, we also developed some services by wrapping parallel mathematic library functions of PETSC. It is clear that the client of the parallel computation services does not need the knowledge of parallel programming. On the opposite, the developer of the services would appreciate greatly the knowledge and experiences of parallel programming. This is exactly one of the objectives of THCORE.

5 Performance Issues and Experiments

The performance of service runtime is important to the practices of service oriented software development. Compare to the monolithic program, the overhead of service oriented software comes from two main sources: additional code and data movement. Additional code is consisted of the code of service wrapper and the "glue code" that connects the service and its client. In our model of parallel computation

service, the code that connects the service deputy and service entity is considered as glue code as well. Data movement includes the movement between client and services and the movement inside the service between its deputy and entity. To evaluate the overhead of the proposed model and its runtime system, a simple experiment is conducted.

A parallel computation service is implemented to offer the service of solving linear systems. A client program requests the service by invoking the parallel computation service. Timers are installed inside the program to measure the time that spent by different portions of the program. The experiment is carried out on a small cluster of 2 nodes. Each node has a 2.4G CPU and 512M memory. The nodes are connected by 100M LAN. The OS is linux-2.6.16 and the parallel programming environment is LAM-MPI 7.1.2. The parallel computation service is implemented using PETSC version 2.3.1.

Following quantities are measured and the results are collected:

1. Service invocation time (T1): The time cost by invoking a non-parallel component of THCORE. This time could vary by the location of the service execution. When the service is executed in the same process as its client, this time will be the smallest. It is about 0.0015 seconds in the experiment. This quantity shows the minimum overhead when connecting service and client in THCORE.
2. Parallel service invocation time (T2): The time cost by invoking a parallel computation service only without data transferring and computation. It is about 0.4158 seconds. This quantity shows the minimum overhead when connecting a parallel computation service with its client. The big gap between T1 and T2 is from the additional work in activating additional processes and establishing the communication channels among the processors.
3. Data movement time (T3): The time cost by invoking a parallel computing component with data transferring but no computing. When the input data is of the size of 16400 double precision numbers and the output data is of the size of 4000 double precision numbers, it is about 8.660 seconds. The quantity T3-T2 tells the overhead of data movement for this test problem. It is determined mainly by the speed (latency and bandwidth) of the interconnection network between the nodes of the cluster.
4. Total overhead: We measure the total time of solving linear systems by invoking a parallel computation service (T4). It is about 59.020 seconds when using 2 nodes. It is about 104.625 seconds when using one node. As the comparison, we also measure the total time of solving the same problem using the same parallel functions from the library in a monolithic style of programming (T5). It is about 49.905 seconds when using 2 nodes and about 96.625 seconds using one node. The quantity T4-T5 gives the total overhead of service oriented programming in THCORE.

From the experiment results, we know that the overhead is mainly consisted of the data transferring. As the scientific computation often has a large size of input/output data, the cost of transferring data may become dominant. We should reduce the time cost by transferring data to make better efficiency. Besides the improvement of interconnection network, to increase the service granularity is a way

to reduce the data traffic. It should also be pointed out that the increase in the granularity of a service may reduce the reusability of the service.

During an invocation of a parallel computation service, the input data is sent from client to the service via interface and then distributed internally from the service deputy to the service entity and the output data is sent in the opposite direction back to the client. If all three parties could be located as close as possible, the efficiency could be increased. Considering the client is not located on the cluster, the possible location plans of the three parties are:

1. The service deputy is located together with the client in the same process and the client invokes the service in the way as a local component. The data transferring between the client and the service deputy could be ignored and only the remote data transferring between the service deputy and the entity remains.
2. The service deputy is located in a cluster node and the client invokes the service as a remote component. The data transfer will be the sum of the remote data transfer between the client and the service deputy and the remote data transfer among the nodes of the cluster.
3. The service deputy is neither located with the client nor with the entity in the cluster. Although this is not practical, it is valid in THCORE. In this case, the data transferring is consisted of the remote data transfer between the client and the entity and the remote data transferring between the deputy and the entity using the external network, which cost the most.

The first location plan is the most efficient. A mechanism to arrange the location of the parties may reduce the cost of data movement. To further reduce the data movement cost is an important issue of performance for the future work.

6 Conclusion and Future Work

This paper presents a parallel computation model for computer clusters and the design of its runtime system for supporting service-oriented programming on clusters. This model will bring the great convenience to the scientists and engineers who have to deal with large scientific computations but may not have enough experiences of parallel/distributed programming. A simplified prototype of the runtime system on cluster is realized for experiments. The preliminary experiments are conducted. The analysis of the results shows that the overhead of the service-based program comparing to the conventional parallel program is mainly from to the data movement. The strategy of overhead reduction includes the trade-off between the granularity of services and their reusability and the proper arrangement of the location of the parties associated in the data movement.

More study and further research efforts will be put in the investigation for efficient methods and tools to wrap the popular parallel libraries/packages into parallel computation services and the optimization of the performance of the runtime system. The future work will also include the study in the algorithms of resource management and the improvement of the quality of service of the runtime system. The coordination of the runtime system with the operating system of the cluster and

the batch job management software, such as PBS [16], to improve the performance of the applications and the throughput of the computer systems will be an interesting subject of the future research as well.

Acknowledgement

Authors are very grateful to all the members of the THCORE project team for their contribution to make this paper possible. They are Yu Chen, Xing Fang, Kuo Zhang, Yanni Wu, Du Zhao, Zhenkun Zheng and Gang Feng besides the two authors of this paper. Thanks also go to Professor Craig C. Douglas for his encouragement and helpful suggestions towards the research in this direction.

References

1. S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith, PETSc home page. http://www.mcs.anl.gov/petsc, July 1997.
2. S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith. PETSc 2.0 Users Manual, Tech.Rep. ANL-95/11 – Revision 2.0.22, Argonne National Laboratory, Apr 1998.
3. J. Dongarra and L.S. Blackford, ScaLAPACK tutorial, Proc. of Applied Parallel Computing, Industrial Computation and Optimization, Third International Workshop, PARA '96, Aug. 1996.
4. J. Dongarra and R. C. Whaley, A User's Guide to the BLACS v1.1 , Technical Report CS-95-281, University of Tennessee, Knoxville, Tennessee, 1997.
5. NASA Jet Propulsion Laboratory, Matpar; http://www-hpc.jpl.nasa.gov/PS/MATPAR
6. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. University of Tennessee, Knoxville, Version 1.1, June 1995.
7. Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface. University of Tennessee 1997; http://www.mpi-forum.org/docs/docs.html
8. X.G. Wang, X. Fang. THCORE home page; http://os.riit.tsinghua.edu.cn
9. R. Choy. Parallel MATLAB survey, 2001; http://www.interactivesupercomputing.com/reference/ParallelMatlabsurvey.htm
10. M.D. Barnell, B.J. Rahn. Migrating modeling and simulation applications on to high performance computers. SPIE Vol. 6227, p. 198-205
11. R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker and, B. Smolinski, Toward a Common Component Architecture for High-Performance Scientific Computing, Proceedings of the 8th IEEE International Symposium on High-Performance Scientific Distributed Computing, August 1999
12. K. Zhang, X.G Wang, Y.N. Wu, Z.K. Zheng, PURPLE: a reflective middleware for pervasive computing, ICITA 2005: the third Information Technology and Applications 2005, vol.1, p. 64- 69, Tsinghua University, July 2005
13. K. Zhang, Y.N. Wu, Z.K. Zheng, X.G Wang, Y. Chen, A component-based reflective middleware approach to context-aware adaptive systems, ICWE2005: the fifth International Conference on Web Engineering, vol. 3579, p. 429-434, Tsinghua University, July 2005

14. G. Feng, Q.X. Yin, X.G. Wang, THAOP: An Aspect Oriented Programming Framework, SPCA 2006: the first International Symposium on Pervasive Computing and Applications Proceedings, p. 127-132, Tsinghua University, August 2006
15. Q.X. Yin, G. Feng, X.G. Wang, Y. Chen, Increase Reliability of Pervasice Oriented Component Platform via N-Version, SPCA 2006: the first International Symposium on Pervasive Computing and Applications Proceedings, p. 95-98, Tsinghua University, August 2006
16. Veridian Systems, OpenPBS v2.3: The Portable Batch System Software, Veridian Systems, Inc., Mountain View, CA, September 2000. http://www.openpbs.org/scheduler.html