Component-Based Distributed Computing for Numerical Simulation

BEN CHOI

Computer Science
College of Engineering and Science
Louisiana Tech University, LA 71272, USA

pro@BenChoi.org

CHAOYANG ZHANG

Department of Computer Science
University of Vermont, VT 05405, USA

czhang@emba.uvm.edu

Abstract. This paper describes the use of a component-based distributed computing, DCOM, for a numerical simulation problem. Component-based distributed computing, such as CORBA, DCOM, and Java RMI, by distributing objects to be executed by networked computers, network of workstations, or PC clusters, has potential for use in large scale scientific simulation problems. DCOM developed by Microsoft is used here to exploit large installation base of networked Windows computers, such as PC labs in our campus, which is used in our experiments. A numerical heat transfer problem is used to illustrate the development of a DCOM application. The application dynamically partitions (and distributes) the problem into a number of components based on the number of available networked computers. Our tests show that large overhead in communications between the components and between the networked computers making such a distributed computing system sometimes performs slower than a simple computer.

Keywords: Distributed Computing, Network of Workstations, DCOM, Remote Method Invocation, Component-based Computing, and Numerical Simulation

1. Introduction

For many scientific simulation problems, large computing resources that require the use of supercomputers are needed. One affordable alternative is to develop distributed computing systems that use the resources available on a large number of networked computers, such as network of workstations and PC clusters. Distributed computing splits an application into tasks that executed at different locations, using different resources such as processors, memories, and storage devices. This technology allows an application to be built using many existing pre-built software modules (called components) that reside in different machines. Protocols and standards to promote distributed computing, such as Common Object Request Broker Architecture (CORBA), Java's Remote Method Invocation (RMI), and Distributed Component Object Model (DCOM) have been developed in recent years to facilitate the communications between the components [1-8].

Ben Choi and Chaoyang Zhang, "Component-Based Distributed Computing for Numerical Simulation," International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 2143-2149, 2002.

This paper emphasizes on DCOM technology and its applications. A component-based distributed computing application has been developed using DCOM technology for solving computational problems making use of available networked resources. A dynamically problem-partitioning method and a distributed topology have been explored in scientific application context. With security authentication, a client can check and remotely activate the available components deployed on servers, choose components to complete particular tasks, and specify all the computational parameters at runtime. Several aspects in distributed computing for solving scientific problems are discussed to optimize the functional behavior of a component and performance of distributed computing.

Component Object Model/Distributed Component Object Model (COM/DCOM) was developed by Microsoft for developing software components [9]. COM/DCOM itself is a specification and many programming languages can be used to write COM/DCOM components. DCOM is a protocol that enables software components to communicate directly over a network in a reliable, secure and efficient manner. DCOM technology is primarily applied in Windows-based environments and best supported on Windows NT-XP platforms. For our project the distributed computing application is developed under the following environments: COM/DCOM, Visual C++ 6.0, Microsoft Interface Definition Language (MIDL), Active Template Library (ATL), Microsoft Foundation Class (MFC), and Windows NT 4.0. After components have been developed they are deployed on several NT machines in our lab. In addition, several important aspects such as deployment, security configuration, dynamically remote activation, and data passing efficiency are discussed. The performance analysis of the distributed computing system is also provided.

2. Partitioning a Problem into Components

To make use of distributed resources, a computing problem is partitioned into many sub-problems that are solved by several networked computers. For scientific computation, a computing task is usually partitioned into a number of subtasks by decomposing its domain into a number of sub-domains. We use a 2D heat transfer problem in microstructure fabrication [10] to illustrate how to dynamically partition a computing problem. We assume that the length and width of the domain are L and W, respectively, as shown in Figure 1, in which four sub-domains are indicated.

Our application dynamically partitioned the problem into a number of sub-domains at runtime based on the number of available components in the networked computers. For simplicity, one sub-domain can be solved by one computing object in one networked computer. However, the one-to-one mapping does not take load balancing into consideration. The optimum way to partition domain should attempt to partition the entire computation domain to balance computing load, reduce data transfer between computers, and reduce

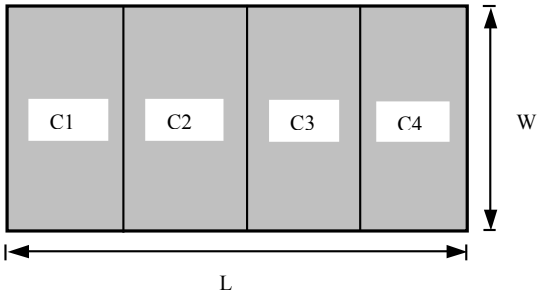

Figure 1. Problem partitioning

the number of remote procedure call (RPC).

Reducing data transfer between computers is done by reducing the length of boundaries between sub-domains for our heat-transfer simulation problem. During the simulation, it is necessary to pass the boundary conditions of one sub-domain to its adjoining sub-domains (Figure 2). In Figure 2(a), B0 is the boundary between two adjoining sub-domains COMi and COMi+1. COMi needs to pass data (boundary condition) to COMi+1 and vice versa. Bi and Bi+1 are used to represent the boundary data that need to be passed. Assuming that W < L, then the entire heating surface is divided into slices as shown in Figure 1.

3. Implementation Using DCOM Technology

DCOM objects are installed and registered in server computers. A client computer requests service from the server computers by specifying various computational parameters and send messages to the servers. After the completion of the computations, the results are returned to the client.

We developed DCOM objects for the specific purpose of simulation heat transfer. Each of DCOM objects is identical in functionality but has a universally unique identity (UUID). For specifying interface of DCOM objects, we designated memory management by using IDL attributes. Our DCOM objects support dynamic invocation, which is made possible by inheriting *IDispatch* interface. Dynamic invocation allows an object to dynamically inquire other objects at runtime to see whether they support particular interface methods. This is different from static invocation where the interface details and the complete method signatures are known at compile time.

We developed a client object to interface with the DCOM objects running on the servers. There are several methods for a client to call a server and use the DCOM objects, such as #import technique and manual technique [11]. Their advantages and disadvantages of each technique depend upon our needs and preferences. For example, #import techniques is easy to use but lack flexibility. On the other hand, the more flexible techniques such as manual technique are more complex. To make dynamic remote calls, manual

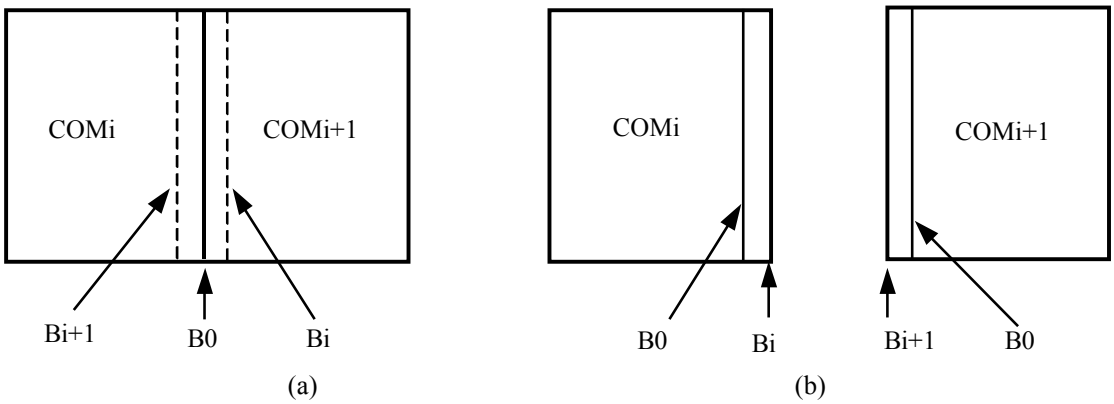

Figure 2. Local boundary between two neighboring sub-domains

technique is required since it allows us to control which machine the client should communicate. In addition, to improve the performance of data transfer between objects, we use an array in a dispatch interface by using a SAFEARAAY with automation data type.

We prepared a Windows NT computer to function as a server for our DCOM objects by setting up Active Configurations and building Proxy-Stub DLL. Two files, DPServer1.exe and DPServer1ps.dll are copied into C:\Winnt\System32\ directory. They are manually started by running the commands: DPServer1 /Service, and regsvr32 DPServer1ps.dll. In fact, servers don't have to be started manually because the DCOM infrastructure supports dynamic object activation. With DCOM, a server doesn't have to listen for client requests the way legacy servers do in a client/server environment because DCOM can dynamically activate services upon client requests. The DCOM Service Manager (SCM) supports remote activation. Since information regarding distributed objects is recorded in the registry, SCM works in conjunction with the system registry to locate and activate distributed objects dynamically upon client requests.

After DCOM objects has been installed and registered on the servers, a client can dynamically activate the services and call methods to perform distributed computation. From our interface (Figure 3), a user can specify the parameters for processing and for controlling DCOM objects. In Figure 3, parameters for our simulation are provided in the Processing Control fields, and the data for DCOM objects are provided in the DCOM Control fields.

For our test, sixteen DCOM objects were developed and registered on four Windows NT machines. For security purpose, administrator privilege is needed to run the application. A user can specify IP address to activate services on each machine. A client can also specify one or two machines to perform computation by specifying only one or two IP in the interface. The services are automatically started and the client can call methods through interface defined in

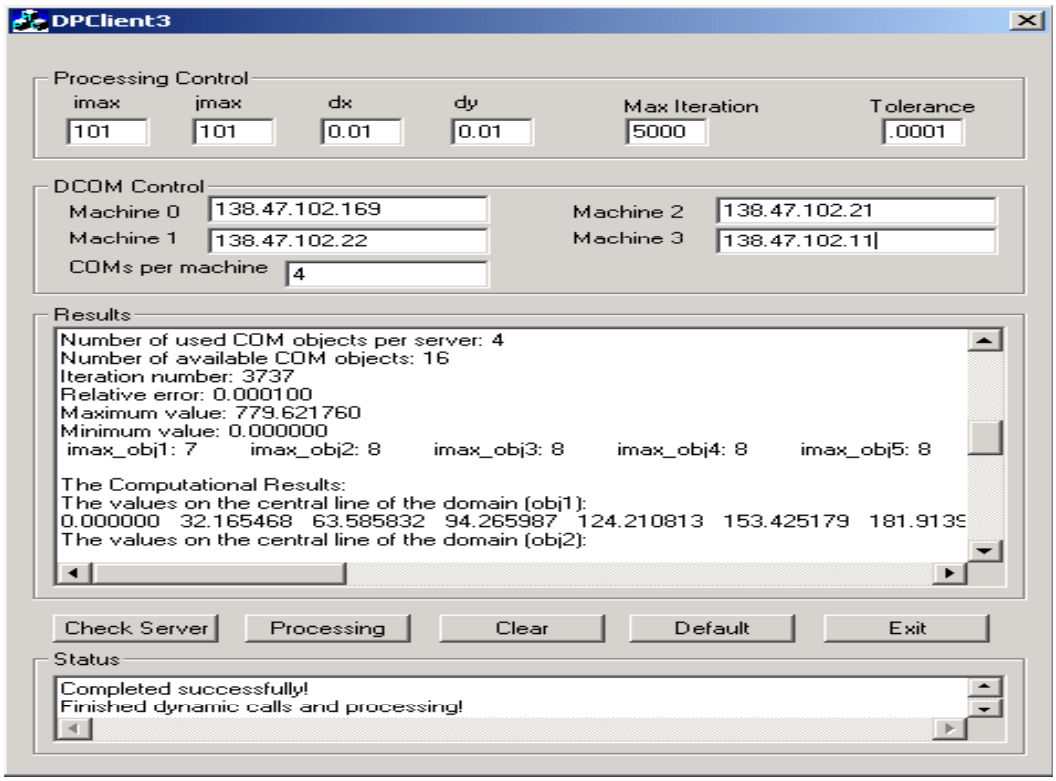

Figure 3. User interface

each DCOM object. By specifying the number of DCOMs per machine we can control the total number of DCOMs used in the distributed computation.

Security allows us to protect private or classified data in a distributed system where objects can be potentially accessed by anyone in cyberspace. DCOM provides a powerful security infrastructure that leverage RPC security. Windows NT provides the NT LAN Manager (NTLM) authentication package, which is an authentication protocol that DCOM supports in Windows NT 4.0. DCOM supports launch access and call-level security. Launch security determines who can activate the server component and thus protects the server machine. Access controls allow the server component to limit user access to its objects, thereby protecting its objects from offenders. In our application development, we don't have to write code to take advantage of DCOM's powerful security infrastructure because Windows NT 4.0 provides the DCOM Configuration (dcomcnfg.exe) tool to allow easy configuration of DCOM security [12].

4. Results and Analysis

The times needed for processing ten testing cases are shown in the Figure 4. In the figure, the top line shows four, eight, and sixteen DCOM objects running on a single machine. It shows that the computational overhead as the number DCOM objects increases the

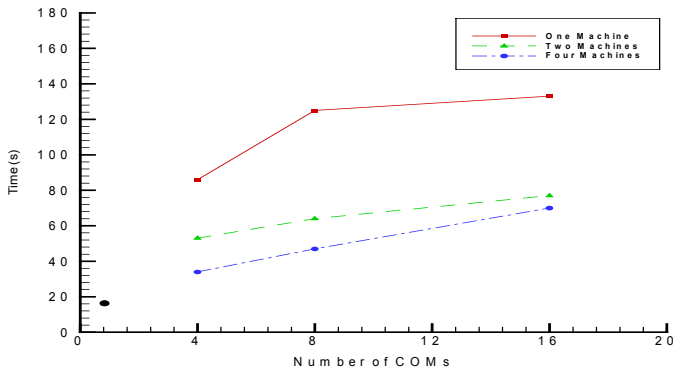

Figure 4. The processing time for ten test cases

processing time also increases. The middle line shows four, eight, and sixteen DCOM objects running on two machines and similarly the lower line shows that on four machines. Theses lines also show that as the number of machines increases the required computational times decreases. However, the dot on the lower left of the chart shows the processing time for one DCOM object running on a simple machine. This indicated that the amount of overhead far exceeds the benefit of distributed processing under these tests. The overhead is due mostly to communications large amount of data between DCOM objects as implied in the top line of the figure. Other work [12] also shown that a remote call in DCOM is five to ten times slower than a single local object call.

A simple analysis may shows that it may not be efficient to use distributed computing for solving small problem. Assuming T_1 is the time needed to solve a problem using one DCOM object running on one machine; T_2 is the time needed to solve the problem using two DCOM objects running on two machines; and T_a is the time for overhead, such as marshaling or unmarshalling, remote invocation, and data communication between two components. For simplicity, we let $T_2 = T_1/2 + T_a$. For small problem, T_1 is small; usually T_a is larger than $T_1/2$; while T_2 is larger than T_1. Thus, it is not efficient to using distributed computing for solving small problem. For large problem, T_1 is large; usually T_a is much smaller than $T_1/2$; while T_2 is smaller than T_1. Thus, there is benefit for using distributed computing for solving large computing problem.

5. Conclusion

DCOM developed by Microsoft is used here to exploit large installation base of networked Windows computers. We developed a distributed DCOM application to simulation a heat transfer

problem. Our application dynamically partitions (and distributes) the problem into a number of components based on the number of available networked computers. Our tests show that large overheads in communications between DCOM objects and between the networked computers making such a distributed application performs slower than one DCOM object running on one computer. However, component-based distributed computing technology makes it flexible to split a task into a number of subtasks and to deploy the components across networked computers, which has potential for use in large scale scientific simulation problems, although care must be taken to reduce communication overheads.

References

[1] Scott M. Lewandowski, "Frameworks for component-based client/server computing," *ACM Computing Surveys (CSUR)*, Volume 30 Issue 1, March 1998.

[2] Jon Hopkins, "Component primer," *Communications of the ACM*, Volume 43 Issue 10, October 2000.

[3] Steve J. Caughey, Daniel Haggmont, and David B. Ingham, "Deploying Distributed Objects on the Internet," *Advances in Distributed Systems: Lecture Notes in Computer Science*, Spriner-Verlag, Vol. 1752, 213-237, 2000.

[4] Valerie Issarny, Luc Bellissard, Michel Riveill, and Apstolos Zarras, "Component-Based Programming of Distributed Applications," *Advances in Distributed Systems: Lecture Notes in Computer Science*, Spriner-Verlag, Vol. 1752, 317-353, 2000.

[5] Wolfgang Emmerich and Neil Roodyn, "Distributed objects," in *Proceedings of the 21st international conference on Software engineering*, May 1999.

[6] Asuman Dogac , Cevdet Dengi , and M. Tamer Öszu, "Distributed object computing platforms," *Communications of the ACM*, Volume 41, Issue 9, September 1998.

[7] Mohamed Fayad and Douglas C. Schmidt, "Object-oriented application frameworks," *Communications of the ACM*, Volume 40 Issue 10, October 1997.

[8] Charles R. Harrell and Donald A. Hicks, "Simulation software component architecture for simulation-based enterprise applications," in *Proceedings of 1998 conference on Winter simulation*, December 1998.

[9] Microsoft, *The Component Object Model Specification*, Version 0.9, Microsoft Corporation, 1995.

[10] W. Dai, R. Nassar, C. Zhang, S. Shabanian, and J. Maxwell, "A numerical model for simulating axisymmetric rod growth in three-dimensional laser chemical vapor deposition," *Numerical Heat Transfer*, Part A, vol. 36, pp. 251-262, 1999.

[11] Thuan L. Thai, *Learning DCOM*, O'Reilly & Associates, Inc. 1999.

[12] Richard Grimes, *Professional DCOM Programming*, Wrox Press Ltd., 1997.