

A Middleware Based Mobile Scientific Computing System—MobileLab

Xi Wang, Xu Liu, Xiaoge Wang, Yu Chen

Department of Computer Science & Technology,
Tsinghua University, Beijing, P.R. China
LX2000@mails.tsinghua.edu.cn

Abstract. In this paper we present a scientific computing system named MobileLab which is designed for pervasive computing on the Internet. A middleware named TOM (stands for Tsinghua Operation Middleware) is designed and employed in MobileLab as a key to access the grid. Based on TOM, MobileLab can access the computational resource transparently; distribute the components to optimize the computation procedure and computation can be moved from one node to another. To use a middleware model in the architecture of scientific computing system is our new approach corresponding to the grid computing and mobile computing.

1. Introduction

Grid computing and the Internet are revolutionizing scientific and technological computing which has gained extremely huge capability from the computational grid. However, there has not been a computing environment for non-expert users to benefit the computational power from the grid, especially on the mobile equipments. There has been large amount of work dealing with networking, data management and power management issues on mobile computing but few focuses on the software architecture of mobile scientific computing. Related work named “Science Pad” [1] was published by Purdue University during the rush of “Pen-Like Computers” in mid 1990s. However it no longer fits the requirements of today’s mobile application. Flexibility and usability is highly required which grid computing requires the computational resources on the internet to be re-coordinated. Thus a middleware model which bridges the mobile device and super computers is designed as a novel feature of our system. The middleware model acts as the “software glue” [2] which enables supercomputers on the Internet work cooperatively to gain a much higher capability than the traditional parallel computer. Although recent technology of embedded system has advanced the computational ability of mobile equipments most of the scientific computing problems can not be solved on these “weak” equipments. These problems often require super computers (cluster) and a long solving period. Unfortunately, scientists who need to run these computational tasks know little about programming on high-performance computers while a lot of computational abilities on super computers were left idle. Mobile Lab is designed for this purpose.

MobileLab is a robust computing environment for high performance computing based on middleware technique. A middleware model named TOM^[7] is designed and

implemented with C++ for the Mobile Lab environment. Distribution of resources, including computation time, is transparent in MobileLab. MobileLab can handle failure gracefully, including network failures, hardware failures and software bugs. Mobile Lab would achieve its best performance when distributing tasks to peers.

2. Architecture of MobileLab

The following picture shows the system architecture of MobileLab:

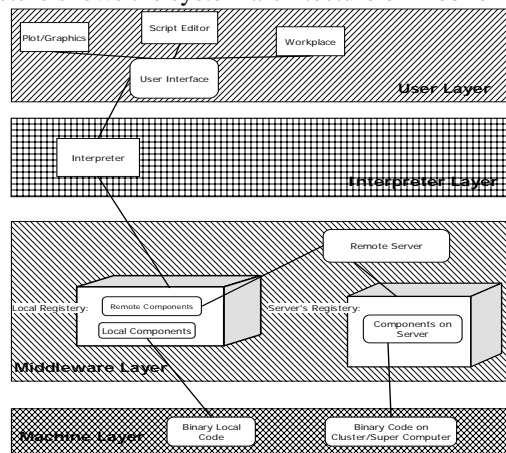


Figure 1: System Architecture of MobileLab

The MobileLab uses a server-client model: The server runs on PC/cluster. When receiving a requirement to create a component it starts a new thread to maintain it.

As the MobileLab system is designed as an integration of computational resources on grid^[5], the server and client must not be closely coupled. Thus when a server is temporarily unavailable the computation can be moved to another node.

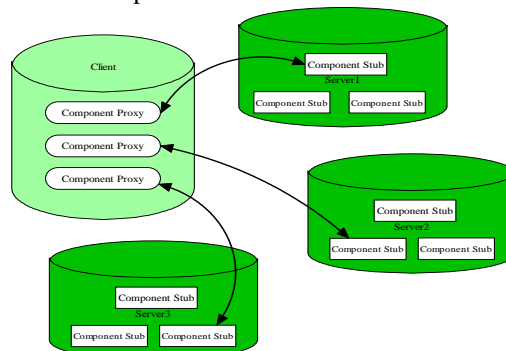


Figure 2: Client-Server relationship of MobileLab

MobileLab has a lightweight client which runs on PC/Laptop/PDA. The client maintains a local registry recording the location of components, both local and remote, for the the interpreter to load the corresponding components when necessary.

There are 3 ways to load a component:

- (1) the component is local : query its location by UUID and load it into process
- (2) the component is remote: query its URL by UUID then we have 2 choices:
download and load it into process or call it remotely.

A client may use several components on several different servers. In whichever way the component is loaded it is transparent to the interpreter, the interpreter need not to where actually the component is running

3. Middle-ware Layer

Most research on distributed and parallel computing is based on an existing and fixed network infrastructure and therefore leads to fairly static solutions such as PVM, MPI, CORBA, RMI, etc. With the emerging ubiquitous computing paradigm, certain significant limits relevant to current middleware are addressed such as: middleware usually tied up with a central naming mechanism.

Middleware are usually built for use of workstations or servers, meaning most current middlewares are heavy and therefore unsuitable for use on lightweight devices. [6]Unlike CORBA or DCOM, our middleware model TOM is designed specifically for MobileLab. Hence, it's a middleware with the feature of moveable and lightweight. TOM is also compatible with Microsoft COM.

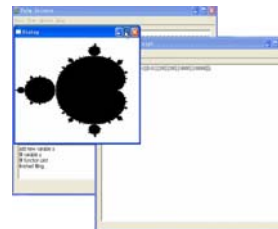
As the component is location transparent, TOM employs a proxy-stub structure. If the component is instantiated inside local process the proxy exposes the actual interface directly; parameters and results are transferred via stack. If the component is remote the proxy marshals the parameters to a consecutive data block and transfers it to the stubs which will un-marshal the parameters and start a new thread. Inside the thread the un-marshaled parameters are passed to the actual components. A call back address is also passed to the thread, when the computation is finished, the thread uses this address to call back and pass back the result. The stub then marshals the result which will be sent back to the proxy and un-marshaled by the proxy [8].

4. Example & Performance Analysis

In the following example a components which generates the Mandelbrot-set is employed. We also use this example to compete the efficiency between local (serial) and remote (parallel) components.

```
a=mandelbrot ( [[-1.4] [-1] [.01] [200] [200] [10000] [100000]] );  
plot(a) ;
```

The script contains only 2 lines but it covers all. The first line requires a component named "mandelbrot" to calculate the mandelbrot set. When the interpreter reads the first line, it searches the local registry for a component named "mandelbrot" and tries to instantiate it. Then it inquires its "I_Complexity" interface to estimate approximately how much time it will cost. If the complexity is acceptable then do the computation locally, otherwise the interpreter searches another remote component with the same name "mandelbrot" on the parallel computer and try



to compute remotely. When the computation is finished the user interface runs a “plot” function to show the result. All these procedures are done automatically.

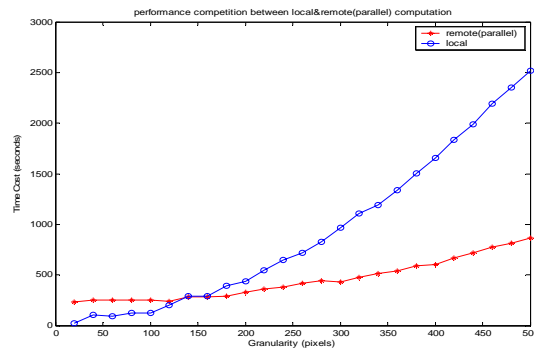


Figure 4: Performance Comparison between Local and Remote Computation

Client: AMD Duron 800, 256 Mb memories, Windows XP

Server: 4-nodes cluster with Intel P4 1.4 on every node, 1 GB memories, Red-hat Linux 9.0

From this comparison result we can see that when the pixel number is small, the computation load is light, the local computation is faster than remote computation, because the network overhead costs more time than the computation itself^[10]. While the pixel number is growing, the advantage of remote computing is distinctive since it costs much less time. It is especially superior when the client is on a PDA-like system.

References

1. A. Joshi, S. Weerawarana, T.T. Drashansky and E.N. Houstis, SciencePad: An intelligent electronic notepad for ubiquitous scientific computing, Proc. Int. Conf. on Intelligent Information Management Systems (June 1995) pp. 107-110.
2. <http://www.gnu.org/software/glue/glue.html>
3. A. Athan and D. Duchamp, Agent-mediated message passing for constrained environments, Proc. Mobile and Location-Independent Computing, USENIX, Cambridge, MA (August 1993) pp. 103-107.
4. A. Joshi, S. Weerawarana, R.A. Weerasinghe, T.T. Drashansky, N. Ramakrishnan and E.N. Houstis, A survey of mobile computing technologies and applications, Tech. Report TR-95-050, Dept. of Computer Sciences, Purdue University (1995).
5. Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, the Physiology of the Grid an Open Grid Services Architecture for Distributed Systems Integration, Feb. 2002. <http://www.globus.org>
6. Luc Hogue, Christian Hutter, Middleware Support for Ubiquitous Computing, Contents ERCIM News No. 54 SPECIAL THEME: Applications and Service Platforms for the Mobile User, July 2003
7. <http://hpclab.cs.tsinghua.edu.cn/~liuxu/mobilelab/doc/tom/>
8. Guy/Henry Eddon, Inside DCOM, Microsoft Press, 1997
9. J. Xu, B. Randell and A. F. Zorzo, "Implementing Software Fault Tolerance in C++ and Open C++: An Object-Oriented and Reflective Approach", in Proc. Int. Workshop on Computer-Aided Design, Test, and Evaluation for Dependability (CADTED'96), (Beijing, China), pp.224-9, International Academic Publishing, 1996.
10. Ian Foster, Designing and Building Parallel Programs, Chapter 8.7, 1995-1999 DARPA Intrusion Detection Evaluation Dataset. <http://www.ll.mit.edu/IST/ideval/index.html>