netCFD: a Ninf CFD component for Global Computing, and its Java applet GUI

Mitsuhisa Sato, Kazuhiro Kusano,
Real World Computing Partnership, Tsukuba, Japan
{msato,kusano}@trc.rwcp.or.jp

Hidemoto Nakada, Satoshi Sekiguchi
Electrotechnical Laboratory, Tsukuba, Japan

Satoshi Matsuoka
Tokyo Institute of Technology, Tokyo, Japan

Abstract

Ninf is a middleware for building a global computing system in wide area network environments. We designed and implemented a Ninf computational component, netCFD for CFD (Computational Fluid Dynamics). The Ninf Remote Procedure Call (RPC) provides an interface to a parallel CFD program running on any high performance platforms. The netCFD turns high performance platforms such as supercomputers and clusters into valuable components for use in global computing. Our experiment shows that the overhead of a remote netCFD computation for a typical application was about 10% comparing with its conventional local execution. The netCFD applet GUI which is loaded in a web browser allows a remote user to control and visualize the CFD computation results interactively.

1 Introduction

Remarkable growth of computer network technology has resulted in a variety of information services being accessible through the Internet. Most existing global network services, such as e-mail, file archives, and the WWW, are limited to the mere sharing of data resources. The global network could be far better utilized, as it embodies a potential to provide a computational environment for sharing of computational resources, including CPUs and disk storage. The coming gigabit information superhighways will further enhance the vision of world-wide global computing resources,as it will be more possible to tap into power of an enormous number of computers with idle computation cycles. For the global network computing, libraries and systems such as Globus[3], Legion[4], NetSolve[2] and other systems[1] have been proposed.

We are currently pursuing the Ninf (Network based Information Library for high performance computing)[6] project, as an infrastructure for world-wide global computing in scientific computation. Ninf is a Japanese global computing infrastructure project which involves the Real World Computing Partnership(RWCP), the Electrotechnical Laboratory (ETL), the Tokyo Institute of Technology, and several other collaborators. Ninf is intended to provide a useful interface to a variety of high performance computing platforms such as clusters, MPP's and supercomputers, for global scientific computing which uses computational resources that are distributed across a world-wide global network. The basic Ninf system supports the client-server based computing. The computational resources are made available as remote libraries on remote computation hosts which can be called over the global network by a programmer's client program written in an existing language such as Fortran, C, or C++ by using a Ninf remote procedure call (RPC). The programmer can build a global computing systems by using Ninf remote libraries as components, without being aware of the complexities and hassles of network programming. Since the Ninf RPC can also be asynchronous, a user can issue multiple calls in parallel to exploit network-wide parallelism.

In this paper, we describe a Ninf computational component, netCFD, which allows remote computational resources such as powerful workstations and parallel computers to be used as components for CFD (Computational Fluid Dynamics) applications. NetCFD is a set of Ninf interfaces for a CFD program which can be used via Ninf RPC from a remote client. We designed a Java applet to visualize the result of computation by netCFD as a sample netCFD application, By implementing the GUI client as a Java applet, a user can use the netCFD application from anywhere via a web browser without installing any programs.

The next section overviews the Ninf as a background. In Section 3, we describe netCFD, and we present some preliminary results in Section 4. Section 5 presents some concluding remarks.

2 Ninf: a Network based Information Library

2.1 Overview

The basic Ninf system employs a client-server model. The server and client may be connected via a local area network or over the Internet. Machines may be heterogeneous: data in communication is translated into the common network data format.

A Ninf server process runs on a Ninf server host. Ninf remote libraries are implemented as executable programs which contain network *stub* routine as their main routines, and are managed by the server process. We call such executable programs *Ninf executables (programs)*. When the library is called by a client program, the Ninf server searches the Ninf executables associated with the library's name, and executes the matched executable, and sets up an appropriate communication with the client. The stub routine handles communication with the Ninf server and its client, including argument marshaling. The underlying executable can be written in any existing scientific language, such as Fortran or C, as long as it can be executed in the host.

Ninf delivers the following benefits:

- A client can execute the most time-consuming part of his program on multiple remote high-performance computers, such as vector supercomputers and MPP's, without any requirement for special hardware or operating systems. If such supercomputers are reachable via a high speed network, the application will naturally run considerably faster. Ninf also provides uniform access to a variety of supercomputers.
- The Ninf programming interface is designed to be extremely easy-to-use and familiar-looking for programmers of existing languages such as FORTRAN, C and C++. The user can call remote libraries without any knowledge of network programming, and easily convert existing applications that already use popular numerical libraries such as LAPACK.
- The Ninf RPC can be asynchronous and automatic: for parallel applications, a group of *Ninf metaservers* [5] maintains the information of Ninf servers in the network in a distributed manner, and automatically allocates remote library calls dynamically to appropriate servers for load balancing. Ninf provides a transaction system to allocate multiple calls to achieve network-wide parallelism. The data-dependencies among Ninf calls are automatically detected and scheduled by the metaserver. The Ninf metaserver could be regarded as a network agent which locates an appropriate server depending on the client request and state of network resources.

2.2 The Programming Interface

`Ninf_call()` is the sole client interface to the Ninf servers. As an example to illustrate the programming interface, let us consider a simple matrix multiplication routine in C programs with the following interface:

```
/* declaration */
double A[N][N],B[N][N],C[N][N];
....
/* calls matrix multiply, C = A * B */
dmmul(N,A,B,C);
```

When the `dmmul` routine is available on a Ninf server, a client program can call the remote library using `Ninf_call`:

```
/* call remote Ninf library */
Ninf_call("dmmul",N,A,B,C);
```

Here, `dmmul` is the name of a library that is registered as a Ninf executable on a server, and `A`,`B`,`C` and `N` are the same arguments, declared above. As we see here, at the client, the user only needs to specify the name of the function as if he were making a local function call: `Ninf_call()` automatically determines the function arity and types of arguments, appropriately marshals the arguments, makes the remote call to the server, obtains the result, places the result in the appropriate argument, and returns to the client. The Ninf RPC is designed to make it appear as if arguments are being shared between the client and the server. Note that the physical location of the Ninf server is specified in an environment variable, or a setup file.

To realize such a simple programming interface for the client, we designed Ninf RPC so that client function calls obtain all the interface information regarding the called library function at runtime from the server. As shown in Figure 1, a client function call requests the interface information of the calling function to the Ninf server, which in turn returns the registered Ninf executable interface information to the client. Using this information, Ninf RPC automatically performs argument marshaling, and generates the sequence of sending and receiving data to/from the Ninf server. The client library then interprets and marshals the arguments on the stack according to the supplied information. For variable-sized array arguments, the interface description must specify an expression that includes the input scalar arguments so that the size of the arrays can be computed. In the above example, the client function call sends the input arrays, `A` and `B`, both of which are of a size given by the parameter `N`. The Ninf server invokes the Ninf executable of library `dmmul`, and forwards the input data to it.

This design is in contrast to traditional RPCs, where stubs are generated on the client side at compile time. The Ninf RPC's dynamic interface acquisition eliminates the need for such compile-time activities at all, and thus relieves the user

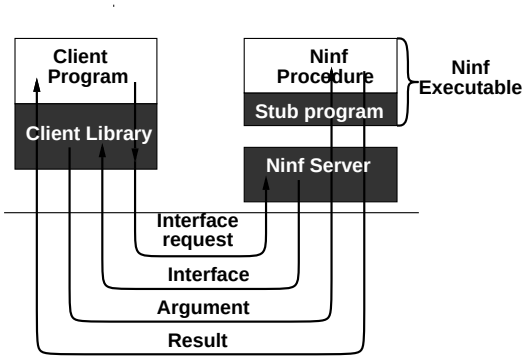

Figure 1. Ninf RPC

from related code maintenance. Although this will cost an extra network round trip time, we judged that typical scientific applications are both compute and data intensive such that the overhead should be not significant.

The Ninf RPC may also be invoked asynchronously, to exploit network-wide parallelism. It is possible to issue a request to a Ninf server, continue with other computation, then poll for the result later. Multiple RPC requests to different servers are also possible. Asynchronous Ninf RPC is thus an important feature for parallel programming.

Since the Ninf client programming interface has been designed to be as language independent as possible, a Ninf client may be written in any one of a variety of programming languages. It is usually easy to design a client interface to Ninf, so long as the language supports a standard external function interface for C programs. We have already designed and implemented client `Ninf_call` functions for C, FORTRAN, Java, and Lisp. The client and remote library can be written in totally different languages.

2.3 Ninf IDL

A *library provider* provides the numerical library and computational resources to the network by preparing Ninf executables in the Ninf IDL (Interface Description Language). For example, the interface description for the matrix multiplication given in the previous section is:

```
Define dmmul(IN int n, IN double A[n][n],
   IN double B[n][n], OUT double C[n][n])
"... description ..."
/* library including this routine. */
Required "libxxx.o"
/* Use C calling convention. */
Calls "C" dmmul(n,A,B,C);
```

where the *access specifiers*, `IN` and `OUT`, specify whether the argument is read from or written to. To specify the size of the arguments, other `IN` arguments can be used to give a size expression. In this example, the value of `n` is referenced to calculate the size of the array arguments `A`, `B`, `C`. Since Ninf is designed for numerical applications, the data types supported in the Ninf IDL is tailored for such purposes: for example, data types are limited to scalar types and their multi-dimensional array types. The interface description is compiled by a *Ninf interface generator* to generate a stub program for each library function described in its interface information. The interface generator also automatically outputs a makefile with which Ninf executables can be created by linking stub programs and library functions. Once the Ninf executables are "registered"on the server, anyone in the network can use the libraries in a transparent manner by issuing Ninf RPCs. Some existing libraries, such as LAPACK, have already been 'Ninfied' in this manner.

3 netCFD: a Ninf CFD component

NetCFD is a Ninf global computing component for CFD (Computational Fluid Dynamics) computation, and it provides interfaces to CFD programs that run on MPP's and clusters or single workstations via Ninf RPCs. NetCFD turns high performance platforms such as supercomputers, MPPs and clusters into valuable components for global computing.

3.1 netCFD Ninf Interface

In the current version of netCFD, the interface is designed for our CFD program called femFlow which simulates a given system using the Finite Element Method (FEM).

The main function in netCFD is 'femFlow' which starts execution of the CFD program. The following IDL fragment describes the interface to 'femFlow' in the Ninf IDL:

```
Define femFlow(IN int id, IN float header[24],
 IN int n_node,IN int n_elem,
 IN float x[n_node],IN float y[n_node],
 IN float z[n_node], /* coordinate */
 IN float u[n_node],IN float v[n_node],
 IN float w[n_node], /* velocity */
     /* pressure and temperature */
 IN float t[n_node],IN float p[n_node],
     /* initial conditions */
 IN char uf[n_node],IN char vf[n_node],
 IN char wf[n_node],
 IN char tf[n_node],IN char pf[n_node],
     /* define finite elements */
 IN short elems[n_elem*8],IN int npe,
 IN char  pe[n_elem], /* partitioning */
 flowDisplay(IN int step[1],IN double ctime[1],
        IN double delta[1],
        IN float data[n_node*5])){
   ... call CFD program ...
}
```

The `IN` arguments specify arrays of node coordinates and the initial conditions of finite elements at the nodes. The finite elements are specified by the array `elems` which stores the connectivity information with their node indexes. The arguments `pe` contains the portioning information of nodes in case of parallel computation. The argument `header` contains several parameters for a simulation. The IN arguments are sent to the 'femFlow' Ninf executable via the Ninf server. The Ninf executable then calls the main CFD program. The last argument `flowDisplay` is the call back function on a client side that receives the result in the form of the velocity, pressure and temperature at each of the mesh node at each simulation time step.

Other interfaces are provided to restart suspended simulations and to replay the last few steps of a simulation. Some sample mesh data files are also stored in the Ninf server, and may be retrieved via Ninf RPCs.

3.2 femFlow: a CFD program with FEM

The CFD program that underlies netCFD is a parallel 3D CFD program, femFlow, written in C++ and MPI. In a parallel environment, the Ninf executable in the server forks the parallel process within the parallel platform, and communicates with them. In a single processor environment, the sequential version is directly linked to the stub program to call the routine.

The used modeling and numerical methods are as follows:

- Basic Equations
  - Equation of Continuity
  - Equation of Motion (Navier-Stokes Equation)
  - Equation of Mechanical Energy
- Discretization Method
  - Finite Element Method (FEM)
  - 8-node isoparametric element
  - High-order upwind finite element scheme
- Matrix Solver
  - SCG (Scaled Conjugate Gradient) Method
- Time integration
  - SMAC Method
- Parallelized by domain decomposition
- Object-oriented design in C++

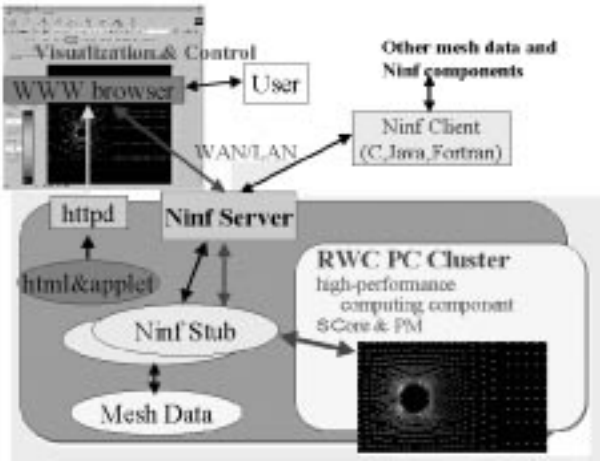

Figure 2. netCFD and Java applet GUI

3.3 netCFD Java applet GUI

The netCFD Java applet GUI is a Java applet which allows a remote user to interactively control CFD computation and visualize results in a web browser. It uses a Java Ninf RPC interface to retrieve the mesh data and run the simulation in the Ninf server.

The client applet receives the data of velocity and pressure, temperature at each time step by "call back" from the server. In the netCFD applet GUI, the call back function is a Java method function to visualize the data. There are a few modes of displaying the result: In the vector mode, a vector at each node indicates the direction of flow. Its color shows the value of the selected data item which may be including velocity, pressure or temperature, at each node. In the element mode, each element is painted using the color of the value of the selected item at its center. The user can zoom into a part of the simulated area to see a result more precisely during the simulation.

Figure 2 illustrates the netCFD system, which was demonstrated at Supercomputing '98. The netCFD Java applet GUI is one of netCFD applications, and the netCFD Ninf component can be used as a component in other global computing systems.

4 Preliminary Performance Evaluation

This section shows the result of a preliminary evaluation of netCFD. To demonstrate the feasibility of global computing using netCFD, we have built a global computing environment to link the RWCP Tsukuba Research Center (TRC) and the Electrotechnical Laboratory(ETL), as shown in Figure 3. We set up a Ninf server on a Sun Ultra Enterprise (4 CPU, 300 MHz Ultra Sparc) at the TRC, and the

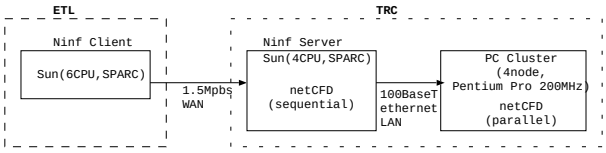

Figure 3. Experiment Test-bed between TRC and ETL

sequential version of the netCFD component runs on this server machine. The parallel version of the netCFD component runs on a four-node PC cluster (Pentium Pro 200 MHz), connected with a 100BaseT Ethernet local area network. The client is a Sun Ultra Enterprise (6 CPU, 336 MHz Ultra Sparc) at the ETL. These sites are connected over a 1.5 Mbps wide-area network.

4.1 Basic Performance of Ninf RPC

To measure the basic performance of the Ninf RPC, we prepared the following Ninf RPC entry:

```
Define test(IN int in_size,
   IN int input[in_size],
   IN int out_size,
   OUT int output[out_size])
{ /* do nothing */  }
```

This Ninf entry sends an input array of size `in_size` from the client to the server and returns an output array of size `out_size` back from the server to the client. Figure 4 shows the basic performance of Ninf call from both a local client (at TRC) and a remote client at ETL. The graph of 'up' shows the execution time taken for a call of this test entry for varied values of the size `out_size`, with `in_size` fixed at one. The graph 'down' shows the execution times for varied values of the size `in_size`, with `out_size` fixed at one. This graph indicates that the overhead is 0.05 sec, with a bandwidth of 0.2 KBytes from the ETL site.

4.2 Performance of a netCFD application

We measured the performance of a typical netCFD client application, which computes the velocity, temperature and pressure of each finite elements and stores them in a local file in client site at every time step, as shown in Figure 5. The netCFD executable can execute either sequentially on the Sun workstation or in parallel on our PC cluster.

Figure 6 shows the execution time of the application for both local (TRC) and remote (ETL) clients. We made the mesh data around a cylinder with two sizes: 'Small' (2720

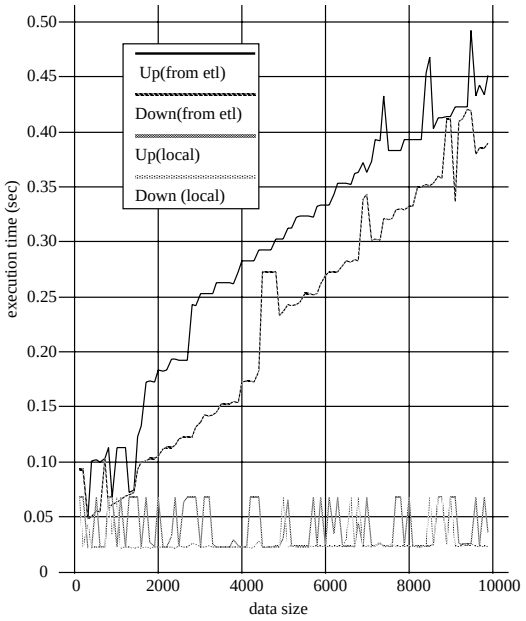

Figure 4. Basic performance of Ninf Call

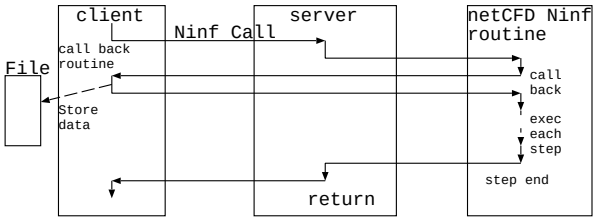

Figure 5. A typical netCFD application

nodes, 1289 elements) and 'Large' (10560 nodes, 5120 elements) for this experiment. Each program runs for 200 steps. In the netCFD Ninf executable, the netCFD routine executing in server side calls back the specified routine on the client at every step, so that the call back routine on the client stores data sent back in a file. The amount of data sent back to the client at each step is 106 KB in 'Small', 412 KB in 'Large' respectively. For comparison, 'local' indicates the same application program running without netCFD. As can be seen in the Figure, the overhead for remote execution of netCFD is less than 10% of the total execution time. For the PC cluster, execution time for a remote client takes almost the same amount of time as execution of a local client. This is because the data sent from the parallel CFD program is buffered in the sever, and the sending of data to the remote client can thus overlap with computation. The call back from server side are waiting for returning from the call back. We

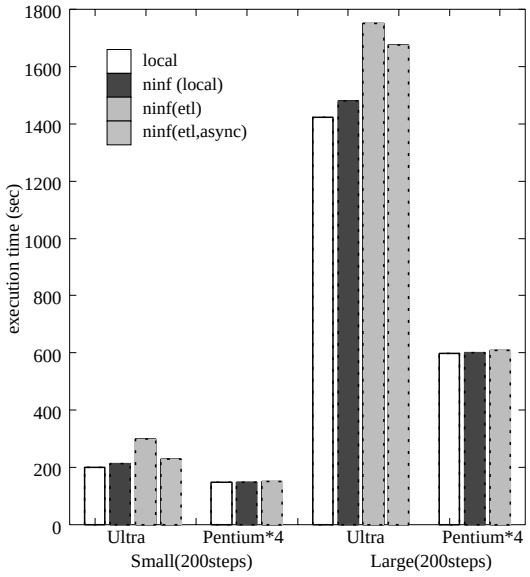

Figure 6. netCFD performance

implemented an asynchronous call back mode in which the server does not wait for the return. 'ninf(etl,async)' shows the execution time using the asynchronous call back mode. It appear to improve the performance for this application.

4.3 Other issues for remote computing system

One of the most important issues for global computing systems is authentication. A simple authentication mechanism is included in the current implementation of netCFD so that the library provider can restrict use of his computing resources to a particular set of users.

The Ninf metasever provides a scheduling mechanism which automatically selects an appropriate server to execute the requested Ninf calls. This will be useful when several computational resources are available for, for example, netCFD applications. When many Ninf applications are running, the metaserver locates the least loaded server and forwards the Ninf call there. We did not use the metaserver in our experiments.

The Java applet is a very useful mechanism because it makes Ninf applications available via web servers anywhere, without the need for a program to reside on the remote client. There are, however, many restrictions on using applets because of security reason. The netCFD Java applet cannot upload a remote user's data, or store results. While the applet GUI is useful for demonstration, the netCFD program should be executed as a Java application if the remote user wants to upload and store data.

5 Concluding Remarks

In this paper, we have presented a Ninf computational component for CFD, and some applications of netCFD with result of a preliminary evaluation. Ninf enables a remote user to use several computational resources for CFD computation in a global computing environment with a single interface. Our experimental results show the possibilities of global computing for CFD applications. The netCFD can also be used as a component for other applications, such as parallel domain decomposition method and heterogeneous computation.

As another demonstration of a Ninf component, we have also designed netMO for the molecular orbital (MO) computations in chemistry. NetMO allows remote users to access to a famous MO program, GAMESS, via a Ninf interface. It provides a variety of functionality given by GAMESS, as well as fast computation of MO.

Some demonstrations of netCFD and netMO are currently available on our web site at:

`http://pdplab.trc.rwcp.or.jp/`

References

[1] Notes of the 1st Intl. Workshop on 'Desktop Access to Remote Resources'. http://www-fp.mcs.anl.gov/ gregor/datorr.

[2] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. Technical report, University of Tennessee, 1996.

[3] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. In *Proc. of Workshop on Environments and Tools*. SIAM, 1996. http://www.globus.org/.

[4] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr. Legion: The Next Logical Step Toward a Nationalwide Virtual Computer. Technical report, University of Virginia, 1994.

[5] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi. Utilizing the Metaserver Architecture in the Ninf Global Computing System. In *Proc of HPCN'98 (LNCS 1401)*, 1998.

[6] M. Sato, H. Nakada S. Sekiguchi, , S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A Network based Information Library for Global World-Wide Computing Infrastructure. *Proc. of HPCN'97 (LNCS 1225)*, pages 491–502, 1997.