World-Wide Computing Infrastructure: Global and Local Partnership

Satoshi Sekiguchi
Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba,
Ibaraki 305, Japan
sekiguchi@etl.go.jp

Mitsuhisa Sato
Real World Computing Partnership
Takezono 1-6-1, Tsukuba,
Ibaraki 305, Japan
msato@trc.rwcp.or.jp

Abstract

The world-wide computing infrastructure on the growing computer network technology is a leading technology to make a variety of information services accessible through the Internet for every users from the high-end high performance computing users through many of personal computing users. The important feature of such services is location transparency; information can be obtained irrespective of time or location in virtually shared manner. In this article, we introduce Ninf, an ongoing global network-wide computing infrastructure project which allows users to access computational resources including hardware, software and scientific data distributed across a wide area network. Preliminary performance result on measuring software and network overhead is shown that promises the future reality of world-wide network computing.

1. Introduction

Remarkable growth of computer network technology has spurned a variety of information services accessible through the Internet. The important feature of such services is location transparency; information can be obtained irrespective of time or location in a virtually shared manner. However, most existing global network services such as e-mail, file archives, and the WWW, are limited to merely sharing data resources. The global network could be far better utilized, embodying the potential to provide a computational environment to share computational resources including CPUs and disk storage. The coming of gigabit information super-highway will further enhance the vision of world-wide global computing resources, being able to tap into powers of enormous numbers of computers with idle computation cycles.

From the high performance computing perspective, the challenging application programs require the ability to exploit diverse, physically distributed resources, such as access to supercomputers, large databases, storage devices, visualization devices, and sensing devices as well. These applications are not able to run using resources within a single computer center, thus the wide variety use of the meta-computing system with heterogeneous and dynamic nature is becoming much more important being built on the world-wide computing infrastructure.

The network infrastructure makes possible the realization of high performance world-wide global computing. A few systems are proposed for such computing system: Legion [5] is a project for constructing the nationwide virtual computer comprised of many supercomputers and workstations. Legion uses an object-oriented language, Mentat [4], to program global distributed computing systems. NetSolve [2] provides a easy-to-use programming interface similar to ours, but it has no description language for interface information. An agent process is provided for load balancing in the system. The Remote Computation System (RCS) [1] is a remote procedure call facility that provides uniform access to remote computers ranging from high-end workstations to supercomputers. In order to minimize response time, the RCS allocates the solution of a problem dynamically to the least loaded machine in the network that provides the desired service. When using the system for parallel applications, the dynamic task allocation leads to a balanced load among the involved processors. The prototype system is implemented on top of PVM. The Globus [3] is a toolkit which intended to achieve a vertically integrated treatment of application, middle-ware, and network. The Globus project provide basic infrastructure that can be used to build portable, high-performance range of world-wide computing services.

As an infrastructure for world-wide global computing in scientific computation, we are currently pursuing the *Ninf (Network based Information Library for high performance computing)* project [7][1]. Our goal is to provide a platform for global scientific computing with computational resources distributed in a world-wide global network. The paper describes the motivation of Ninf, its components, and the underlying technologies that support such global computing.

2. Ninf Overview

2.1. Ninf Design Concept

The basic Ninf system supports client-server based computing. The computational resources are available as remote libraries at a remote computation host which can be called through the global network from a programmer's client program written in existing languages such as Fortran, C, or C++. The parameters, including large arrays, are efficiently marshalled and sent to the *Ninf server* on a remote host, which in turn executes the requested libraries, and sends back the result. The *Ninf remote procedure call (RPC)* is designed to provide programming interface which will be very familiar to the programmers of existing languages. The programmer can build a global computing systems by using the Ninf remote libraries as its components, without being aware of complexities and hassles of network programming.

The benefits of Ninf are as follows:

- A client can execute the most time-consuming part of his program in multiple, remote high-performance computers, such as vector supercomputers and MPPs, 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. It 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 the remote libraries without any knowledge of the network programming, and easily convert his existing applications that already use popular numerical libraries such as LAPACK.

- The Ninf RPC can also be asynchronous and automatic: for parallel applications, a group of *Ninf metaservers* maintains the information of Ninf servers in the network in a distributed manner, and automatically allocates remote library calls dynamically on appropriate servers for load balancing. To allocate multiple calls to achieve network-wide parallelism, Ninf provides a transaction system, where the data-dependencies among the 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 client request and status of network resources.

- The Ninf network database server provides query on accurate constant database needed in scientific computation, such as important constants of physics and chemistry. By doing so, the user is freed from the burdens and mistakes of inputting accurate constant data from printed charts. Furthermore, we plan to develop a framework where data values of reports on experiments by researchers could be referenced directly over a network in the midst of user's computation.

While Ninf can serve to speed-up the applications by exploiting the power of remote supercomputer servers and network parallelism, it is a convenient tool for obtaining the answer of a particular numerical function and values quickly. We have designed a WWW interface, called *NinfCalc* as a front-end so that Ninf libraries registered in computation servers could be cataloged, browsed, and tried out through Web browsers.

From the user's perspective, Ninf offers yet another way to share resources over a global network. On the other hand, there are already various network infrastructures and tools already in use, and one might ask would another infrastructure be needed; for example, one could claim the extreme that, one could use existing file transfer services such as ftp to remotely obtain numerical libraries, compile on his local machine, and execute the program there. Aside from the issue of having a supercomputer locally on hand, there are many other advantages to Ninf in this case — security and proprietary issues naturally are obvious topics; there are other practical issues of expending the efforts of fetching, compiling, and maintaining code in heterogeneous computing environments. Ninf allows reduction of maintenance costs by concentrating the efforts of high-quality, well-maintained libraries on compute

[1] The URL of Ninf project is `http://phase.etl.go.jp/Ninf/`. The nickname "Ninf" comes from a similar pronounce word "Nymph", which we expect it, living in a network world, to make the dream of quickly getting information resource come true.

servers and not on each machine. Thus, even for slower networks where there are no speed advantages to be gained, Ninf is still beneficial in this regard.

2.2. The Ninf System Architecture

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

A Ninf server process runs on the Ninf computation server host. The Ninf remote libraries are implemented as executable programs which contain network *stub* routine as its main routine, and 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 its name, and executes the found executable, setting up an appropriate communication with the client. The stub routine handles the communication to the Ninf server and its client, including argument marshaling. The underlying executable can be written in any existing scientific languages such as Fortran, C, etc., as long as it can be executed in the host.

A *library provider*, who provides the numerical library and computational resource to the network at large, prepares the Ninf executables by (1) writing the necessary interface description of each library function in *Ninf IDL (Interface Description Language)*, (2) running the Ninf IDL compiler, which emits the necessary header files and stub code, (3) Compiling the library with the compiler for the programming language the library is written in, and, (4) linking with the Ninf RPC libraries, finally, (5) *registering* them with the Ninf server running on his host. After these steps, anyone in the network can use the libraries by the Ninf RPC in a transparent. Some existing libraries, such as LAPACK, have already been 'Ninfied' in this manner.

In the current implementation, the communication between a client and the server is achieved by means of standard TCP/IP connection to ensure reliable communication between processes. In an heterogeneous environment, Ninf uses the Sun XDR data format as a default protocol. Also, clients can also specify call back functions on the client side for various purposes, such as interactive data visualization, I/O of data, etc.

2.3. The Programming Interface

`Ninf_call()` is the sole client interface to the Ninf compute and database servers. In order to illustrate the programming interface with an example, let us consider a simple matrix multiply routine in C programs with the following interface:

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

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

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

Here, `dmmul` is the name of library registered as a Ninf executable on a server, and `A`,`B`,`C`,`N` are the same arguments. As we see here, the client 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 the type of each argument, appropriately marshals the arguments, makes the remote call to the server, obtains the results, places the results in the appropriate argument, and returns to the client. In this way, the Ninf RPC is designed to give the users an illusion that arguments are 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 simplicity in the client programming interface, we designed Ninf RPC so that client function call obtains all the interface information regarding the called library function at runtime from the server. 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 small enough. The interface information includes the number of parameters, these types and sizes and access modes of arguments (read or write). Using these informations, Ninf RPC automatically performs argument marshaling, and generates the sequence of sending and receiving data from/to the Ninf server.

As shown in Figure 1, the 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. The client library then interprets and marshals the arguments on the stack according to the supplied information. For variable-sized array arguments, the IDL must specify an expression that includes the input scalar arguments whereby the size of the arrays can be computed. This design is in contrast to traditional RPCs, where stub generation is done on the client side at compile time. As a result of dynamic interface acquisition, Ninf RPC does not require such compile-time activities at all, relieving the users from any code maintenance.

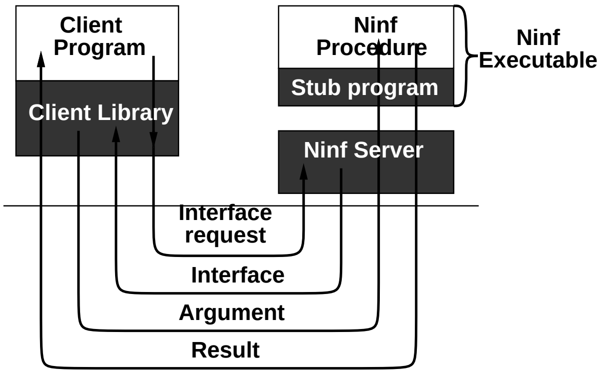

Figure 1. Ninf RPC

In the above example, the client function call sends the input arrays, `A` and `B`, whose size is computed by the parameter `N`. The Ninf server invokes the Ninf executable of `dmmul` library, and forwards the input data to it. When the computation is done, the result is sent back to the client through the server. The client function call stores the returned data at the location pointed by the argument, `C`.

The Ninf RPC may also be invoked asynchronously to exploit network-wide parallelism. It is possible to issue the request to a Ninf server, continue with the other computation, and poll for the request later. Multiple RPC requests to different servers are also possible. For this reason, the asynchronous Ninf RPC is an important feature for parallel programming, as we describe in Section 3.

Ninf RPC also supports a call-back functional argument to communicate with the client during executing the RPC in a server. A Ninf library routine can take the functional argument to call the user-supplied routine from the Ninf executable. Consider a scientific simulation as an application of the Ninf. A typical simulation program initializes the state, and updates the state to display or records it at every certain time steps. The callback argument is useful to call the user-supplied routines to send the data at each time-step while keeping the internal state in the sever.

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

The interface information may also be used on the client side at compile time. The remote user can get the interface information from the server to generate appropriate client stub routine and its prototype header files for its library which is registered there. In this case, the program will be entirely typechecked, improving code reliability. Also, dynamic interface information acquisition will be skipped as mentioned earlier, improving performance slightly. On the other hand, instead of using only `Ninf_call()`, the user must call the client stub routine that will be created for each function available at the server.

2.4. Ninf IDL (Interface Description Language)

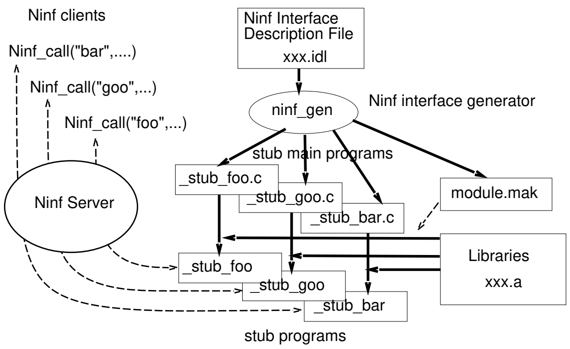

Figure 2. Ninf interface generator

The Ninf library provider describes the interface of the library function in Ninf IDL to register his library function into the Ninf server. Since the Ninf system is designed for numerical applications, the supported data type in Ninf is tailored for such a purpose; for example, the data types are limited to scalars and their multi-dimensional arrays. On the other hand, there are special provisions in the IDL for numerical applications, such as support for expressions involving input arguments to compute array size, designation of temporary array arguments that need to be allocated on the server side but not transferred, etc.

For example, the interface description for the matrix multiply given in the previous section is:

```
Define dmmul(long mode_in int n,
 mode_in double A[n][n],
 mode_in double B[n][n],
 mode_out double C[n][n])
"... description ..."
Required "libxxx.o"
/* specify library required
            by this routine. */
```

```
Calls "C" dmmul(n,A,B,C);
/* Use C calling convention. */
```

where the *access specifiers*, `mode_in` and `mode_out`, specify whether the argument is read or written. To specify the size of each argument, the other `in_mode` arguments can be used to form a size expression. In this example, the value of `n` is referenced to calculate the size of the array arguments `A`, `B`, `C`.

In addition to the interface definition of the library function, the IDL description contains the informations needed to compile and link the libraries.

As illustrated in Figure 2, the interface description is compiled by the *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 the Ninf executables can be created by linking the stub programs and library functions.

The interface description can also include the textual information, which is used to automatically create manual pages for the library functions. The remote users can browse the available library functions in the server, its interface, and the other information generated by the interface generator through a Web browser interface.

2.5. Preliminary Performance

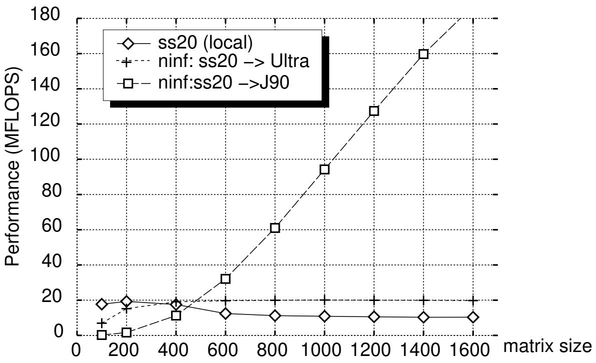

Figure 3. Performance of Linpack via Ninf RPC

Figure 3 shows preliminary performance of Linpack benchmark with the Ninf. We use a SparcStation-20 (60 MHz SuperSPARC, 128MB memory) as a client. The Ninf server runs at JCC Ultra Sparc JU1/140 (143 MHz UltraSPARC, 96 MB memory) connected via 100 base-TX ether switch, and at Cray J90 (4CPU) connected via FDDI network (Figure 4). Note that the performance of the client local computation varies from 10 MFlop/s to 20 MFlop/s as indicated in the dashed line. In SS20, the performance depends on the problem size due to cache effect.

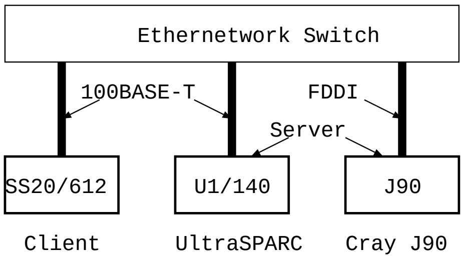

Figure 4. Testbed configuration

In the Ninf program of Linpack, see Figure 5, the client calls a Ninf library function which executes two Linpack routines, `dgefa` and `dgesl`, in the Ninf server. The Ninf program using Ultra achieves about 20 MFlop/s for large sizes. The performance program using J90 increases upto 180MFlop/s at size 1600 as the problem size becomes larger. Note that its performance is not so good that the local performance in J90 is 600 MFlop/s. We found serious overhead of invoking stub executable programs and interprocess communication in J90 systems, which we could not identify in Ultra. Currently, we are identifying the performance bottleneck in J90 to remove this overhead.

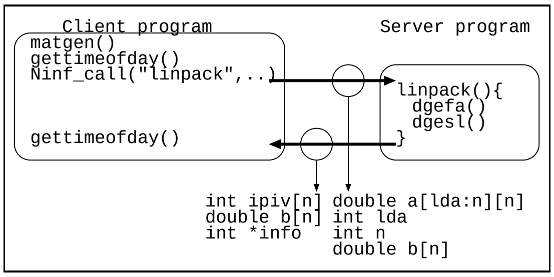

Figure 5. Program structure used for the performance measurement

For the matrix size n, the computation time is $O(n^3)$ and the data transfer time is $O(n^2)$. As the matrix size is increased, the Ninf program can take more advantage of high performance remote machines.

3. Current Status and Future Work

As the Ninf project is an on-going effort, we do expect various changes and extensions to occur as we gain more experience from real applications, and as the system becomes more mature through widespread use.

Currently, we are preparing a suite of Ninf servers on Cray J-90, Sun UltraSparcs, and a 32-processor DEC Alpha workstation cluster at Electrotechnical Laboratory of Tsukuba, Japan for public release. On these servers, we have ported existing mathematical libraries by defined Ninf IDLs and registering with the Ninf server. The libraries include linear algebra packages such as LAPACK, Cray's LibSci, and other special mathematical libraries. A scientific constant database for physics and chemistry is also being designed and being built. The database will have flexible query facilities to cope with physical constants that change minutely over time.

Since our objective is for Ninf to be a global service infrastructure available for free for a wide variety of scientific and engineering use, involving not only high performance but also quality-of-use, there are still numerous research issues to be addressed:

- Authentication and accounting: Although Ninf itself will be available for free, institutions will naturally want to establish its own authentication and accounting policies.

- Exporting of client code: The current Ninf has no facilities to execute client's code on a remote host. Although Ninf has call back mechanism, in some cases, it will be desirable to export client's code for efficiency reasons.

- Fault tolerance: Since global network is not fault-safe, checkpointing and recovery facility will be needed for fault tolerance. We are currently planning to extend the Ninf transaction facilities into full-fledged recoverable transactions; however, doing so without sacrificing computation speed or wasting too much resources is not an easy issue.

- Security: Security is naturally important, especially since each Ninf server will act as computation server and not mere database server. Provisions for entrusted Ninf server nodes, as well as encryption, will be an important issue for future evolution of Ninf.

- Interactive User Interface Tool: Interactive user interfaces are often very attractive for real-world applications. *Ninf-Calc* is a simple WWW-based Ninf interface, provided as a part of the Ninf bowser. The Ninf-Calc allow the users easily to handle linear system solver, eigenvalue problem, and other matrix operation with click and quick operation. Each of the solver routines behind the Ninf-Calc is provided as a Ninf Executable being called through Ninfcall.

- Load Distribution: When the number of Ninf servers are increased, it becomes increasingly difficult for a client to determine which server he should select among the ones in the network. The Ninf metaservers [6] helps to alleviate this situation by gathering the meta-information of the status and the resource made available by the servers, and also by making Ninf services location transparent.

These and other issues will be developed in accordance with advancements in other global networking standards, and other efforts on global network computing.

Acknowledgments

The authors thank the following people for their continuous help concerning this research project: Prof. Umpei Nagashima at Ochanomizu University, Prof. Satoshi Matsuoka at Tokyo Institute of Technology, Prof. Hiromitsu Takagi at Nagoya Institute of Technology, and Dr. Hidemoto Nakada of Electrotechnical Laboratory.

References

[1] P. Arbenz, W. Gander, and M. Oettli. The remote computational system. *Lecture Note in Computer Science, High-Performance Computation and Network*, 1996.

[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.

[4] A. S. Grimshaw. Easy to use object-oriented parallel programming with mentat. *IEEE Computer*, 1993.

[5] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds. Legion: The next logical step toward a nationalwide virtual computer. Technical report, University of Virginia, 1994.

[6] H. Nakada, Kusano, S. Matsuoka, M. Sato, and S. Sekiguchi. A metaserver architecture for ninf: Networked information library for high performance computing. In *SIGNote of Information Prorcessing Society*, number 96-HPC-60, 1996. in Japanese.

[7] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. — ninf — : Network based information library for globally high performance computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA)*, Feb. 1996.