Design and Implementations of Ninf: towards a Global Computing Infrastructure

Hidemoto Nakada [a], Mitsuhisa Sato [b], Satoshi Sekiguchi [a]

[a] *Electrotechnical Lab. Tsukuba, Ibaraki, 3058568, Japan*

[b] *Real World Computing Partnership, Tsukuba, Ibaraki, 3050032, Japan*

http://ninf.etl.go.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 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 overview 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, and that promises the future reality of world-wide network computing.

Keywords: Distributed computing, Global computing, Scheduling

1 Introduction

Remarkable growth of computer network technology has spurred 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, which is being built on the world-wide computing infrastructure, is becoming much more important.

A few systems are proposed for such computing system: Legion [1] is a project for constructing the nationwide virtual computer comprised of many supercomputers and workstations. NetSolve [2] provides an easy-to-use programming interface similar to ours and has a scheduling module called Agent. The Remote Computation System (RCS) [3] is a remote procedure call facility that provides uniform access to remote computers ranging from high-end workstations to supercomputers. The Globus [4] is a toolkit which intended to achieve a vertically integrated treatment of application, middle-ware, and network.

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 [5]. Our goal is to develop a platform for global scientific computing with computational resources distributed in a world-wide global network. This article describes the motivation of the Ninf, its components, the underlying technologies that support such global computing, and the current status of the project.

2 Overview of the Ninf System

2.1 Design Concept

The basic concept of the Ninf system is to support client-server based computing. The computational resources are available as the *Ninf remote libraries* at remote computation hosts. The remote libraries can be called through the global network from a programmer's client program written in existing languages such as Fortran, C, or C++. Parameters, including large arrays, are efficiently marshaled and sent to the *Ninf server* on a remote host, which in turn executes the requested libraries and sends back the results. The *Ninf remote procedure call (RPC)* is designed to provide programming interface that will be very familiar to the programmers of existing languages. The programmer can build 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 one's 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 *Ninf metaserver* maintains the information of Ninf servers in the network, and automatically allocates remote library calls dynamically on appropriate servers for load balancing, or scheduling. Ninf also provides an API for aggregate invocation of multiple remote computation.
- 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.

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 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 the 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 manner. 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 a heterogeneous environment, Ninf uses the Sun XDR data format as a default protocol. Also, clients can 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 a C program 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 not specified here. The server is allocated by the scheduler.

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. Moreover, once a function is called, interface information of the function is cached in the client program, allowing to skip the interface obtaining phase for subsequent invocation of the function. The interface information includes the number of parameters, these types and sizes and access modes of arguments (read or write). Using these information, 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, such as Sun's RPC[6] or CORBA[7], where stub generation has to be 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.

Ninf computational server has an ability to access Web servers directly, enabling users to specify data on Web as arguments for mathematical libraries registered with Ninf. Just using URL string as the data, users can use the data located with the given URL. This code below uses data on the Web as

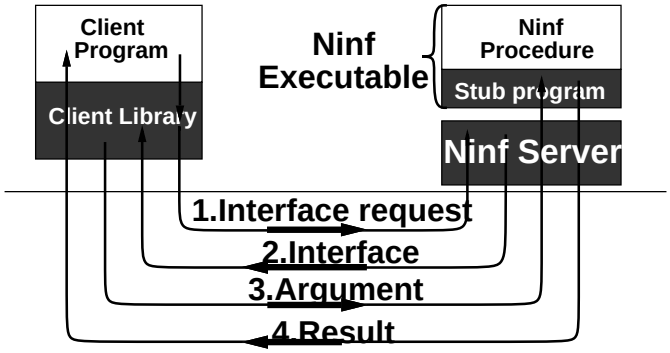

Fig. 1. Ninf RPC

the second argument matrix.

```
Ninf_call("dmmul",N, A, "http://.../..", C)
```

The Ninf client library automatically detects that the argument specifies an URL, and sends the URL to the server. The Ninf server receives the URL and connects to the Web server, gets the specified data by issuing the HTTP GET command, stores it in memory, and then invokes the numerical routine.

The users can also store there data on Web servers[1]. Ninf executable can upload data directly to Web servers. This facility can be used, for example, to snapshot the intermediate results of the server computation on a Web server nearby the Ninf computational server.

The Ninf RPC may also be invoked asynchronously to exploit network-wide parallelism. The API Ninf_call_async is almost equivalent to the ordinary Ninf_call, but it does not wait for the completion of the request, instead, it returns with a request ID, immediately, With the ID, programmers can wait or poll the reply for the request. It is possible to issue the request to a Ninf server, continue with the other computation. Multiple RPC requests to different servers are also possible. For this reason, the asynchronous Ninf RPC is an important feature for parallel programming. There are several APIs for waiting the reply, enabling programmers to wait an arbitrary set of NinfCalls.

Ninf also provides aggregate call interface, which schedules user specified code block, called *transaction* as one scheduling unit. The block of code surrounded by Ninf_transaction_begin and Ninf_transaction_end are not executed immediately; rather, at the end of the code block, the metaserver schedules the computation to multiple computational servers accordingly. When all the computations are complete, the metaserver returns the result to the client. This facility is useful for the parallel execution of the problems that can be easily

[1] It uses http PUT command. The PUT command is optional and public Web servers often do not implement it because of security.

divided into several sub problems, such as, monte-calro simulation or parameter survey computation. The transaction also can be used for macro-dataflow execution. In the transaction, data-dependencies between Ninf_calls are automatically detected, and the metaserver takes it into account for scheduling.

As an example, consider to add up four matrices A, B, C, and D, with the result in G:

```
Ninf_transaction_begin();
Ninf_call("madd", n, A, B, E);
Ninf_call("madd", n, C, D, F);
Ninf_call("madd", n, E, F, G);
Ninf_transaction_end();
```

As the addition of A, B and C, D could be executed in parallel, they are executed in parallel, while the addition of the resulting E, F are executed after their termination.

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 functional arguments to call user-supplied routines 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 call-back facility is useful to call user-supplied routines to display or record data at each time-step while keeping internal state in a server.

Since the Ninf client programming interface is designed to be as language independent as possible, the Ninf client can be written in a variety of programming languages. We have already designed and implemented the client `Ninf_call` functions for C, FORTRAN, Java. Adding to it, we provide a sub API for API implementation. It is possible to implement a client interface to Ninf for any language, so long as it supports standard foreign function interface to C programs. However, it is not a trivial job to implement the Ninf API on a language that employs an array representation that is totally different from C, such as Lisp. In order to make implementation of Ninf API easy, we provide sub-API called Ninf CIM(Common Interface Module). It requires API implementers to provide a few basic routines, called *traversers*. The routines traverse on the language native data structure and set/get data at that position. Ninf_cim_main(), CIM version of Ninf_call(), marshals / unmarshals data calling these traversers. We already implemented APIs for Excel, Mathematica and Lisp using the CIM.

2.4 Ninf IDL (Interface Description Language)

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, 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 information needed to compile and link the libraries.

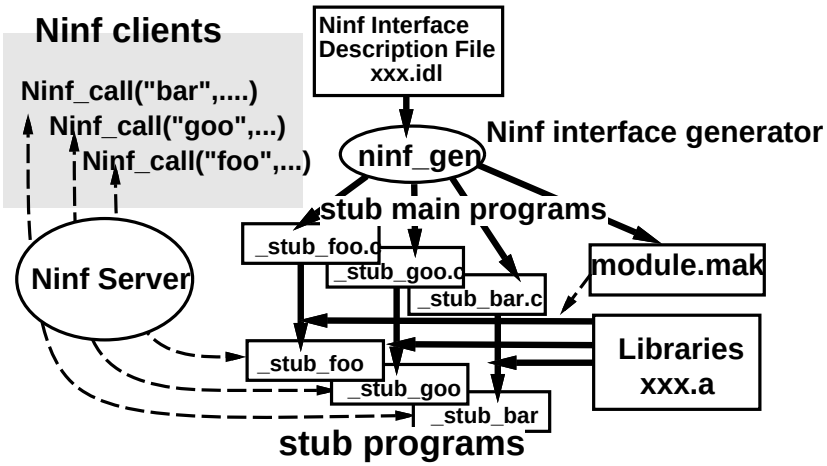

Fig. 2. Ninf interface generator

As illustrated in Fig. 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 create manual pages automatically 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.

3 Scheduling for Global Computing

One of the biggest research issue for global computing is scheduling, i.e., how to achieve efficient usage of computing resources. In this section, we show our scheduling framework, *Ninf metaserver* , which provides scheduler and resource monitor. We also provide our simulation model, which will be used to pursue the optimal scheduling algorithm.

3.1 Requirements for global scheduler

A global scheduler must gather globally distributed information on computational and communication resource utilization. Based on the collected information, which only represents the past snapshot information of the entire system, the scheduler must also make prediction on the current status of resource usage. To facilitate such prediction, all the collected information must be maintained in a global database manager.

To be more specific, a scheduler for a global computing system will consist of the following modules:

- Load monitor for computing resources (i.e. computation server node),
- Throughput monitor for networks,
- Resource information database manager,
- Resource status predictor
- Resource Scheduler (embodying some scheduling algorithm)

Note that, for the throughput monitor, throughput among the client and the computing nodes will have to be measured by each client, as under most WAN settings including the current Internet, throughput between any two nodes is difficult to interpolate from other throughput information. On the other hand, load monitor can be performed by just one representative node

over a collection of computing server nodes.

Monitors report status to the resource information database manager periodically; based on the information in the database, the resource status predictor predict the current and the future status of the resources. Then, the scheduler combines the information of the database and the predictor, and allocates a proper computing node for each request.

3.2 Ninf MetaServer

Accounting the above issues, we have implemented a prototype scheduling framework called the *Metaserver*. The metaserver consists of following modules;

- Load monitor for computing resource
- Client proxy
- Resource Information Database Manager
- Resource Scheduler

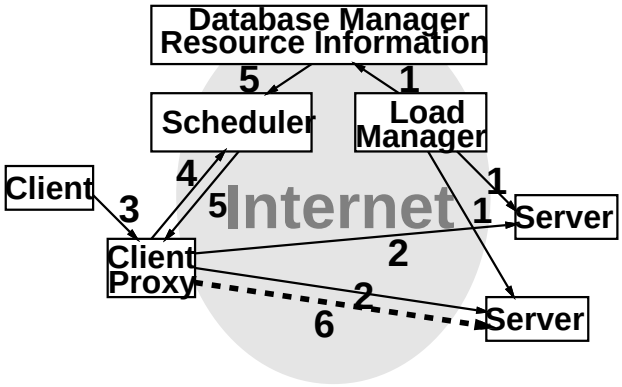

Fig. 3. Ninf MetaServer

The client proxy has two roles. Primary, it's a throughput monitor for network. It monitors throughput on behalf of the entire client site, reducing the cost of measuring the throughput from each client in the site. It also acts as the proxy server for firewall protected networks.

The metaserver system functions in the following way (Fig. 3):

(i) The load monitor monitors the computing servers' load status, and stores it to the resource information database.
(ii) The client proxy periodically monitors the communication status (such as network load) to servers on behalf of the client, and maintains the results internally.
(iii) The client sends a scheduling request of a Ninf invocation to the client proxy.
(iv) The client proxy, in turn, issues a query to the scheduler, attaching its internal communication status information between it and the servers.
(v) The scheduler obtains the computing servers' load information from the resource information database and decides the most appropriate server for

the invocation depending on the characteristics of the requested function (usually the server which would yield the best response time, while maintaining reasonable overall system throughput), and informs the Client proxy of the server identity.

(vi) The client proxy forwards the request to the allocated server.

All of the modules are implemented in Java, and the core scheduling modules are implemented as being "pluggable", allowing the scheduling policy to be replaced on the fly via network protocols.

3.3 Simulation model for global computing

There are several work in scheduling for global computing, include AppLeS[8], Prophet[9] and Condor [10]. However, these systems do not prove the validity of their scheduling algorithms. Experiments on an actual system are not sufficient to discuss the general performance of these algorithms. Rather, an appropriate performance evaluation model, which can represent various global computing systems and their behavior, would be required. To study validity of scheduling algorithms on various environments, we propose a simulation model based on queuing model.

Fig. 4 gives the overview of our proposed simulation model of a typical RPC-based global computing system. In the figure, the clients A–A', B–B' and C–C' are the same clients acting as senders and receivers.

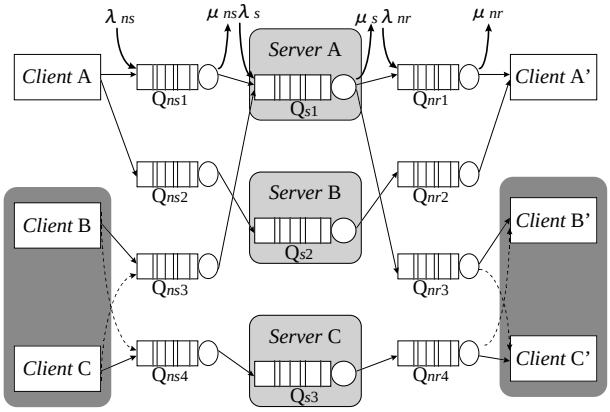

Fig. 4. A Simulation Model of Global Computing System

In the model, invocation and argument communication from the client to the server, processing at the server, and returning of the reply from the server to the client, are modeled as queues Q_{ns_i}, Q_{nr_i}, and Q_{s_i}, respectively. λ_{ns}, λ_{nr} are arrival rates of requests in the network other than Ninf_call ; in other words, they model congestion and delays encountered in a wide-area network. μ_{ns},

μ_{nr} are service rate of the queues, i.e., they effectively represent the network bandwidth, and they follow exponential distribution. λ_s, μ_s similarly represent degree of server congestion, and server processing power.

A simulator based on this model is already implemented in Java, and we confirmed the validity of the simulation model comparing the results of virtual simulations and actual experiments. We are pursuing the optimal scheduling algorithm using the simulator and the metaserver.

4 Current Topics

4.1 Performance Evaluation using the Ninf RPC

Here, we give some preliminary evaluations results. As we already reported more precise evaluation result[11], we concentrate simple single client - single server case, here. As a benchmark program, we employ Linpack which requires extensive communication to ship dense matrices over the network. We employed SuperSPARC(40MHz) and UltraSPARC(143MHz) as the clients, and DEC Alpha(333MHz) and Cray J90(with 4CPUs) as the servers.

Fig. 5 shows the results of SuperSPARC and UltraSPARC clients. The horizontal axis indicates the size of the matrix, and the vertical axis indicates the performance of n Ninf_call and Local in Mflops. The performance of Local remains relatively constant across for both SPARCs. On the other hand, Ninf_call performance improves steadily as n increases, and for both SPARCs, exhibited superior performance to Local at approximately n = 200-400.

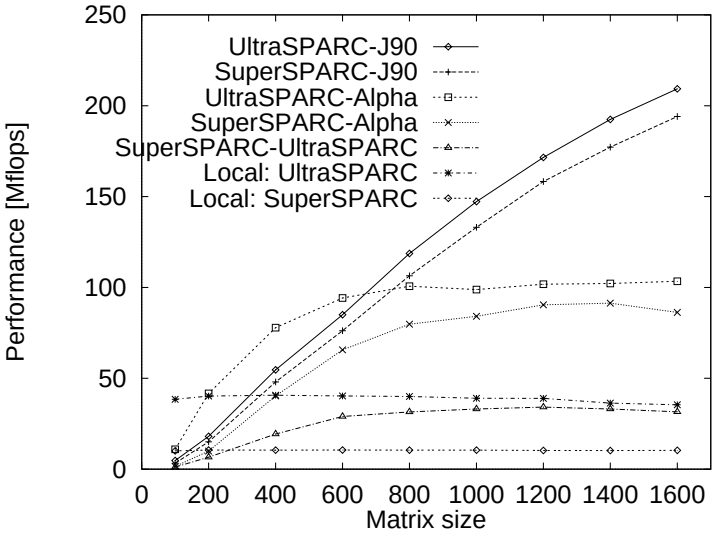

Fig. 5. Ninf_call performance

4.2 Ninf-NetSolve Adapter

NetSolve[2] is a global computing system being developed at Univ. of Tennessee. The functionality and the architecture of NetSolve are similar to Ninf: however, there is a great deal of difference in the protocols and some of the system APIs. By interfacing Ninf and NetSolve, we expect that not only it will benefit the users of both systems due to mutual interoperability, but also we expect that by comparing the two systems and their protocols, we could distill the advantageous characteristics of each, and design new systems and standard protocols. In fact, we are already in the process of proposing a new protocol based on the experiences gained from the adopters. Here, we describe how the adapters work, and their performance.

For brevity, we describe the implementation of the Ninf → NetSolve adapter: the inverse adapter is similar in principle. The adapter acts as a computational server on the Ninf side, and acts as a client on the NetSolve side. It is written in Java, and has the following roles:

- *Conversion of NetSolve interface information to Ninf interface information* — Both systems make a two-phase call to the server, where the interface information of the callee library is sent to the client.
- *Network data transfer format conversion* — The network data format differs considerably between Ninf and NetSolve.

We are now under negotiation with NetSolve people to standardize a data transfer protocol, which can be used by not only our systems but also other metacomputing systems.

4.3 GUI based client for Ninf

Interactive user interfaces are often very attractive for real-world applications. *NinfCalc+* is a simple WWW-based Ninf interface. which is implemented as a Java applet and allows 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 NinfCalc+ is provided as a Ninf Executable being called through Ninfcall.

NinfCalc+ utilizes direct web access facility, i.e., it does not store matrices in itself, uses Web servers as storages, instead. As it performs calculation control only, it does not require broad band-width between Ninf server and itself. It can control huge matrix calculation interactively even from your home via thin phone line.

Fig. 6. NinfCalc+

5 Future Plans

In this article, we described our global computing infrastructure: Ninf, and showed preliminary performance evaluation results. 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.
- Security: Security is naturally important, especially since each Ninf server will act as computation server and not mere database server. Provision of entrusted Ninf server node, as well as encryption, will be an important issue for future evolution of Ninf.
- 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 many resources is not an easy issue.
- Scheduling: Although we already have the flexible scheduling framework: Metaserver, the scheduling and resource prediction algorithm is still an open issue. We will pursue the issue utilizing our simulator.

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

Acknowledgment

We would like to thank other members of the Ninf group, especially Satoshi Matsuoka(Tokyo Institute of Technology), Umpei Nagashima (National Institute of Materials and Chemical Research), and Atsuko Takefusa (Ochanomizu University) for their discussions and encouragements.

References

[1] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds Jr. Legion: The next logincal step toward a natiowide virtual computer. CS 94-21, University of Virginia, 1994.

[2] Henri Casanova and Jack Dongarra. Netsolve: A network server for solving computational science problems. In *Proceedings of Super Computing '96*, 1996.

[3] Peter Arbenz, Walter Gander, and Michael Oettli. The remote computation system. Technical Report 245, ETH, 1996.

[4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. In *Proc. of Workshop on Environments and Tools, SIAM.*, 1996.

[5] Mitsuhisa Sato, Hidemoto Nakada, Satoshi Sekiguchi, Satoshi Matsuoka, Umpei Nagashima, and Hiromitsu Takagi. Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture. In *Proc. of HPCN'97 (LNCS-1225)*, pages 491–502, 1997.

[6] Sun Microsystems, Inc. RPC: Remote procedure call protocol specification version 2. RFC 1057, June 1988.

[7] OMG. *The Common Object Request Broker: Architecture and Specification. Revision 2.0.* OMG Document, 1995.

[8] Fran Berman, Rich Wolski, Silvia Figueira Jennifer Schopf, and Gary Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Super Computing '96*, 1996.

[9] J.B. Weissman and X.Zhao. Scheduling parallel applications in distributed networks. In *Journal of Cluster Computing*, 1997.

[10] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Submitted to HPDC'98*, 1998.

[11] A. Takefusa, S. Matsuoka, H. Ogawa, H. Nakada, H. Takagi, M. Sato, S. Sekiguchi, and U. Nagashima. Multi-client lan/wan performance analysis of ninf: a high-performance global computing system. In *Supercomputing '97*, 1997.