Design issues of Network Enabled Server Systems for the Grid

Satoshi Matsuoka,[1,2*] Hidemoto Nakada,[3**]
Mitsuhisa Sato[4***] and Satoshi Sekiguchi[3†]

[1] Tokyo Institute of Technology, Tokyo, Japan
[2] Japan Science and Technology Corporation, Tokyo, Japan
[3] Electrotechnical Laboratory, Tsukuba Ibaraki, Japan
[4] Real World Computing Partnership, Tsukuba Ibaraki, Japan

Abstract. Network Enabled Server is considered to be a good candidate as a viable Grid middleware, offering an easy-to-use programming model. This paper clarifies design issues of Network Enabled Server systems and discusses possible choices, and their implications, namely those concerning connection methodology, protocol command representation, security methods, etc. Based on the issues, we have designed and implemented new Ninf system v.2.0. For each design decision we describe the rationale and the details of the implementation as dictated by the choices. We hope that the paper serves as a design guideline for future NES systems for the Grid.

1 Introduction

A Network Enabled Server System(NES) is an RPC-style Grid system where a client requests the service of a task to a server. There are several systems that adopt this as the basic model of computation, such as our Ninf system[1], Netsolve[2], Nimrod[3], Punch[4], and Grid efforts utilizing CORBA[5, 6, 7].

NES systems provides easy-to-use, intuitive, and somewhat restricted user and programming interface, This allows the potential users of Grid systems to easily make his applications "Grid enabled", lowering the threshold of acceptance. Thus, we deem it as one of the important abstractions to be layered on top of lower-level Grid services such as Globus[8] or Legion[9].

Since 1995, we have been conducting the Ninf project, whose goal has been to construct a powerful and flexible NES system [10, 11], and have investigated the utility of such systems through various application and performance experiments[12]. There, we have gained precious experience on the necessary technical aspects of NES systems which distinguishes them from conventional RPC systems such as CORBA, as well as various tradeoffs involved in the design of such systems[13]. Based on such observations, we have redesigned and reimplemented version 2.0 of the Ninf system from scratch.

* matsu@is.titech.ac.jp
** nakada@etl.go.jp
*** msato@trc.rwcp.or.jp
† sekiguchi@etl.go.jp

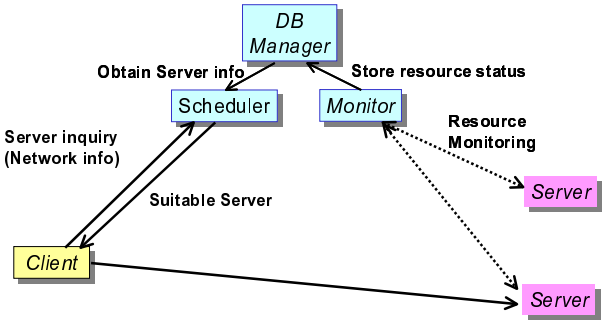

Fig. 1. General Architecture of NES Systems

The purpose of this paper is to discuss the notable technical points which led to the design decisions made for Ninf v.2.0. In particular, for the latter half of the paper we focus on the security issues, which is mostly lacking in the current generation of NES systems.

2 General Overview of NES Systems

In general, NES systems consists of the following components: (Figure 1)。

- **Clients:** Requests execution of grid-enabled libraries and/or applications to the server.
- **Servers:** Receives request from clients, and executes the grid-enabled libraries and/or applications on clients' behalf.
- **Scheduler:** Selects amongst multiple servers for execution according to the information obtained from the resource database.
- **Monitors:** Monitors the status of various resources, such as computing resource, communication resource, etc., and registers the results in the resource database.
- **Resource Database:** Stores and maintains the status of monitored resources.

The Monitors periodically "monitor" the status of resources such as the server, network, etc., and registers the results in the Resource Database. The users of Grid systems modifies his applications to utilize the servers with the use of client APIs, or tools that have been constructed using the client APIs. The Client inquires the Scheduler for an appropriate Server. The Scheduler, in turn, acquires the info on computing resources, and selects the appropriate server according to some scheduling algorithm, and returns the selection to the Client. The Client then remotely invokes the library/application on the selected server by sending the appropriate argument data. The server performs the computation, and returns the result data to the client.

2.1 Design Issues in NES Systems

There are several design issues regarding the construction of NES systems, including the connection methods of client and servers, communication protocols, and security. Moreover, there is an issue of how we make the system open to future extensions.

Client-Server Connection Methodologies The client must first establish a connection with the selected server. The sub-issues involve 1) continuous connection versus connection-by-necessity, and 2) usage of proxies.

Continuous Connection versus Connection-by-Necessity: Continuous connection maintains connection between the server and the client during the time server is performing the computation. Contrastingly, Connection-by-Necessity makes fine-grain connection/disconnection between the client and the server on demand.

Continuous connection is typically employed for standard RPC implementations; it is easy to implement under the current TCP/IP socket APIs, and furthermore, allows easy detection of server faults via stream disconnection. The drawback is the restriction on how many parallel tasks that can be invoked by a client. Since the connection to the server must be maintained, the client process is requires more file descriptors than the number of parallel tasks being invoked. However, since the number of file descriptors per process is restricted for most OSes, this limits the number of parallel tasks. Such has not been a problem for traditional RPCs, since most transactions are short-lived, and/or the number of connections were small since the user tasks are sequential.

Moreover, continuous connection requires the client to constantly be on-line, without any interruption in the communication. Thus, the client cannot go off-line, neither deliberately nor by accident; even a momentary failure in the communication will cause a fault. This again is a restriction, since some Grid-enabled libraries may take hours or even days to compute.

By contrast, in Ninf v.2.0 we have adopted Connection-by-Necessity. Basically, when the client makes an RPC request to the server, it disconnects once the necessary argument data had been sent. Once the server finishes the computation, it re-establishes a new connection with the client, and sends back the result. This overcomes the restriction of the Continuous Connection, but a) the protocol becomes more complex, due to the requirement of server-initiated and secure connection re-establishment, b) there need to be an alternative method of detecting server faults, and c) performance may suffer due to connection costs.

Direct Connection versus Proxy-based Connection Another concern is whether to connect the client and the server directly, or assume a dedicated, mediating proxies, for various purposes including connection maintenance, performance monitoring, and firewall circumvention.

The "old" Ninf system (up to v.1.2.) employed proxy-mediated connection, for the purpose of simplifying the client libraries. All traffic was mediated by the proxy; in fact, communication with the Scheduler for server selection was

performed by the proxy and not by the client itself. On the other hand, routing the communication through the proxy will result in performance overhead, which is of particular concern for Grid systems since communication of large bulk of data is typical.

Communication Protocol Commands for Grid RPC Communication Protocol Commands, or simply Protocol Commands are a set of commands that are used to govern the communication protocol between the client and the server. They can be largely categorized into binary formats and text-based formats.

Binary formats allow easy and lightweight parsing of command sequences, but are difficult to structure, debug and extend. Contrastingly, well-designed text-based formats are well-structured, easy to understand and extent, but are less efficient and require more software efforts to parse.

Although traditionally text-based commands for communication protocols were typically simple, involving little structure such as S-expressions, there is a recent trend to employ XML for such purpose. Although XML requires more efforts on the software side for parsing etc., we can assign schema in a standard way using DTD. Since command overhead can be amortized over relative large data transfer, we believe XML is a viable option given its proliferation as well as availability of standard tools.

Security Mechanism Security is by all means an important part of any Grid system. However, there several options for security, depending on the operating environment of the system.

If the operating environment is totally local within some administrative domain, where all the participants can be trusted, we can merely do away with security. In a slightly more wide-area and well-administered environment, such as within a University campus, it suffices to restrict access based on, say, client IP address. On the other hand, if global usage is assumed, then by all means we must guard against malicious users, and thus require authentication based on encryption. Examples are Kerberos, which employs the symmetrical key technology, and SSL, which utilizes the public key algorithm.

System Openness and Interoperability with Other Grid Systems One important design choice is how much we make the system open to customization, especially with respect to other, more general Grid software infrastructure, and/or Grid component with some specific function. More concretely, Grid toolkits such as Globus provide low-level communication layer, security layer, directory service, heartbeat monitoring, etc. Components such as NWS(Network Weather Service[14]) provides stable monitoring and prediction services for measuring resources on the Grid, such as node CPU load and network communication performance. Conventional components which had initially not intended as a Grid services could be incorporated as well, such as LDAP, which provides a standard directory service API; Globus employs LDAP directly with its MDS(Metacomputing Directory Service), providing a Grid directory service.

By using such existing subsystems and components, we can directly utilize the functionalities which had been tried and tested, and also subject to independent improvement. On the other hand, because such subsystems are designed for generality, they have larger footprint, and could be tougher to manage. Moreover the supported platform would be the intersection of the platforms supported by individual subsystems.

3 Design and Implementation of the New Ninf System

3.1 Conceptual Design Decision Overview

We designed and implemented a new version of the Ninf system (Ninf version 2.0) with the abovementioned design issues in mind. The new system is designed to be flexible and extensible, with interoperability with existing Internet and Grid subsystems in mind. Because NES systems typically involve tasks where computation is dominant, we made design decisions that gave precedence to interoperability and flexibility over possible communication overhead if such could be amortized.

Client-Server Connections In order to accommodate multiple, fault-tolerant, long-running calls in Grid Environments, we adopted for connection-by-necessity over continuous connections. We have also decided to employ proxy-based connections in order to simplify client structure. However, in order to avoid bandwidth bottlenecks, proxies only intervene on command negotiations between the client and server; when the actual arguments of the remote call is being transferred, the client and the server communicate directly, unless a firewall must be crossed.

Communication Protocol Commands For flexibility, extensibility, and interoperability, we decided to adopt the usage of XML-based text commands. In the latter sections we present an overview of the DTD schema for numerical RPCs. Free parsers for C and Java are available, which simplified our implementation.

Security Mechanism To allow Ninf to be used in a global Grid environment, we opted to construct a Globus-like, SSL-based authentication and authorization layer, which allows delegation of authentication along a security chain. Kerberos was an obvious alternative, but SSL was becoming a commercial standard, and multiple free library implementations in C and Java are available.

System Openness and Interoperability with Other Grid Systems This was the most difficult decision, since advantages and disadvantages of employing existing Grid components could be strongly argued both ways. As a compromise, we have decided to provide default implementations of all our basic submodules;

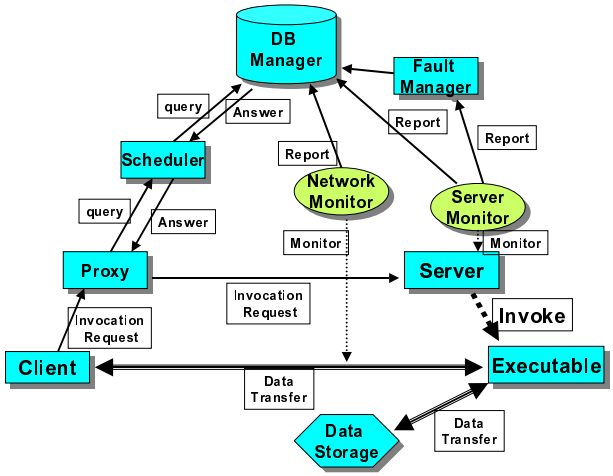

Fig. 2. Overview of the New Ninf System

however, we have designed them to have well-defined interfaces, to be pluggable with existing modules in operating environments where such services are already available. For example, although the default implementation of the resource database lookup service has its own LDAP lookup feature, it could also directly utilized Globus MDS services where they are available.

3.2 Overview of the New Ninf System V.2.0

The new Ninf system v.2.0 is composed of the following subsystems(Figure 2)。

- **Client**
  A user-side component which requests (parts of) computing to be done on remote servers in the Grid. The client is "thin" in a sense that as little information as possible is retained on the client side; for example IDL of the remote call is not maintained by the client, but rather automatically shipped on demand from the server.
- **Server**
  Receives remote compute requests from the clients and invokes the appropriate executable. The server might act as a backend for invoking parallelized libraries on multiple compute nodes, such as a library written in C/Fortran+MPI served by a Cluster.
- **Proxy**
  Communicates with a Scheduler on behalf of the Client, and decides upon which server to invoke the remote computation, and forwards the request to

the server. (The behavior of the proxy is similar to Netsolve Agents in this case.)

- **Executables**
  Components which actually embeds each remote applications or libraries to be invoked. They are invoked by the server, and communicate with the client to perform the actual computations.
- **Data Storage**
  Temporary storage on the Grid to store intermediate results amongst multiple servers.
- **Scheduler**
  The scheduler receives requests from the proxy, and selects an appropriate server under some scheduling algorithm. The scheduler communicates with the database server in order to drive the scheduling algorithm.
- **Database Manager**
  Manages the Information Stored in the Grid resources database. The database itself utilizes existing distributed resource database for the Internet and/or the Grid (e.g., LDAP or Globus MDS, which in turn uses LDAP itself); the resource lookup request from the client is delegated through the manager. This naturally allows other database infrastructure to be utilized.
- **Network Monitor/Server Monitor**
  Monitors the status of the network, servers, and other resources. The result is reported periodically and automatically to the database manager.
- **Fault Manager**
  Performs recovery action when some fault or error that affects the system in a global way, is detected. For example, if the server is found to be down (using heartbeat monitoring), the server is deleted from the resource database.

3.3 Client-Server Communication in the Ninf System 2.0

The new Ninf system manages the client-server communication in the following manner(Figure 3):

The client first requests the interface information of the executable to be invoked to the proxy. It then requests the invocation of the executable. The client immediately disconnects its connection with the proxy, and enters the state waiting for a callback from the proxy. The client then can proceed to issue hundreds of simultaneous requests, as there are no other pending connections.

The proxy in turn inquires the Scheduler for selection of an appropriate server (or a set of servers) to perform the invocation. The scheduler inquires the database manager for information on servers and network throughput information, as well as other resource information such as location of files used in the computation. The scheduling algorithm selects an appropriate server (or set of servers) and returns the info to the proxy. The algorithm itself is pluggable; one can employ simple algorithm as is employed with netsolve (sorting by server load), or more sophisticated algorithm such as those employed by Nimrod.

The proxy forwards the invocation request to the selected server. The server in turn invokes the executable for performing the actual computation. The exe-

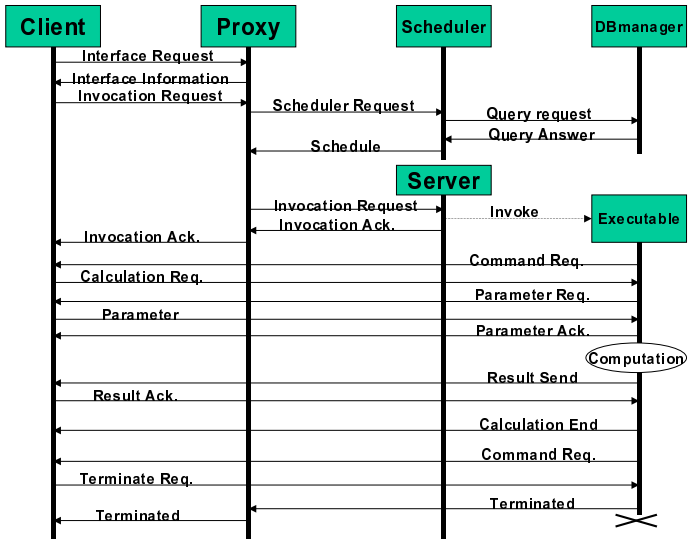

Fig. 3. Invocation Protocol

cutable then requests to the client the necessary arguments by sending the appropriate IDL program for marshalling. When all the arguments have been received, the executable notifies the client, disconnects the connection, and proceeds to compute the request. The client again enters the state to wait for callback from the executable on completion of the invocation.

When the computation is finished, the executable reconnects with the client, and transmits the result, indicating termination of the invocation. The client acknowledges the receipt with the termination command.

Finally, the executable notifies the proxy that the invocation has terminated. The proxy in turn forwards this to the client. The proxy notifies the Database Manager of the termination, allowing it to update the resource database.

3.4 Communication Protocol Commands in New Ninf

As an example of communication command protocol, we demonstrate the DTD of the protocol command for specifying and invoking on a server a remote executable, in Figure 4. Based on this DTD, here is sample invocation command in XML(Figure 5).

One may notice that the invocation command embodies two addresses, client and observer; here, client is the address used for client callbacks, where as observer is the address used to notify termination of invocation to the proxy.

```
<!ELEMENT invoke_executable
  (issuer ,function_name, client, observer)>
<!ELEMENT issuer EMPTY>
<!ATTLIST issuer process CDATA #REQUIRED>
<!ATTLIST issuer host CDATA #REQUIRED>
<!ATTLIST issuer port CDATA #REQUIRED>
<!ATTLIST issuer session_key CDATA #REQUIRED>
<!ELEMENT function_name EMPTY>
<!ATTLIST function_name module CDATA #REQUIRED>
<!ATTLIST function_name entry CDATA #REQUIRED>
<!ELEMENT client (peer)>
<!ELEMENT observer (peer)>
<!ELEMENT peer EMPTY>
<!ATTLIST peer host CDATA #REQUIRED>
<!ATTLIST peer port CDATA #REQUIRED>
```

Fig. 4. Remote Executable Command DTD

```
<invoke_executable>
<issuer process="nsserver"
    host="hpc.etl.go.jp"
    port="30000" session_key="12345" />
<function_name module="test" entry="mmul" />
<client>
   <peer host="hpc.etl.go.jp" port="30000" />
</client>
<observer>
    <peer host="hpc.etl.go.jp" port="30001" />
</observer>
</invoke_executable>
```

Fig. 5. Example Invocation Command

3.5 Security Layer in the new Ninf system

Security in a NES system involves Authentication, Authorization, Privacy. Authentication identifies *who* is connecting to the server; authorization is *what* resources to permit to the user that has been identified; and privacy is to make communication and computation private to other users connecting to the NES system.

The new Ninf system has the client connect to the server via a shared proxy; however, server authentication and authorization must be performed with client identify, with (rather remote but still existing) possibility that the proxy may be

spoofed. Another situation is when server A acts as a client and delegates part of its work to server B on another machine. There, not only that server A needs to be authenticated, but the client identity must be authenticated and authorized at server B as well. Such "delegation of identity" we deem as essential part of a NES system

The new Ninf system implements the NAA (NES Authentication Authorization) module. NAA employs SSL as the underlying encryption mechanism and implements delegation of identity and authorization on top of those. Delegation of identity is done automatically by the NAA, and the client user merely needs to specify his certificate as is done with SSL. NAA itself is relatively self-contained, and thus could be used by other NES systems such as Netsolve.

Delegation of Identity Identity in SSL consists of a certificate certified by a CA (Certificate Authority). CA's can be made hierarchical—it is possible to sign a certificate using another (signed) certificate. In NAA, we have implemented delegation of identity by not merely directly tying in user identity with his certificate, but rather, broadened the 'identity' to include all the certificates signed using the user's certificate.

SSL employs the public key encryption algorithm, where its certificate consists of user's public key being encrypted by CA's private key. We can form a so-called *certificate chain* by generating another key pair, and encrypting them with the user's private key. On authentication, CA's public key is used to decrypt the certificate, which reveals the public key of the user. This could be used for identification (by decrypting data which had been encrypted with the private key of the user), or for a chain, could in turn be used to obtain the public key of the next element in the chain. NAA uses such a certificate chain for authentication, in that if the user's certificate appears somewhere in the chain, it is regarded as providing the user's indentity. The security layer of Globus employs a similar strategy[15].

As an example, let us consider when the client calls server A, which in turn calls server B as a client. When server A receives a connection request from the client, it generates a new key pair, and sends back its public key to the client, which is asked to create a session certificate embodying its identity. The client generates a session certificate by signing (encrypting) the public key with its own private key, and sends it back to server A. When server A connects to server B, server B must 1) authenticate the identity of the client, as well as 2) identify that the call is being made through server A. This is achieved by server A connecting with the session certificate received from the client along with the original certificate of the client. This is shown in Figure 6.

NAA Policies We have designed NAA policies to be extensible and customizable by the system administrators.

After the client is authenticated, authorization in NAA is performed using a structure similar to Java 1.2 Policy class. A *policy* is a set of structures called

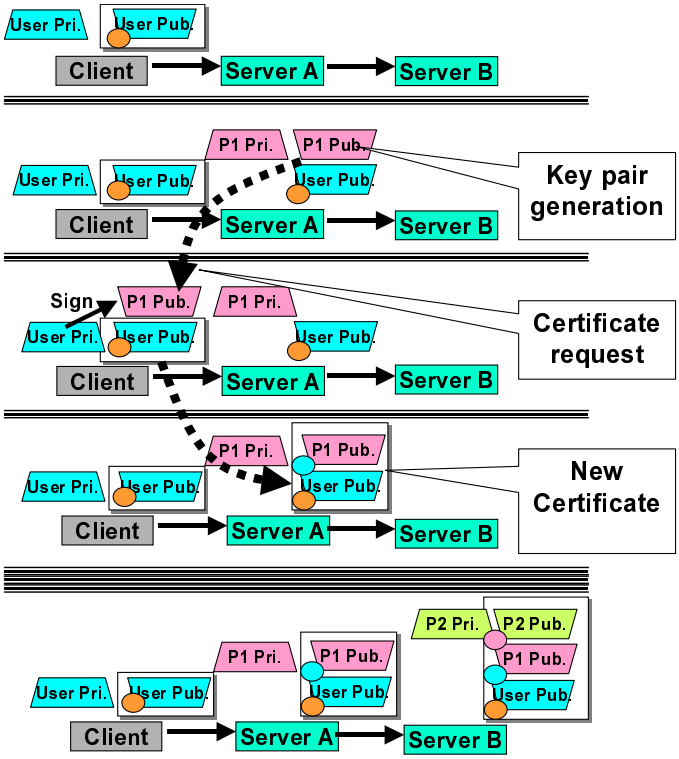

Fig. 6. Delegation of Identity

grants, which in turn are sets of *permissions* to the user. The NAA library manages the policy structures and the identities of current clients of the system. In addition, NAA namespace is tree-structured according to the X.509 conventions. Access control is done hierarchically done along this tree using the permissions.

The server program inquires whether the certain permission is applicable to the client. The library checks the policy if there are grants that contain the particular permission. Each permission consists of three attributes, *class*, *target*, and *action*. Class indicates the operation that the permission allows the client to perform. Target and action designates the subject of the operation, along with the type of the operation to be performed.

Policies are described using XML in a policy file. We illustrate the policy file DTD and an example of policy description in Figure 7と Figure 8, respectively.

In the example, we have defined two grants. The first grant indicates that the user whose identity includes `c=jp,o=etl` (meaning the Electrotechnical Lab) can remotely execute `test/entry0` and `test/entry1`. The second grant restricts `test/entry3` to only be remotely executed by userID `c=jp,o=etl,CN=nakada`.

```
<!ELEMENT policy (grant)*>
<!ELEMENT grant (permission)*>
<!ATTLIST grant userid CDATA #REQUIRED>
<!ELEMENT permission EMPTY>
<!ATTLIST permission class CDATA #REQUIRED>
<!ATTLIST permission target CDATA #REQUIRED>
<!ATTLIST permission action CDATA #REQUIRED>
```

Fig. 7. Policy DTD

```
<policy>
  <grant userid="c=jp,o=etl">
    <permission class = "stubexec"
        target = "test/entry0" action="100 20"/>
    <permission class = "stubexec"
        target = "test/entry1" action="100 20"/>
  </grant>
  <grant userid=" c=jp,o=etl,CN=nakada">
    <permission class = "stubexec"
        target = "test/entry3" action="100 20"/>
  </grant>
</policy>
```

Fig. 8. Example Policy

Thus, the client `c=jp,o=etl,CN=nakada` can execute all the remote libraries (`entry0, entry1, and entry3`), while the client `c=jp,o=etl,CN=sekiguchi` can execute only (`entry0 and entry1`); furthermore, the client `c=jp,o=titech, CN=matsuoka` cannot execute any of the libraries.

We can also grant rights to specific calls made by the client through delegation of identity; for instance, in the delegation of identity scenario described earlier, we can specify a certain executable to be invoked only if a particular client was executing a library in server A which in turn had called the executable in server B. Such a case is conceivable, when a large compute server B is used as a backend for a server A, which is more subject to public usage; contrastingly, only a restricted set of jobs could be run on server B, and users are not allowed to invoke a remote library on server B directly; rather, they must do so via server A.

In this manner, the hierarchical namespace, along with the policy structure, gives fine-grained access control of resources for remote libraries in a NES system. Preliminary measurement have shown that such mechanisms do not impose significant overhead, as long as the calls granularity is large enough such that

the overhead could be amortized (beyond 10s of seconds).

4 Conclusion

We have covered the technical tradeoff points of NES systems, and described how the new Ninf system v.2.0 had been designed with the tradeoffs in mind, with descriptions of why a particular choices in the tradeoffs had been made. We hope that most of the design spaces have been covered, and will serve as a guide for designing future NES systems.

We are currently in the stage of deploying Ninf v.2.0 alongside v.1.0 to compare and verify the effectiveness of the design decisions, along with performance analysis to assess the their impact as well.

Acknowledgements

Part of this research had been performed under the sponsorship of Information Promotion Agency of Japan (IPA), under the program "The Development of Wide-Area, Distributed Computing Applications". We also would like to thank NTT Software and Computer Institute of Japan who had contributed in the design, our collaborators and users of the Ninf system, and the rest of the Ninf project team for their technical discussions and support.

References

1. Ninf: Network Infrastructure for Global Computing. http://ninf.etl.go.jp/.
2. Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proceedings of Super Computing '96* (1996).
3. Buyya, R., Abramson, D. and Giddy, J.: Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *Proceedings of HPC Asia 2000* (2000).
4. Kapadia, N. H., Fortes, J. A. B. and Brodley, C. E.: Predictive Application-Performance Modeling in a Computational Grid Environment, *Proc. of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)* (1999).
5. René, C. and Priol, T.: MPI Code Encapsulating using Parallel CORBA Object, *Proc. of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pp. 3–10 (1999).
6. Imai, Y., Saeki, T., Ishizaki, T. and Kishimoto, M.: CrispORB: High performance CORBA for System Area Network, *Proc. of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pp. 11–18 (1999).
7. Butler, K., Clement, M. and Snell, Q.: A Performance Broker for CORBA, *Proc. of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pp. 19–26 (1999).
8. Foster, I. and Kesselman, C.: Globus: A metacomputing infrastructure toolkit., *Proc. of Workshop on Environments and Tools, SIAM.* (1996).

9. Grimshaw, A., Wulf, W., French, J., Weaver, A. and Jr., P. R.: Legion: The Next Logincal Step Toward a Natiowide Virtual Computer, CS 94-21, University of Virginia (1994).
10. Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashima, U. and Takagi, H.: Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture, *Proc. of HPCN'97 (LNCS-1225)*, pp. 491–502 (1997).
11. Nakada, H., Takagi, H., Matsuoka, S., Nagashima, U., Sato, M. and Sekiguchi, S.: Utilizing the Metaserver Architecture in the Ninf Global Computing System, *High-Performance Computing and Networking '98, LNCS 1401*, pp. 607–616 (1998).
12. Takefusa, A., Matsuoka, S., Ogawa, H., Nakada, H., Takagi, H., Sato, M., Sekiguchi, S. and Nagashima., U.: Multi-client LAN/WAN Performance Analysis of Ninf: a High-Performance Global Computing System, *Supercomputing '97* (1997).
13. Suzumura, T., Nakagawa, T., Matsuoka, S., Nakada, H. and Sekiguchi, S.: Are Global Computing Systems Useful? - Comparison of Client-Server Global Computing Systems Ninf, NetSolve versus CORBA, *Proc. of International Parallel and Distributed Processing Symposium* (2000).
14. Wolski, R., Spring, N. and Peterson, C.: Implementing a Performance Forecasting System for Metacomputing: The Network Weather service, *Proceedings of the 1997 ACM/IEEE Supercomputing Conference* (1997).
15. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S.: A Security Architecture for Computational Grids, *Proc. 5th ACM Conference on Computer and Communication Security* (1998).