Are Global Computing Systems Useful ?
— Comparison of Client-Server Global Computing Systems Ninf, NetSolve versus CORBA —

Toyotaro Suzumura, Takayuki Nakagawa
Tokyo Institute of Technology
Satoshi Matsuoka
Tokyo Institute of Technology/JST

Hidemoto Nakada, Satoshi Sekiguchi
Electrotechnical Laboratory

Abstract

Recent developments of global computing systems such as Ninf, NetSolve and Globus have opened up the opportunites for providing high-performance computing services over wide-area networks. However, most research focused on the individual architectural aspects of the system, or application deployment examples, instead of the necessary charactersistics such systems should intrinsically satisfy, nor how such systems relate with each other. Our comparative study performs deployment of example applications of network-based libraries using Ninf, NetSolve, and CORBA systems. There, we discover that dedicated systems for global computing such as Ninf and NetSolve have management, progammability, and in does not suffer performance disadvantages over more generic distributed computing capabilities provided by CORBA. Such results indicate the advantage of dedicated global computing systems over general systems, stemming further basic research is necessary across multiple systems to identify the ideal software architectures for global computing.

1. Introduction

Research on global computing systems such as Globus[3], Legion[5], and Condor[14] are starting to proliferate thanks to the advances in high-speed networking. However, although some deployments in realistic situations have been reported, such systems are still largely experimental, and thus face various software challenges to see wide-spread, practical use.

The fundamental question in such a situation is, to what extent are such systems useful and/or required, as compared to existing software infrastructures, especially those for distributed computing. Distributed computing has been studied for a number of years, resulting in industry standards such as CORBA[11]. What are the concrete advantages or disadvantages of new global computing infrastructures compared to such existing infrastructures? Are there features lacking in existing systems so as to make them insufficient or harder to use, or are we not merely 'reinventing the wheel' while existing software infrastructures suffice?

In particular, we have been working on the Ninf system[16, 17, 10], which is an RPC-based global computing system for creating numerical libraries that can be called remotely in a transparent manner. There have been other similar systems, such as Netsolve[2]. The purpose of such system is not be a comprehensive layer to construct numerous possible types of global computing applications; rather, the intent is to provide ease-of-use interfaces to "globalize" (parts of) existing applications. Thus, they can be considered a higher-level middleware serving more specific needs and possibly built on systems such as Globus.

Both Ninf and Netsolve have been 'criticized' for providing an RPC infrastructure, which at a higher abstract level is what essentially industrial RPC standards such as DCE and CORBA already provide. In fact, there have been several attempts at utilizing CORBA for scientific computing including [8, 15, 6, 1]. What then, are the qualitative and/or quantitative advantages of Ninf and/or Netsolve, over existing RPC infrastructures? The aim of this research is to perform various systematic comparisons of RPC-based global computing systems against existing distributed-computing RPC systems, to investigate the viabilities of dedicated global computing systems. This is the first such research we have seen to date as far as we know, and we hope that other types of objective comparisons can be made for other breeds of global computing systems.

More concretely, we have investigated the 'globalization' of an example numerical library as well as a real numerical application (advanced Semi-Definite Programming solver) under Ninf, Netsolve and two CORBA systems: TAO[13] and OmniORB[12], and compared their performance as well as ease of programming, maintainability, etc. Performance-wise we have also studied the viability of substituting the underlying communication layer of Ninf with Nexus library from Globus. We have found that not only that Ninf has performance advantage over the CORBA systems, but also that both Ninf and Netsolve offer functionalities not available in CORBA which greatly reduces the globalization effort of existing libraries and applications. Thus, we have been able to provide strong evidences for viability of dedicated RPC-style global computing systems.

2. Brief Overview of Ninf and Netsolve versus CORBA

Both Ninf and Netsolve are tailored for numerical computing, offering network-based numerical library functionality via the use of RPC technology. Parts of applications making procedure calls can be replaced with high-performance, remote equivalents in a transparent manner, usually only a small modification to the call itself, without any RPC data types, prologue/epilogues, IDL management, etc.

Compared to existing, general-purpose RPC techniques, this is not simple as it seems: for example, for local procedure calls most numerical data structures such as arrays are passed by reference, so the numerical RPC mechanism must provide some shared memory view of the arguments, despite being a remote call.

Furthermore, no information regarding its physical size, or which portions are to be used are maintained by the underlying language. Rather, such information are passed as parameters, which must be appropriately given by some calling convention which must be obeyed by the application. Also, the underlying language might not provide sufficient type information to the RPC system, as the types in the numerical RPC are somewhat 'finer grained', in that it must contain info such as the leading dimension of the array/matrix usage.

Moreover, both Ninf and Netsolve locate appropriate computing resources on the network without explicit specification, achieving load balancing and fault tolerance. Various info are contained in their specialized IDL language.

For example, in Ninf, if a client's C program involved calling a matrix multiply routine:

```
#define N 1000
double A[N][N], B[N][N], C[N][N];

matrix_multiply(N, A, B, C);
// C = A * B
```

The procedure call in the last line is merely replaced with:

```
Ninf_call("matrix_multiply",N,A,B,C);
```

Here, notice that:

1. Only one generic, polymorphic call (`Ninf_call()`) given as the basic API for RPC. This not only allows simple one-line substitution of call sites to make the call be remote, but also allows the clients *not* to maintain stub libraries or IDLs (see below).

2. The size of the matrix is given by `N`, but C (nor Fortran) has no inherent ways to determine the size of an array data structure, let alone which parts of the matrix is used for more intricate algorithms.

3. Both C and Fortran assumes that array arguments `A`, `B`, and `C` are passed by reference. Moreover, there is no way to tell from the program that C is being used as an output, while A and B are inputs to the procedure.

4. The first argument of the call is the signature of the library or an application being called. For Ninf, unless a URL is specified to target a particular library on a host, the metaserver infrastructure selects an appropriate server from a set of servers under its maintenance, depending on monitored network throughput, server performance and load, and the availability of the library.

In order to facilitate such flexibility for program globalization and ease-of-maintenance on the client side, both Ninf and Netsolve provide specialized IDLs that embody sufficient information to implement the features described above. For example, for Ninf, the IDL for the matrix_multiply would look like the following:

```
/* Ninf interface */
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" /* link libs */
  /* lang.  and call seq.  */
  Calls "C" dmmul(n,A,B,C);
  Complexity n^3
  ...
```

Here, the IDL embodies the necessary information to describe the in/out values of the call, just as is with the CORBA IDL. Some additional information are present, such as the computational complexity of the call with its arguments. Moreover, the IDL compiler automatically deduces the dependencies of scalar parameters versus the array indices. The example IDL describes only the simple situation of shipping the entire matrix based on n; more complex descriptions such as leading dimension, stride, dependencies on linear combinations of multiple indices, etc. are supported.

In both Ninf and Netsolve, the client does not maintain *any* form of IDLs; rather, the client only contains a small IDL interpreter. When a call is made, the metaserver (Agent in the case of Netsolve) locates an appropriate server, and lets it connect to the client. Given the signature of the call, the server sends the compiled IDL to the client; the client IDL interpreter in turn uses it to marshal and demarshal the arguments in order to make the call. This is somewhat similar to the Dynamic Invocation Interface (DII) of CORBA, but is completely transparent on the client side, unlike DII.

For the underlying communication, Ninf and Netsolve are very similar. Both employ Sun XDR encoding to marshal arguments, and try to pack and send the numerical data in a packed manner as a bytestream. One difference is that Ninf packetizes data stream into 4KByte packets, whereas Netsolve does not.

In summary, Ninf and Netsolve are *asymmetrical* systems in that the client and server side software packages being different. Not only this is the matter of size, but also for

functionality, as well as the necessary of maintenance; the client side need not be updated to a limited degree even if there is some protocol change, thanks to the IDL info being uploaded dynamically. By contrast, CORBA and other typical RPC systems are *symmetrical*, in that the same software packages are used for both client and the server. While this provides better flexibility for example clients can become servers and vice versa, it could put the complexity of the management not only on the server but also on the client side.

2.1. Systems Used for Benchmarking

Here are the systems used for benchmarking, consisting of Ninf, Netsolve and two representative CORBA systems:

- Ninf: ver. 2.0 — The latest release of the Ninf system available from `http://ninf.etl.go.jp/`.
- NetSolve: ver. 1.2 — The latest release of the Netsolve system available from `http://www.cs.utk.edu/netsolve/`.
- Ninf-on-Globus — A prototype implementation of Ninf ported to run on top of Globus. More specifically, the only changes made so far have been to change the communication layer from TCP/IP to Nexus; other features of Globus are yet to be utilized. Still, this serves as a good vehicle for testing the communication performance as well as how well Globus interfaces to existing global computing systems.
- ACE: 5.0 + TAO: 1.0 — The CORBA system under development led by Doug Schmidt at Washington University [13]. TAO is reputed for its high performance[4] and various additional features such as real-time. Although originally developed in the academia, ACE/TAO has seen production use in several projects. ACE/TAO is freely available for ftp from `http://www.cs.wustl.edu/~schmidt/TAO.html`
- OmniORB: ver. 2.7.1 — OmniORB is another representative high-performance ORB in wide use. It is also freely available at `http://www.uk.research.att.com/omniORB/omniORB.html`, including raw performance numbers.

3. The Benchmark Global Computing Applications

Here we briefly discuss the benchmark applications. One is the Linpack numerical library, which we have employed as a test case for our previous benchmarking work on Ninf[17]. The other is the *SDPA* application, which is a full-fledged numerical application used in advanced operations research. The former incurs substantial communication and is predictable, representing worse (if not worst) case scenario in terms of data transfer. The latter is representative of a typical global computing application where data size is significant, but computation itself dominates the computation time.

3.1. Overview of Linpack

For double precision Linpack, we execute the LU-decomposition(dgefa) and backward substitution (dgesl) remotely. The overall execution time of Ninf_call T_{Ninf_call} consists of communication time T_{comm} + computing time T_{comp}. Given a Linpack matrix size n, they are:

$$T_{comm} = T_{comm0} + \frac{8n^2 + 20n}{B} \quad (1)$$

$$T_{comp} = T_{comp0} + \frac{2/3n^3 + 2n^2}{P_{calc}(n)}. \quad (2)$$

where T_{comm0} and T_{comp0} are the setup times for communication and computation, respectively, B is the client-server communication throughput, and $P_{calc}(n)$ is the local Linpack performance on the server machine. Then, Ninf_call performance P_{Ninf_call} can be completely predicted as follows:

$$P_{Ninf\_call} = \frac{2/3n^3 + 2n^2}{T_{Ninf\_call}}. \quad (3)$$

Because T_{comm} is $O(n^2)$ while T_{comp} is $O(n^3)$, it becomes computation dominant as n becomes larger. As a result, we can expect that P_{Ninf_call} will exceed the performance on the client machine given sufficient $P_{calc}(n)$.

3.2. Overview of SDPA

SDPA (Semidefinite Programming Application)[9] is an application to solve semidefinite programming problems, which is an advanced form of optimization problems beyond linear programming in operations research. The central algorithm is a form of interior-point method, allowing fast solutions to very large problems not possible with previous methods. SDPA is used for various optimization problems such as architectural optimization in buildings, control-theoretic optimizations, and solving large graph theory problems. SDPA is being developed by a group led by Prof. Masakazu Kojima at Tokyo Institute of Technology, independent of Ninf, and has set several world records in solving combinatorial problems. A standalone version is available for anonymous ftp at `ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA/`.

SDPA is designed as a standalone application; thus, it reads the array describing the problem from a file described with a certain format, computes, and stores the result into an output file. Although this is a typical behavior of a standalone program, neither the original Ninf nor Netsolve had the facility to directly cope with such form of input/output, as the original intent was to "ninfy" libraries as opposed to applications. Moreover, the application reads/writes to/from files at multiple, arbitrary places within the application; thus, there is no single point of entry for data passing as is with library procedures.

We have attempted two ways to solve the problem. First, the new versions of Ninf and Netsolve support remote file pointer abstractions. As long as all file I/O locations are identified, this allows relatively systematic and simple change to the application to read/write with a remote file. As a second solution, we wrote a wrapper library, which transfers the necessary data into temporary files on the server. The application is then invoked with filenames of these temporary files. The results are returned in a similar manner. Although this involves no change to the application, it will incur extra overhead of entire file transfer. Moreover, allowing such uncontrolled potentially persistent data on the server might not be desirable. For CORBA, we only used the wrapper solution.

For both solutions, for LAN, computation dominates the communication time, even for relatively small problems. For WAN setting, this is not necessary the case.

4. Characteristics Comparisons of Ninf, Netsolve and CORBA for Global Computing Applications

For global computing to become ubiquitous, we believe there are several requirements that could be termed as "ease-of-use". To be more technically concrete, we broke this down into three characteristics of pertaining to the usage of each system. We discuss the observed qualitative and quantitative characteristics of each, both from the client and the server sides:

1. Ease of programming: what are the learning and program modification efforts by the client and the server programmers to "globalize" a library or an application? (1C and 1S)
2. Ease of installation: How much footprint do installations require? How easy is it to install the system? (2CS)
3. Ease of maintenance and reliability: Is the system easy to maintain and update? Are the systems robust to failure, especially in global setting? (3C and 3S)

(1C) Ease of programming (client side)

In order for RPC-style global computing systems to become widespread, it is essential that the client-side programming be very simple. This applies to both development of new programs as well as modifying existing programs for delegating parts of their execution to remote libraries. For both Ninf and Netsolve, invocation of remote computing service is almost identical to local procedure calls, requiring essentially a cosmetic change to the call itself (in fact, Ninf and Netsolve themselves are almost identical, as such it is trivial to port one client application to another). More concretely, for each instance of local procedure call

```
func(arg1, arg2, ...);
```

that the client wants to invoke remotely, all he needs to do is to change it syntactically to:

```
Ninf_call("func",arg1,arg2, ...);
```
(Ninf)

```
netsl("func()",arg1,arg2, ...);
```
(Netsolve)

Moreover, the client does not need to maintain any of the stub code for the RPC; all he needs to do is to link his application with the generic Ninf or Netsolve library, which do not need to be changed for different remote libraries being called [1]. The complexity of the remote call is effectively encapsulated on the server side.

Contrastingly, for CORBA, the complexity of RPC call is not encapsulated on the server side, nor in the IDL of the call. One has to understand the various CORBA types defined for each language binding. There has to be various declarations, pre/post processings, and user-level type conversions made before a call can be made. Moreover, the defined data types for differ for each CORBA system; thus, not only that the client program has to be modified considerably, but also a client code is not portable across CORBA systems[2]

More concretely, for CORBA, the client user has to generate the stub code from the latest IDL, and then must write his code in accordance with the code structure of the stub. Below is an outline of this in TAO:

```
#include ''LinpackC.h''
CORBA::Long lda, n;
CORBA::Environment env;
CORBA::ORB_var orb =
  CORBA::ORB_init(argc,argv,0,env);
CORBA::Object_var object =
  orb->string_to_object(ior,env);
Linpack_var s =
  Linpack::_narrow(object.in(),env);
s -> linpack(a,lda,n,ipvt,b,info,env);
```

Below are the number of lines that the client had to dedicate to make a remote call for each of the benchmarks. Note that this is still a very simple case; in cases where numerous remote calls are made from within the application, the effects will be much more substantial.

(1S) Ease of programming (server side).

The efforts required for 'ninfying' a library or an application is similar. For libraries, both Ninf and Netsolve require no modification to the library code itself, as long as the library employs the standard data types (i.e., scalars and

[1] One caveat for such genericity is that static type checking of the arguments are not directly available. Thus, if the user makes a mistake in the type or the arity of the arguments, the result is a dynamic error, or could even crash the client. Fortunately, this could be circumvented in languages such as C, where macros and function prototype declarations are available; the technique is to essentially create a header file containing "wrapper" that defines a function which delegates the call to the generic `Ninf_call`.

[2] One could argue that with IIOP, a client of one CORBA system can be connected to a server of another CORBA system. While this is true and works in practice, most high-performance ORBs support private high-performance protocols if the client and the server are of the same system, bypassing IIOP. The impact of performance loss by the use of IIOP is discussed in Section 5.

	Linpack	SDPA
Ninf	1	1
NetSolve	1	1
TAO	14	42

Table 1. Number of lines for making a remote call (client)

arrays); rather, all the information regarding the call is written in the IDL, and the generated skeleton code is merely linked with the library code. For CORBA, one has to write not only the IDL, but what effectively is a glue wrapper code following the structure of the generated skeleton code from the IDL. Moreover, as was mentioned, CORBA IDL does not support features essential in numerical matrix shipping such as leading dimension, strides, dependencies between the arguments, etc. Thus, one has to program such special features in an explicit way in the wrapper code.

In comparing the IDLs, Ninf resembles ANSI C and is relatively simple to write. CORBA is similarly simple, but lacks the ability to describe some numerical features as described above. Netsolve IDL is a little more complex, however gives finer-grained control to the user. Figures 1 and 2 are extracts of example IDLs for Linpack. In practice, the size of Ninf IDL is 247 bytes, Netsolve is 711 bytes. CORBA is as small but when one includes the necessary stub code the user has to write, the total size increases to 3044 bytes.

```
Module linpack;

Library "linpack_flib.a";
FortranFormat "%s_";

Define linpack (IN double a[lda:n][n], IN int lda,
                IN int n, OUT int ipvt[n],
                INOUT double b[n], OUT int *info)
CalcOrder n^3
Calls "Fortran" linpack (a, lda, n, ipvt, b, info);
```

Figure 1. Ninf IDL

Figure 2. NetSolve IDL

(2CS) Ease of installation (client and server)

The system must be easily installable—this is especially important on the client, since the client user usually will install the client on his desktop machine, just as he would a web browser or a telnet client. It is also important to some degree on the server side for obvious reasons. Since such characteristics are difficult to quantify, we compare the size of the installed package, plus the time required for the installation.

Table 2 shows the (a) maximum amount of installation space required, including source, compiled binaries, documents, demos etc., and (b) the amount of binaries and libraries required for operation. We see that TAO is substantially large due to the numerous robust features it provides for general distributed computing as well as different language bindings. This is understandable, as CORBA must meet the needs of variety of distributed computing needs, but still is substantial, especially for the dedicated purpose of numerical global computing. OmniORB is seemingly small in that regard; in fact the operational size of the server is smaller than Ninf or Netsolve.

Still, for both Ninf and Netsolve, the distinguishing factor from both CORBA implementations is the small operational sizes of the client. This is primarily due to fact that:

- Ninf and Netsolve are asymmetrical systems in that the client and server are separate software packages, with much of the complexity (such as IDL management) residing on the server side and the metaserver (Agent in Netsolve), and

- Ninf nor Netsolve is more specific to numerical global computing, and does not support all the robust features for distributed of CORBA, such as IIOP interoperability, object support including interface inheritance, etc. It remains to be seen if Ninf or Netsolve provide the sufficient features for most likely application scenarios for RPC-style global computing.

	Client Install	Client Opr	Server Install	Server Opr
Ninf	2.0 MB	.56 MB	11 MB	5.6 MB
NetSolve	5.0 MB	.95MB	20 MB	16 MB
TAO	700 MB	37 MB	700 MB	37MB
OmniORB	30 MB	2.8 MB	30 MB	2.8 MB

Table 2. Installation Sizes of Each System

By all means installation size is not the only parameter to indicate the ease of installation. Larger installation size could mean more features, which could especially be important for generic tools such as CORBA. Still, we stress the importance of the client-side especially being lightweight. We do note that the system complexity will increase as code size grows. This complexity will burden the programmer side if it is to be hidden from the user (consider the complexity of Windows installer). Instead, by simplifying what is to be loaded especially on the client side, the entire system will become manageable, more robust, and thus possi-

bly more widely used. [3]

(3C) Ease of maintenance and reliability (client side)

The small size of the client realized by the lightweight client has also another advantage over symmetrical systems. As described earlier, both Ninf and Netsolve has a very lightweight client supporting generic remote calls, allowing IDLs to be maintained solely on the server side. Thus, the client never needs to update his client package, unless drastic changes are made to the system. Even if the API of a particular call is changed, or even if some protocol elements are added or modified such that the IDL compiler generates the stubs differently, the client does not have to update the client package. In CORBA, the IDLs must be shipped to all the clients, and the IDL compiler be run, possibly the client wrapper code for stubs rewritten, everything re-linked etc. One could alternatively use the DII interface in CORBA, but its usage is rather kludgey.

This characteristics is in principle similar to Java Jini[7][4], and is of effective advantage for global-scale distributed computing. In such a setting, to update all the peers in the system in sync would be very difficult, if not impossible endeavor. Rather, what modifiable parts of the system should be dynamically updated on demand . Jini allows this by downloading all the protocol code dynamically upon establishment of a connection. Ninf and Netsolve are little more conservative, in that the downloaded code is not a Turing-complete language, but rather small languages specially tailored for argument marshaling for numerical computing. This is a tradeoff issue; while in Jini one can alter the protocol completely, one must have the complete Java runtime environment underneath, including the standard class libraries. Ninf and Netsolve allow for a smaller client, at the expense of less degree of freedom for updates.

(3S) Ease of maintenance and reliability (server side)

Maintenance on the server side is similar for Ninf, Netsolve, and CORBA systems, but so far as we have seen, only Ninf allows registration of new libraries *while the server is active*. For both Netsolve and CORBA systems we have studied, one has to bring down the server. Such live registration is a desirable property where the client might be anonymous, and thus it could be difficult to notify everyone of the change. It is unlikely that this is a fundamental restriction in Netsolve or CORBA systems, but this is one useful feature one should keep in mind when designing systems where the users may be anonymous.

[3] For fairness, we do also note that some operating systems or popular middleware employ CORBA as an underlying object system, just as is with Microsoft Windows embody DCOM. For example, Linux GNOME desktop environment employs the Orbit CORBA system as the underlying ORB. It remains to be seen if a particular CORBA system becomes a ubiquitous system infrastructure.

[4] In fact, both Ninf and Netsolve predate the development of Jini

5. Performance Evaluation

We now compare the performance of Ninf, Netsolve and the CORBA systems for the benchmark applications in both LAN/WAN settings.

5.1. Evaluation Environment

We employed the Ultra 60 (UltraSparc2 300MHz x2, 256MB) as the test server located at Tokyo Institute of Technology (TITECH)in Tokyo, Japan. The client was an Ultra2 (UltraSparc2 200Mhz x2,256MB) interconnected by a 100 Base-T network for the LAN setting. For the WAN setting, the client SparcStation5 (MicroSparc2 85Mhz, 32MB) was located at the Electrotechnical Laboratory approximately 90 kilometers away, The average ftp throughput between the TITECH server and the ETL client was measured to be approximately 3 Mbps, largely stable irrespective of time of day. We did most of the benchmark runs during nighttime over the course of two weeks, with some SDPA benchmarks requiring over an hour each.

5.2. Evaluation Methodologies

The two aforementioned applications have been employed in the evaluation. The time and communication complexity of Linpack have already been discussed. In the measurements, the performance is given in MFlops, which includes the communication time necessary to send the arguments and the results between the client and the server. For SDPA, since the computational complexity cannot be easily determined from data size, we took the standalone benchmark result of the application, and normalized the remote compute time with respect to the standalone time. Additionally, data transfer time was obtained from the average bandwidth, and subtracted from the total compute time when required.

We tested Ninf and Netsolve under default setting provided by the standard distribution, and also tuned them so as to attempt to achieve best performance. For Ninf, we turned off the metaserver for measurement stability. For both Ninf and Netsolve, multiple benchmarks were taken varying the following parameters:

· Ninf packet buffer size (Ninf only, default = 4KBytes),

· Kernel buffer size (For Ninf, in sync with Ninf packet buffer size. For Netsolve, only kernel buffer size),

· The `NODELAY` option on/off (to attempt to disable the Nagel algorithm),

· For SDPA, versions which utilize the remote filepointer and versions which do not.

Unless otherwise noted, the results indicate the best performance obtained.

5.3. Evaluation Results

5.3.1. Linpack Results

We first illustrate the performance results for Linpack in the LAN setting. Figure 3 shows the effective measured performance in MFLOPS, and Figure 4 shows the measured

communication throughput. The X-axis of the graphs show the size n of the $n \times n$ matrix.

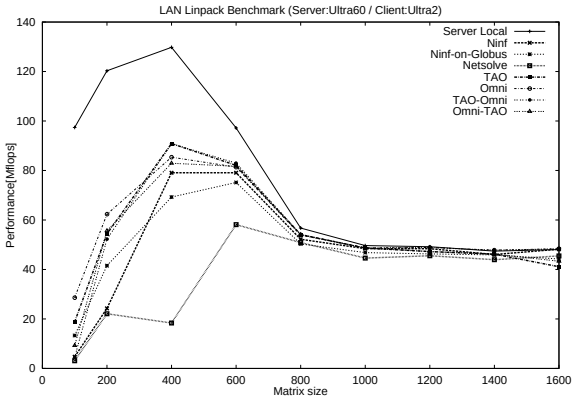

Figure 3. Effective Linpack Performance in LAN Setting

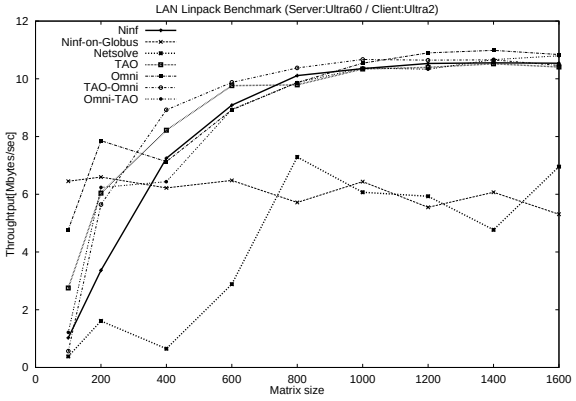

Figure 4. Linpack LAN Communication Throughput

In general for LAN, performance differences between each system is quite small for larger problem sizes. This is understandable since compute time will dominate in that situation.

For smaller problem sizes, Ninf seems to be slightly disadvantaged over the ORB systems that are performing well, even for IIOP communications both ways (TAO client + OmniORB server versus OmniORB client + TAO server). Ninf-on-Globus performs slightly less than Ninf or ORBs, and Netsolve does not perform well. The reason becomes apparent when we observe the communication throughput in Figure 4; Ninf and both ORBs perform almost identically, achieving maximum throughput for 100Base-T of 10MBytes/sec around problem size 1000. Ninf is slightly slower on smaller problem sizes. By contrast, Ninf-on-Globus and Netsolve only achieve the peak of 6MBytes/sec. The reason the throughput is poor for Netsolve for small problem sizes is mostly attributed to the overhead of Agent, which cannot be bypassed.

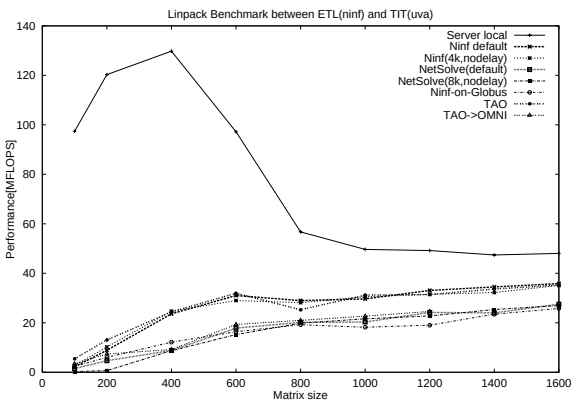

Figure 5. Effective Linpack Performance in WAN Setting

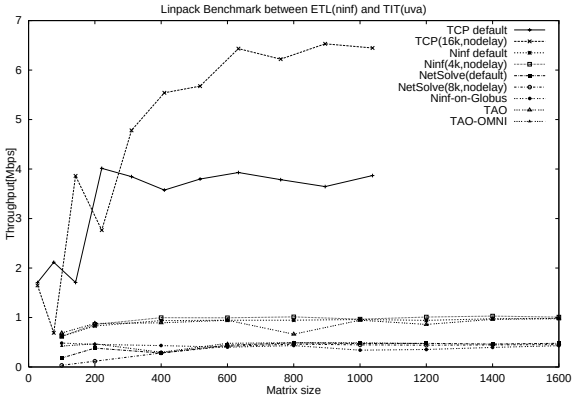

Figure 6. Linpack WAN Communication Throughput

We next show the results for Linpack in the WAN setting. Figure 5.3.1 shows the effective measured performance in MFLOPS, and 6 as well as 7 (magnified) show the measured communication throughput.

In general, because of the slower link and higher latency, communication becomes more dominant, affecting the performance in greater degree compared to the LAN setting. Thus for both Ninf and Netsolve under WAN setting (also for LAN), tuning became crucial. We could not tune the CORBA implementations, which is the subject of further investigation.

For Ninf, the default 4K buffer sizes yield the best performance and throughput compared to larger sizes. The addition of `NODELAY` increased performance slightly. For Netsolve, the default setting without `NODELAY` proved to be the best. Figure 8 shows how the change in the buffer size alters performance for Ninf.

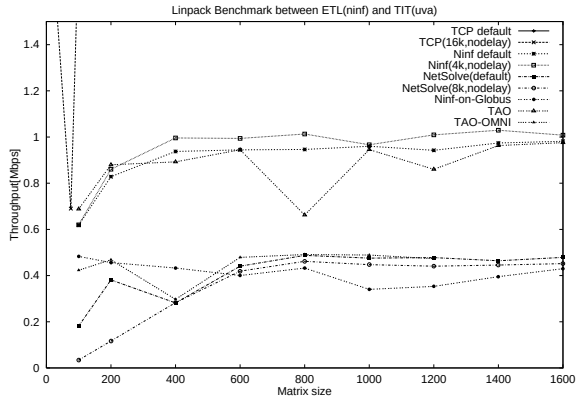

Figure 7. Linpack WAN Communication Throughput (Magnified)

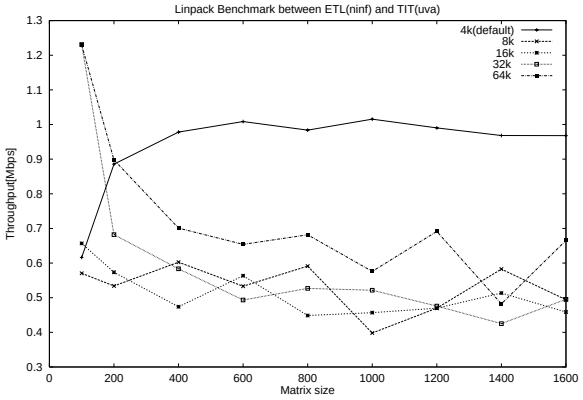

Figure 8. Ninf Kernel Buffer Size Modifications and Performance

TAO closely matches the best Ninf (with NODELAY). For TAO client → TAO server communication, TAO uses a private, high-performance communication protocol, which seems to match the efficiency of dedicated Ninf protocols for sending numerical sequences. Unlike the LAN setting, The TAO client → Omni server[5] using IIOP was slower. This resulted in slower overall performance, even for larger problem sizes where computation becomes dominant. Graph indicates that even for size 1600, the performance is impacted by approximately 20%.

Netsolve performed consistently less than Ninf or TAO, being about the same as the IIOP case. In fact, this was the case for the optimal case (which was the default) we have been able to obtain with the tuning above. This is somewhat mysterious to both us and the Netsolve team, since the capabilities, code structure, and the protocols of both systems are similar, and we have not been able to pinpoint the case.

[5]We could not perform Omni-to-omni benchmark in WAN setting due to unresolvable compilation error on the ETL end; we expect to fix this problem and by the time of publication.

We are conducting some additional tests with Netsolve. So far we have found that modifying the lower-level networking code of Netsolve to packetize data in the same manner as Ninf does not help performance. In fact, we have reasons to believe that the wire transfer time is almost the same for both systems, and the speed difference could be attributed to some other, unknown overhead. In fact some of the more recent experimental versions of Ninf have exhibited similar performance slowdown. As such, we are still investigating the exact cause of the discrepancy between the performance of Ninf vs. Netsolve, and some of the more recent (and more complex) experimental versions of Ninf. Such an investigation we believe, could be generally beneficial for higher performance for all systems.

In contrast to LAN, when we observe the bandwidth against raw TCP throughput, the best tuned result (16KByte kernel buffer size + NODELAY) which achieves approximately 6.5Mbps, we see that both Ninf and TAO barely achieves 1 Mbps. This could be due to various reasons, such as marshaling cost, non-optimal use of TCP read/write versus kernel buffer, etc., but we have not pinpointed the problem. One hint is the discrepancy between the buffer sizes for optimal performance for raw throughput (16KBytes) versus Ninf (4KBytes). In fact, Ninf was significantly slower when the kernel buffer size was set to 16KB.

5.3.2. SDPA Results

We show the results for SDPA for LAN setting in Figures 9 and 10, and the WAN setting in Figures 11 and 12.

For all of the systems, execution time was almost equivalent, both for LAN and WAN, for very large problems requiring almost an hour run. The throughput graphs, however, are vastly different among the systems. For LAN, the Ninf system with the filepointer support is fastest, followed by the CORBA systems. Here, as was with Linpack, IIOP performed quite well, almost matching the private protocols. Ninf without the remote filepointer support is significantly slower than Ninf, or surprisingly, even Ninf-on-Globus. Netsolve performance was slower, hinting that wire transfer time is not the actual factor for performance difference. Here again the remote filepointer support excelled performance.

For WAN, communication different was much smaller. As is with Linpack, Ninf (with filepointer) and TAO formed the faster group, while Netsolve, TAO→OmniORB, and Ninf-on-Globus formed the slower group, but the throughput difference was less than a factor of two. The tuning parameters of Ninf and Netsolve were the same as Linpack. Figure 13 shows the effects of throughput tuning for Ninf and Netsolve.

6. Conclusion and Future Work

We performed side-by-side comparisons of RPC-based numerical global computing systems, Ninf and Netsolve, against existing high-performance CORBA systems, for two types of global computing applications, Linpack which

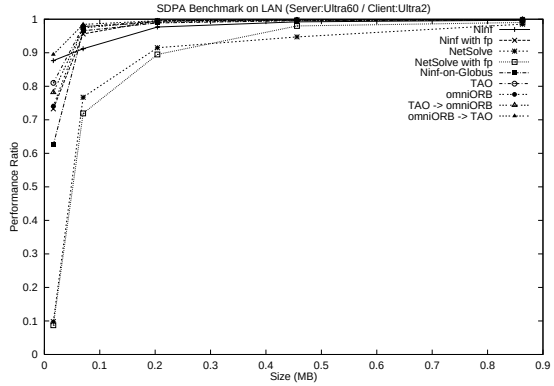

Figure 9. Effective SDPA Performance in LAN Setting

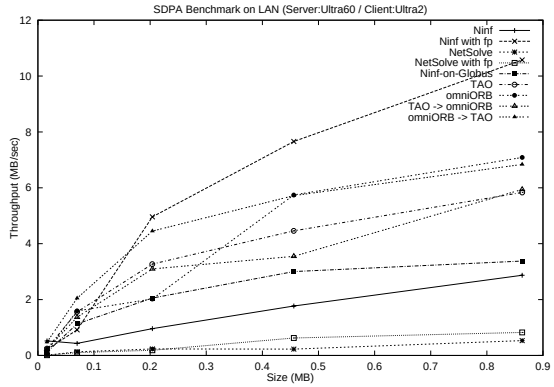

Figure 10. SDPA LAN Communication Throughput

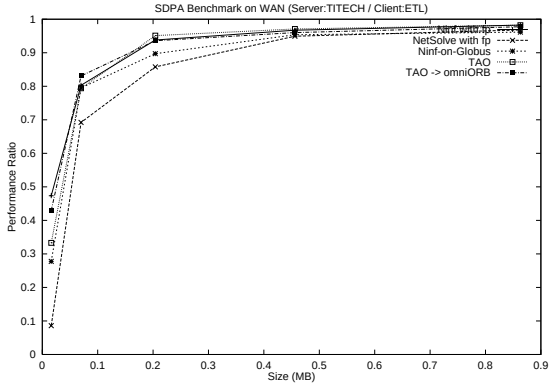

Figure 11. Effective SDPA Performance in WAN Setting

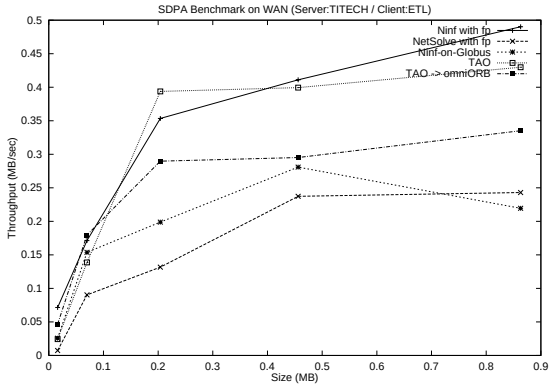

Figure 12. SDPA WAN Communication Throughput

is representative of a typical network library, and SDPA, which is representative of a standalone, long-running application to be "globalized". This is the first such research to be done, and despite that there are much more comparisons that could be performed, we feel that we have identified some key issues relevant to the advantage of developing dedicated global computing systems.

Qualitative as well as quantitative comparisons of the system characteristics revealed advantages of Ninf and Netsolve over CORBA systems. This is primarily due to the fact that Ninf and Netsolve are tailored for numerical global computing, whereas CORBA is a more general-purpose RPC system. Some salient features of Ninf and Netsolve includes support for globalizing typical numerical RPCs, dynamic IDL management, and asymmetry of the systems. This resulted in both Ninf and Netsolve being easier system to program and maintain, for the applications we have looked at.

Performance measurements of the systems revealed that, for larger problems, the overall application performance was not very different between the systems. This is primarily due to the fact that, for larger problems sizes, the application became computation dominant. For WAN setting, however, there was up to 20% performance difference for Linpack.

Also, although Ninf as well as the CORBA systems performed well in LAN setting, for WAN setting there was significant headroom for throughput improvement. This is despite considerable tuning effort attempted, such as modification of the kernel buffer size, `NODELAY` specification, etc. Netsolve was consistently slower, and we are conducting further investigation of the cause.

IIOP performance was surprisingly competitive for the high-performance CORBA systems we have tested. Although this may not hold for all CORBA systems, nevertheless this is an additional benefit for CORBA, since high interoperability does not come at the expense of high performance. Thus, by all means CORBA should not be written off, but certainly it could benefit from the characteristic advantages of dedicated systems such as Ninf and Netsolve.

For future work, we would definitely like to increase the number and the types of applications subject to benchmarking. In particular, an application that calls multiple remote

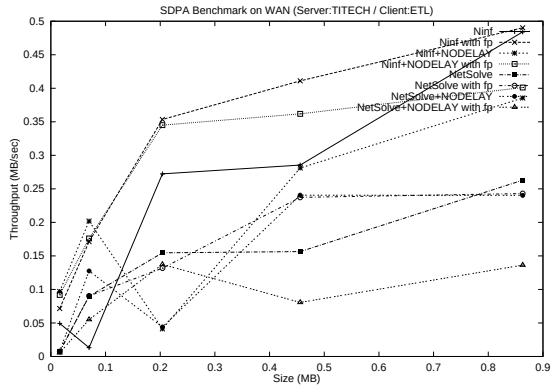

Figure 13. SDPA Throughput Tuning in WAN Setting for Ninf and Netsolve

libraries in an asynchronous ways, or those that employ more complex data types, are highly desirable. Also, other aspects of global computing systems should be covered to qualify the advantages of broader global computing platforms such as Globus, Legion, AppLes, Condor, etc.

6.1. Acknowledgment

Numerous acknowledgments are in line for this work. Firstly, highest appreciations to Prof. Kojima and Dr. Funjisawa for helping us to "ninfy" the SDPA. Secondly, many thanks to Henri Casanova, Rich Wolski and Fran Berman and their groups for discussions and investigations into network throughput performances and differences in performance between Ninf and Netsolve. Appreciations also go to TAO and OmniORB groups for making their high-performance ORBs publically available, with very good documentations allowing us to successfully deploy them in WAN setting. Finally thanks goes to other members of the Ninf project group for their discussions and continued support.

References

[1] Kevin Butler, Mark Clement, and Quinn Snell. A performance broker for CORBA. In *Proc. of HPDC8*, pages 19–26, 1999.

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

[3] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.

[4] Andy Gokhale and Douglas Schmidt. Evaluating CORBA latency and scalability over high-speed atm networks. In *IEEE 17th International Conference on Distributed Systems (ICDCS 97), May*, pages 27–30, 1997.

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

[6] Yuji IMAI, Toshiaki SAEKI, Tooru ISHIZAKI, and Mitsuhiro KISHIMOTO. Crisporb: High performance CORBA for system area network. In *Proc. of HPDC8*, pages 11–18, 1999.

[7] Jini. http://www.sun.com/jini/.

[8] Katarzyna Keahey and Dennis Gannon. Pardis: CORBA-based architecture for application-level parallel distributed computation. In *Supercomputing '97*, 1997.

[9] M. Kojima, M. Shida, and S. Shindoh. Search Directions in the SDP and the Monotone SDLCP: Generalization and Inexact Computation. *Mathematical Programming*, 85(1):51–80, May 1997.

[10] Hidemoto Nakada, Hiromitsu Takagi, Satoshi Matsuoka, Umpei Nagashima, Mitsuhisa Sato, and Satoshi Sekiguchi. Utilizing the metaserver architecture in the ninf global computing system. In *High-Performance Computing and Networking '98, LNCS 1401*, pages 607–616, 1998.

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

[12] omniORB. http://www.uk.research.att.com/ omniORB/omniORB.html.

[13] TAO (The ACE ORB). http://www.cs.wustl.edu/~ schmidt/TAO.html.

[14] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc. of HPDC-7*, 1998.

[15] Christophe René and Thierry Priol. Mpi code encapsulating using parallel CORBA object. In *Proc. of HPDC8*, pages 3–10, 1999.

[16] Satoshi Sekiguchi, Mitsuhisa Sato, Hidemoto Nakada, and Umpei Nagashima. – Ninf – : Network base information library for globally high performance computing. In *Proceedings of Parallel Object-Oriented Methods and Applications (POOMA)*, Feb. 1996. http://ninf.etl.go.jp/.

[17] 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.