Multi-client LAN/WAN Performance Analysis of Ninf: a High-Performance Global Computing System

Atsuko Takefusa
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku, Tokyo 112, Japan
takefusa@is.ocha.ac.jp
http://www.is.ocha.ac.jp/~takefusa/

Satoshi Matsuoka
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo 152, Japan
matsu@is.titech.ac.jp
http://matsulab.is.titech.ac.jp/~matsu/

Hirotaka Ogawa
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan
ogawa@ipl.t.u-tokyo.ac.jp
http://www.ipl.t.u-tokyo.ac.jp/~ogawa/

Hidemoto Nakada
Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan
nakada@etl.go.jp
http://www.etl.go.jp/~nakada/

Hiromitsu Takagi
Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya 466, Japan
takagi@center.nitech.ac.jp
http://www.center.nitech.ac.jp/~takagi/

Mitsuhisa Sato
Real World Computing Partnarship
Tsukuba Mitsui Blg. 16F, 1-6-1 Takezono, Tsukuba, Ibaraki 305, Japan
msato@trc.rwcp.or.jp
http://www.rwcp.or.jp/perple/msato/

Satoshi Sekiguchi
Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan
sekiguchi@etl.go.jp
http://phase.etl.go.jp/people/sekiguchi/

Umpei Nagashima
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku, Tokyo 112, Japan
umpei@hn.is.ocha.ac.jp
http://sap18.is.ocha.ac.jp/

Abstract:

Rapid increase in speed and availability of network of supercomputers is making high-performance global computing possible, including our *Ninf* system. However, critical issues regarding system performance characteristics in global computing have been little investigated, especially under multi-client, multi-site WAN settings. In order to investigate the feasibility of Ninf and similar systems, we conducted benchmarks under various LAN and WAN environments, and observed the following results: 1) Given sufficient communication bandwidth, Ninf performance quickly overtakes client local performance, 2) current supercomputers are sufficient platforms for supporting Ninf and similar systems in terms of performance and OS fault resiliency, 3) for a vector-parallel machine (Cray J90), employing optimized data-parallel library is a better choice compared to conventional task-parallel execution employed for non-numerical data servers, 4) computationally intensive

tasks such as EP can readily be supported under the current Ninf infrastructure, and 5) for communication-intensive applications such as Linpack, server CPU utilization dominates LAN performance, while communication bandwidth dominates WAN performance, and furthermore, aggregate bandwidth could be sustained for multiple clients located at different Internet sites; as a result, distribution of multiple tasks to computing servers on different networks would be essential for achieving higher client-observed performance. Our results are not necessarily restricted to the Ninf system, but rather, would be applicable to other similar global computing systems.

Keywords:

Global network computing, Performance evaluation

1 Introduction

Rapid increase in speed and availability of network of supercomputers is now making so-called high-performance `global computing' possible, in which computational and data resources in the network are collectively employed to solve large-scale problems. There have been several recent proposals of global computing infrastructures, including our Ninf (Network Infrastructure for global computing) [1, 2] system, which makes available multiple remote computing and database servers on the network, allowing clients to semi-transparently call remote numerical applications and libraries from languages such as Fortran, C, and Java. Multiple calls in a network are coordinated by *metaservers*, which keep track of server load/availability, network bandwidth, etc. WWW-based interfaces are also available for easy user access of remote computing and data.

On the other hand, we have so far found that much of system performance characteristics and criteria for enabling the feasibility global computing have not been well-investigated. In order for global computing to succeed, we need to investigate and clarify various characteristics of the technology that are main constituents of a global computing system:

Communication Performance:

Global computing requires large amounts of data to be shipped over the network. Although available bandwidth is rapidly increasing, no quantitative measures are provided to judge its sufficiency. Also, since latency will not improve drastically in the future, it is not clear whether high latency would not be a deterrent to attaining performance.

Computing Server Selection and Performance:

Computation will be typically performed on high-performance supercomputing servers such as vector supercomputers and MPPs. On the other hand, they will be shared amongst multiple remote clients from different places in the global network. It is not clear what criteria should be employed in selecting such supercomputing servers, and whether the current supercomputers/MPPs would be appropriate for such usage.

Sharing by Multiple Clients:

Computing servers will be shared by arbitrary multiple clients on the network. Thus, it must also be shown that the combination of the Ninf computing server (and those in similar systems), supercomputers, and their operating systems, could handle the concentration of requests from multiple clients in a graceful manner, under both LAN and WAN environments. Since we rely on existing supercomputers, it is not apparent whether their operating systems could handle multiple computational requests from clients, whose transactions will tend to be longer duration compared to typical short database query transactions. Also, it must be shown that the system is resilient to various faults that could occur in network computing.

Remote Library Design and Reuse:

When executing remote libraries registered on Ninf computing servers, there is an option as to whether to distribute the computing resources amongst different client requests in a *task parallel manner*, or to allocate all the processors to each client task in a *data parallel manner* in sequence. This directly influences the design of the libraries, whether the client library has to be prepared specifically for Ninf, or the best (parallel) library for the machine could be directly reused and utilized.

We report on the benchmark results of Ninf computing server performance under various LAN and WAN environments to investigate the feasibility of global computing. As a representative of typical application core, we have selected the Linpack benchmark which involves significant amount of data transfer compared to its problem size, and NASPar EP benchmark application, in which communication is minimal. We have conducted the benchmarks in LAN and WAN environments with various combinations of client and server machines. We have also tested and compared multi-client cases in which the clients are located in the same LAN, on multiple nodes in the same remote network, and multiple clients located on a different network. As a result, we have observed the following results:

- We have observed that, with higher network bandwidth, Ninf execution becomes faster than local execution with a relatively small problem size for Linpack and obviously for EP. Here, latency was not a significant issue due to larger grain size of the problem; thus, given sufficient WAN bandwidth, global computing in the wide-area will be quite feasible in the future for a wide class of computational-intensive problems.

- By employing parallel vector machines (Cray J-90, 4PE) and large SMPs (Sun SPARC SMP), we have confirmed that operating systems for such machines are sufficient both functionally and performance-wise for supporting multiple clients over both WAN and LAN. Thus, we claim that, Ninf (and other global computing platforms) could readily be built on existing platforms.

- For J90, data-parallel execution using optimized libraries would be faster for lightly-loaded cases, and would exhibit almost identical performance to task-parallel execution under heavily-loaded cases. Thus, using optimizing libraries maximizing the performance of parallel machines would be appropriate under both LAN and WAN situations. On the other hand, for SMPs, preliminary results show that highly-multithreaded libraries exhibit slowdown in performance in multi-client situations due to thread-switching overhead, and there is a need for determining the optimal number of threads versus the number of clients.

- For compute intensive applications such as EP with little communication, LAN vs. WAN performance would be essentially equivalent, achieving high computing server utilization. Such applications are numerous, such as parallel rendering/imaging, and parameter sensitivity analysis, and they could readily benefit from Ninf and other global computing systems.

- For Linpack, which requires extensive communication, LAN performance was acceptable for practically-sized data sets. For WAN execution, performance will heavily depend on aggregate network throughput of the client, and not on server load, which remained light. Thus, under communication-dominant cases, task assignment and distribution should not be merely based on server load and utilization information, but rather on achievable network bandwidth.

2 Overview of Ninf System

The Ninf system is currently based on the client-server model. The client can make use of various computing library and database resources via server processes on supercomputers and database servers. Figure 1 illustrates the computational aspect of the Ninf system. Ninf users can easily utilize the supercomputing power from his desktop client over WAN without steep learning curves or major modifications to his applications.

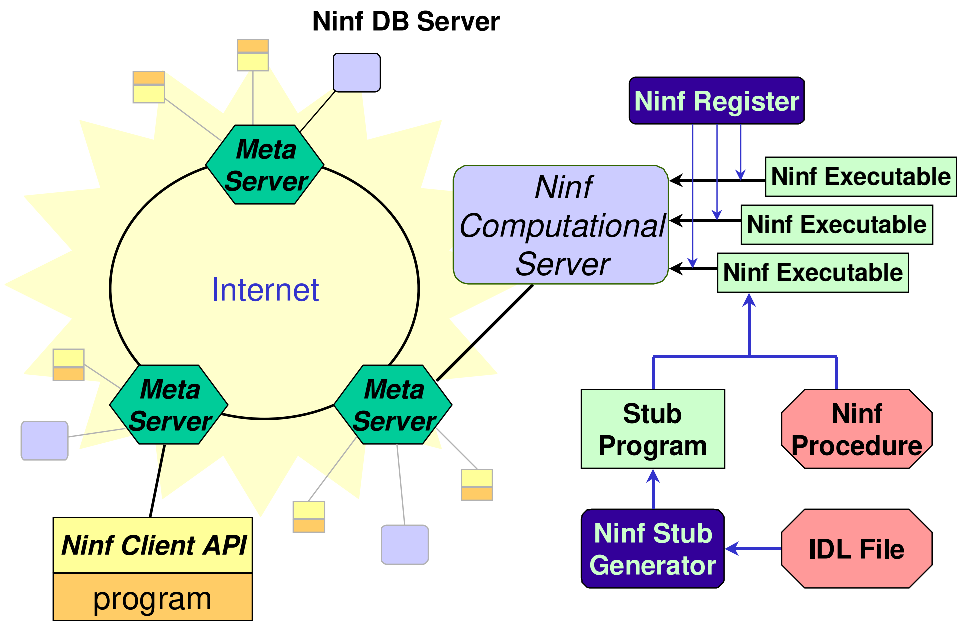

Figure 1: Computational Aspect of Ninf

The Ninf system consists of *Ninf computational and database servers*, *Ninf client API* which defines the client bindings, and *Ninf metaservers* which perform wide-area detection, allocation, and scheduling of resources over the network. The clients and various servers communicate via Ninf RPC (Remote Procedure Call) [3], which is tailored for the needs of high-performance numerical computing. There are also various supportive tools such as server registry tools, stub generators, query proxies, GUI builders, etc.

For the purposes of this paper, henceforth we will concentrate on describing the computational aspects of Ninf.

2.1 Ninf Computational Server

The Ninf computational server is a process which services remote computing requests of remote clients by managing the communication and activation of the services requested via Ninf RPC. Binaries of computing libraries and applications are registered on the server process as *Ninf executables*, which can be semi-automatically generated with IDL descriptions using the Ninf stub generator. The underlying transfer protocol is Sun XDR on TCP/IP, allowing easy porting on most major supercomputer platforms.

2.2 Ninf Client API

Ninf Client API is defined for major programming languages such as Fortran, C, C++, and Java. *Ninf_call* is a representative API used for invoking a named remote library on the server as if it were on a local machine via Ninf RPC. For example, one might call a local matrix multiply library as:

```
double A[n][n], B[n][n], C[n][n];
dmmul(n, A, B, C);
```

while with Ninf, one invokes the remote library with:

```
Ninf_call("dmmul", n, A, B, C);
// or, Ninf_call("http://.../dmmul",...
```

There are also APIs for asynchronous RPCs (*Ninf_call_async*), specification of transactions for parallel execution (*Ninf_transaction_begin*, *Ninf_transaction_end*), and queries of numerical database (*Ninf_query*, etc.).

2.3 Ninf IDL

Ninf executables can be derived directly from existing numerical libraries such as LAPACK. Each subroutine in the library become callable via Ninf RPC by specifying their interface information with Ninf IDL (Interface Description Language). In the matrix multiply example, the IDL for the actual multiply routine on the server would be as follows:

```
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])
"dmmul is double precision matrix multiply",
Required "libxxx.o"
Calls "C" mmul(n,A,B,C);
```

Argument numbers, types, access modes (input/output), array size, are necessary IDL information. As matrices are usually passed in memory with call-by-reference for local libraries whereas they must be shipped back and forth across the network for *Ninf_call*, matrix size, region of usage, stride, etc. that are dependent on scalar input arguments are either automatically inferred from IDL information or could be directly specified by the IDL writer. There are other optional info such as necessary modules for compilation and linking, aliases, client callback functions, comments on functionality, etc.

We also note that, with Ninf RPC, no prior downloading of IDL or stub generation is necessary on the client side. Stub generation are done solely on the server side; the client programmer never sees or manipulates the IDL information. As a result, *Ninf_call*s can be freely made without any library linking or header file inclusion. This is achieved by two-stage RPC in *Ninf_call*; when the client calls the server, it returns the compiled IDL information as interpretable code to the client. *Ninf_call* then interprets the IDL code and marshalls the arguments.1

2.4 Ninf Metaserver

The Ninf metaserver monitors multiple Ninf computing servers on the network, and performs scheduling and load balancing of client requests. The client need not be aware (but could specify) the physical location of computing servers. Additionally, metaserver controls the parallel, fault-tolerant execution of multiple sequence of *Ninf_call*s. The block of code surrounded by *Ninf_transaction_begin* and *Ninf_transaction_end* are not executed immediately; rather, data-dependency graph of the *Ninf_call* arguments are dynamically created, and at the end of the code block, the metaserver schedules the computation to multiple computational servers accordingly.

3 Single Client Benchmarks

We first extensively benchmark Ninf single client performance on multiple client and server platforms. As a benchmark program, we employ Linpack and NASPar EP. The former requires extensive communication to ship dense matrices over the network, but becomes computation dominant for large matrices. The latter is computation dominant irrespective of its arguments, and is used for measuring the effectiveness of load balancing performed by the metaserver onto multiple Ninf servers.

3.1 Single Client Benchmarking Environment

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

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

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

where T_{comm0} and T_{comm0} 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 $P_{N\inf\_call}$ is as follows:

$$P_{N\inf\_call} = \frac{\frac{2}{3}n^3 + 2n^2}{T_{N\inf\_call}}$$

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 $P_{N\inf\_call}$ will exceed the performance on the client machine given sufficient $P_{calc}(n)$.

The client and server machines in the benchmark are shown in Figure 2. We also show the client-server combination in Table. Local indicates the local Linpack performance without Ninf.

As Linpack routines, we registered the sgetrf and sgetrs of the libSci library, which make use of all 4 processors on the J90. On other machines, we employed glub4 and gslv4[5] routines which employ blocking optimizations and could thus be executed efficiently on RISC-based workstations (and thus would be disadvantageous for Ninf). Matrix size (n) was altered from 100 to 1600

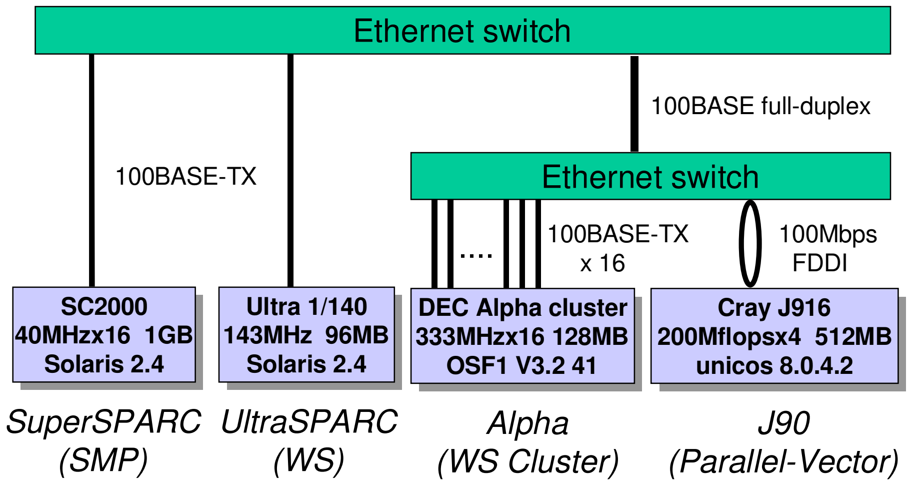

Figure 2: Client and server machines in the LAN benchmark

Client	Local	Remote: Ultra	Remote: Alpha	Remote: J90
SuperSPARC	o	o	o	o
UltraSPARC	o	-	o	o
Alpha	o	-	-	o

Table 1: Combination of clients and servers in the LAN benchmark

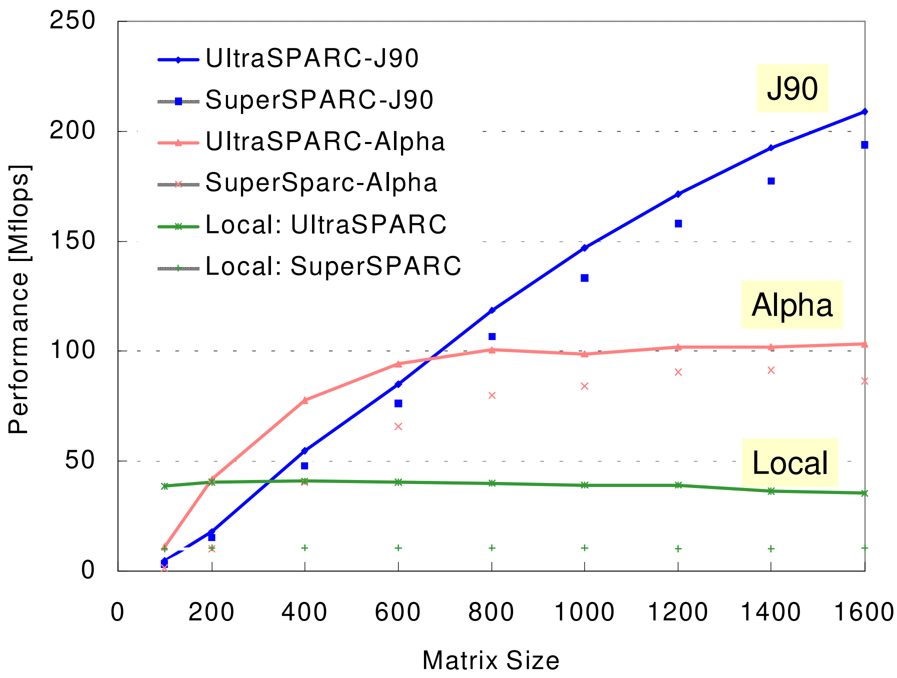

Figure 3: Ninf LAN Linpack Results with Single SPARC clients

3.2 Single Client Linpack Results

Figure 3 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 *Ninf_call* and Local in Mflops. The performance of Local remains relatively constant across n 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 $n = 200 \sim 400$.

Comparing UltraSPARC Local and SuperSPARC to UltraSPARC *Ninf_call*, the latter converges to UltraSPARC Local performance as n becomes larger. This is because $T_{comm} \ll T_{comp}$ for large n, hiding the communication overhead. The same is true for Alpha Local and Super/UltraSPARC to Alpha *Ninf_call*. For *Ninf_call*s to J90, J90's Local achieves 600Mflops when $n = 1600$, while *Ninf_call* exhibited similar performance even for both SPARC clients whose Local performance differs considerably.

Figure 4 shows the results for Alpha clients. Here, we have additionally measured the performance of standard Linpack routines without blocking optimizations, in order to simulate the situations in which the users not taking the extra effort of optimizing his local library routine. Although recent performance improvements of PC and WS are significant, it is difficult for a user to write an optimized program for large-scale problems.

Comparing Alpha Local and *Ninf_call* to J90, *Ninf_call* become faster at approximately $n = 800 \sim 1000$. On the other hand, when employing a standard, non-optimized routine on Alpha, *Ninf_call* became advantageous at approximately $n = 400 \sim 600$.

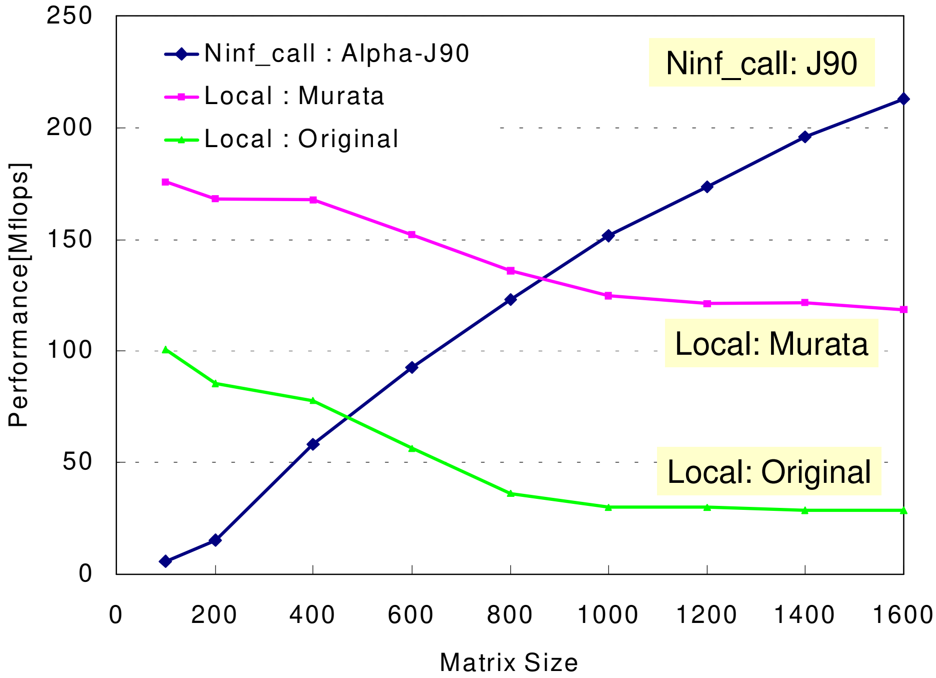

Figure 4: Ninf LAN Linpack performance for Single Alpha client

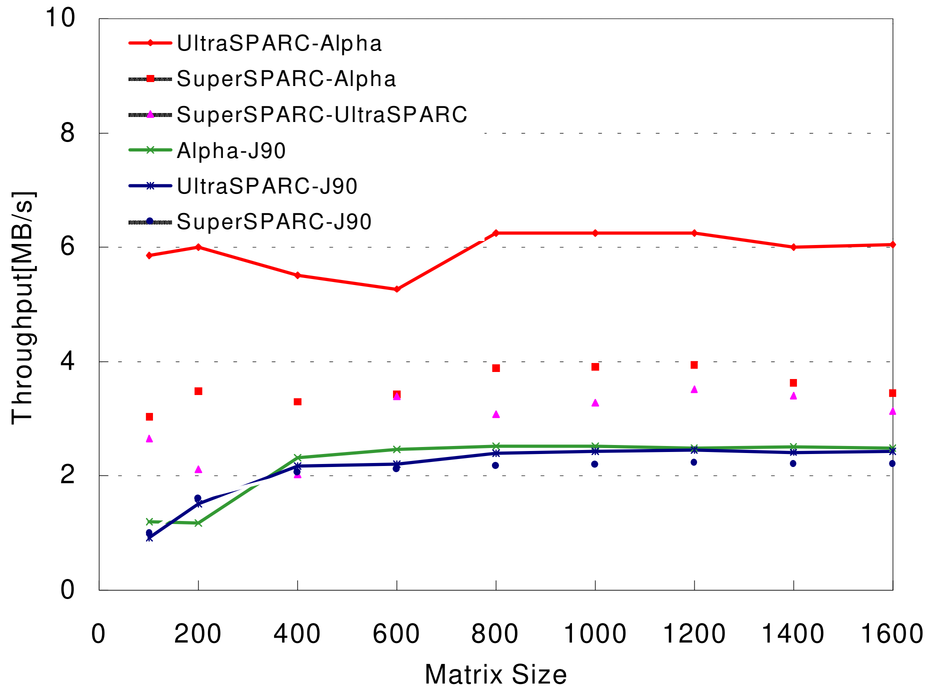

Figure 5: Communication Throughput of *Ninf_call*

Client	Server(Throughput[MB/s])		
	UltraSPARC	Alpha	J90
SuperSPARC	4	4	2.8
UltraSPARC	-	7.4	2.7
Alpha	-	-	2.9

Table 2: Client-Server FTP Communication Throughput

Figure 5 shows the communication throughput of *Ninf_call*. As Ninf sends data in XDR packets, marshalling/unmarshalling at both the client and the server, and communication inbetween, occur in parallel. Thus, we decided to include the time for marshalling the arguments in our throughput figures. We also measured the FTP (raw) communication performance between the client and server for baseline comparison.

In the figure, the three lines saturating at approximately 2MB/s are SPARC and Alpha clients versus J90 Ninf server throughput, indicating that *Ninf_call* to J90 cannot achieve high throughput due to slow baseline communication performance. The two lines saturating at approximately 3.5 MB/s are SPARC clients versus Alpha Ninf server, and those saturating at around 6 MB/s are when the client and the server are the same architecture, UltraSPARCs and Alphas. In all cases, we see that we are attaining nearly the same throughput as those for FTP (Table 2); thus various communication overhead such as XDR marshalling is not affecting performance significantly.

4 Multi-client LAN/WAN Benchmarks

In global computing, Ninf servers will be processing requests from multiple clients making *Ninf_call*s simultaneously. In order for the Ninf system (and other global computing systems) to operate effectively, the average processing time should not abnormally increase, and server throughput must be maintained. It is also necessary to measure and identify the feasibility and the possible bottlenecks in WAN.

Under these observations, we performed extensive benchmarks using multiple clients in both LAN and WAN settings. We again employed the Linpack Benchmark and NASPar EP.

4.1 Multi-client LAN/WAN Benchmarking Environment

As a model client program, we employ a routine which calls either the Linpack (sgetrf and sgetrs) or the EP routine repeatedly. We assume that each client performs a *Ninf_call* on the interval of s seconds with probability p. We also assume that the number of clients and the matrix size are fixed at c and n during the benchmark, respectively. We set the other parameters to be $s = 3$, $p = 1/2$ and $c = 1,2,4,8,16$.

- For LAN benchmarks, we used J90 and SuperSPARC SMP as the computing server, and nodes on an Alpha WS cluster as clients (Figure 2).

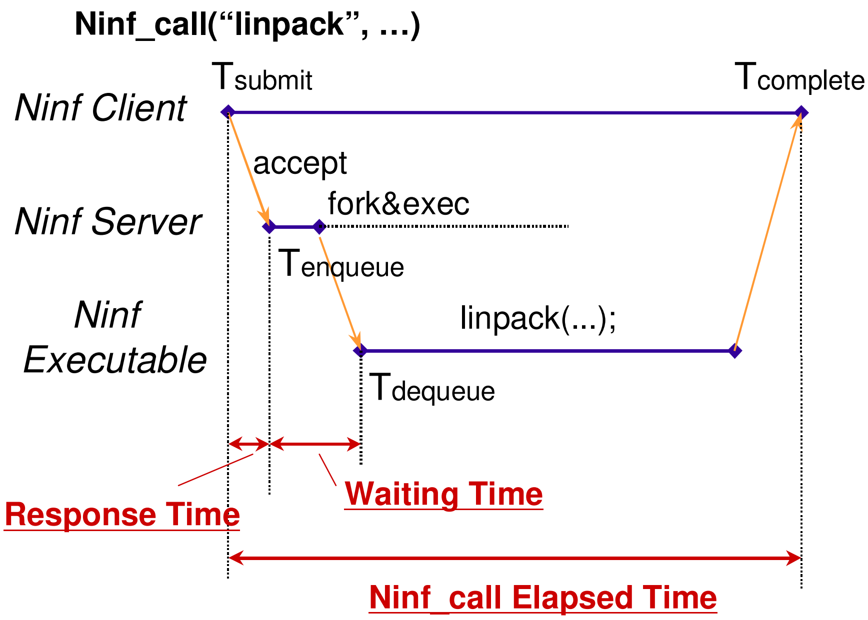

Figure 6: Multi-client *Ninf_call* Measurements

- For single-site WAN benchmarks, we also used J90 as the server, and employed 8 to 16 nodes from the SuperSPARCs at the Ochanomizu University located approximately 60km away. The FTP throughput between the client and the server was measured to be approximately 0.17MB/s.
- Multi-site WAN benchmark environment will be covered in Section 4.2.3.

We measured the server's processor utilization, load average, and for each client *Ninf_call* task, we measured the throughput and various timings: time of task submission T_{submit}, time when the *Ninf_call* task was accepted at the server $T_{enqueue}$, time when the corresponding Ninf executable was invoked $T_{dequeue}$, and the time at which *Ninf_call* was completed $T_{complete}$. Here, we define the response time $T_{response}$ and the waiting time T_{wait} of a *Ninf_call* to be as follows:

$$T_{responce} = T_{enqueue} - T_{submit}$$

$$T_{wait} = T_{dequeue} - T_{enqueue}$$

As stated in Section 1, there is an option as to how to execute the multiple client requests: 1) distribute the computing resources amongst different client requests in a *task parallel manner*, or 2) allocate all the processors to each client task in a *data parallel manner* to be processed in sequence. If the overhead of switching multiple parallel, computationally intensive tasks is high, performance degradation of the latter would be significant. We also note that most typical non-numerical server tasks (such as WWW HTTPD service) take the former approach. To investigate the tradeoffs for J90, the task parallel version has one PE serve each *Ninf_call* up to 4 parallel tasks, and the data-parallel version employs a optimally vectorized and parallelized version with simultaneous execution on 4 PEs for each *Ninf_call*, invoked in sequence.

4.2 Multi-client Linpack Benchmark Result

4.2.1 LAN Linpack Benchmark Results

Tables 3 and 4 show the performance of 1-PE and 4-PE executions. The increase in c both increases the server load and lowers communication throughput, and as a result we see decreasing *Ninf_call* performance in both 1-PE and 4-PE versions. Larger variance in the 4-PE version is due to all the PEs being utilized for Linpack execution. When n increases, both CPU utilization and load average increase, the former to saturation. Even in such as case, the J90 Ninf server continued to work flawlessly with no dramatic drop in performance (avoiding anomalies such as thrashing, even for $n = 1400$, with max. load average 30 for the 4-PE version).

Figure 7 shows the average client-observed performance of *Ninf_call*. The XY-axis denote problem size n and the number of clients c, while the Z-axis indicate the performance Mflops. Compared to the 1-PE version, the 4-PE version exhibits higher load and utilization. On the other hand, the 4-PE version exhibits substantial performance edge for a small c, while there is very little performance edge for the 1-PE version for a larger c. We analyze that this is due to the following reasons: 1) because the numerical core of the 4-PE version is significantly faster due to high vectorization, there is very little loss in average execution time even if all 4 PEs are occupied for a short period, 2) switching parallel tasks on J90 poses small relative overhead compared to large-grain computing tasks, and 3) even though Linpack computation is serialized, communication with clients could be overlapped. In addition, although response and waiting time fluctuated greater for the 4-PE version, their average values did not differ considerably depending on n, c, or number of processors. We thus conclude that, using the optimized parallel library on the J90 would be appropriate for LAN execution, relieving the library providers from preparing multiple versions.

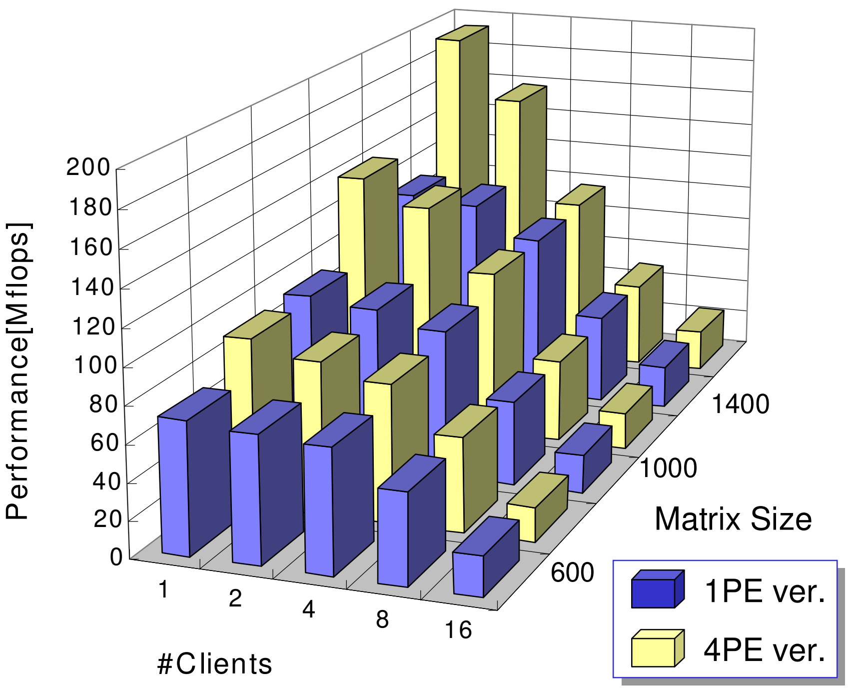

Figure 7: Average performance of multi-client LAN *Ninf_call*

n	c	Performance[Mflops] max/min/mean	response[sec] max/min/mean	wait[sec] max/min/mean	Throughput[MB/s] max/min/mean	CPU Utilization	Load Average	times
600	1	72.71/69.90/71.16	0.03/0.02/0.02	0.03/0.02/0.03	2.57/2.42/2.48	12.63	0.68	30
	2	72.01/64.33/69.63	0.01/0.00/0.00	0.04/0.02/0.03	2.52/2.10/2.41	20.86	1.2	24
	4	72.40/43.85/67.05	1.01/0.01/0.05	0.05/0.02/0.03	2.55/1.89/2.34	42.03	1.99	96
	8	72.04/17.44/49.02	5.04/0.01/0.11	0.07/0.02/0.03	2.55/0.60/1.87	82.2	4.9	184
	16	64.13/8.12/21.27	5.03/0.01/0.22	0.77/0.02/0.04	2.25/0.24/0.86	98.66	13.21	360
1000	1	95.06/93.13/93.40	0.02/0.01/0.02	0.03/0.02/0.03	2.58/2.49/2.53	21.4	1.06	30
	2	96.09/54.10/89.90	5.00/0.00/0.24	0.04/0.02/0.03	2.51/2.25/2.42	37.84	1.62	24
	4	93.64/59.92/81.39	0.20/0.01/0.02	0.08/0.02/0.03	2.58/1.47/2.11	76.02	3.58	93
	8	83.91/30.79/46.48	0.59/0.01/0.03	0.06/0.02/0.03	2.39/0.65/1.35	99.38	7.72	166
	16	33.24/15.26/21.14	5.06/0.00/0.42	0.83/0.02/0.04	1.12/0.31/0.57	100	16.01	212
1400	1	115.01/112.26/113.65	0.02/0.01/0.01	0.03/0.02/0.02	2.58/2.51/2.54	24.27	1.19	30
	2	113.58/85.15/110.48	5.01/0.00/0.21	0.03/0.02/0.03	2.54/2.36/2.44	46.95	2.19	24
	4	109.54/70.89/93.35	5.03/0.01/0.40	0.04/0.02/0.03	2.44/1.60/2.19	88.4	4.14	96
	8	64.74/42.17/50.11	5.02/0.00/0.12	0.04/0.02/0.03	1.72/0.81/1.21	99.97	8.53	104
	16	27.87/19.41/23.93	5.03/0.00/0.14	0.05/0.02/0.03	0.74/0.38/0.51	100	16.64	174

Table 3: Performance Results of 1-PE Multi-client LAN Linpack

n	c	Performance[Mflops] max/min/mean	response[sec] max/min/mean	wait[sec] max/min/mean	Throughput[MB/s] max/min/mean	CPU Utilization	Load Average	times
600	1	94.05/81.76/91.46	0.21/0.01/0.02	0.04/0.03/0.03	2.55/2.40/2.47	14.89	0.87	50
	2	91.05/21.62/83.17	4.50/0.00/0.19	0.06/0.03/0.04	2.46/2.04/2.36	27.56	1.42	24
	4	92.58/21.70/75.83	5.01/0.01/0.18	0.05/0.03/0.04	2.52/1.17/2.12	53.56	4.06	96
	8	89.35/24.31/51.51	0.66/0.01/0.05	0.39/0.03/0.05	2.50/0.55/1.36	90	9.98	184
	16	56.73/8.87/18.69	5.02/0.01/0.06	1.09/0.04/0.10	1.83/0.19/0.46	99.85	19.96	360
1000	1	150.96/70.39/141.43	5.02/0.01/0.31	0.04/0.03/0.03	2.56/2.46/2.51	28.64	1.45	30
	2	148.92/62.15/127.63	5.01/0.00/0.21	0.10/0.03/0.04	2.52/1.87/2.28	55.56	2.89	24
	4	134.26/61.01/92.98	0.04/0.01/0.02	0.15/0.03/0.04	2.53/0.98/1.56	87.9	6.02	96
	8	67.14/27.76/45.85	5.01/0.00/0.27	0.20/0.03/0.05	1.21/0.42/0.69	99.61	11.01	184
	16	32.22/13.93/20.33	3.71/0.00/0.10	1.42/0.03/0.08	0.59/0.20/0.30	99.99	24.81	234
1400	1	196.08/174.28/193.03	1.08/0.01/0.08	0.04/0.03/0.03	2.54/2.47/2.51	40.87	1.86	29
	2	184.17/139.53/157.98	0.01/0.00/0.00	0.05/0.03/0.04	2.48/1.80/2.13	66.79	3.17	24
	4	119.26/74.88/96.26	5.03/0.01/0.39	0.12/0.03/0.05	2.16/0.92/1.26	96.8	7.53	85
	8	69.26/34.23/48.27	5.02/0.00/0.23	0.42/0.03/0.05	0.91/0.43/0.59	99.86	15.11	104
	16	35.27/17.80/23.25	5.01/0.00/0.07	0.89/0.03/0.06	0.48/0.21/0.28	100	30.29	200

Table 4: Performance Results of 4-PE Multi-client LAN Linpack

n	c	Performance[Mflops] max/min/mean	response[sec] max/min/mean	wait[sec] max/min/mean	Throughput[MB/s] max/min/mean	CPU Utilization	Load Average	times
600	4	4.37/2.98/3.80	1.36/1.07/1.18	0.68/0.13/0.16	1.09/0.20/0.43	49.92	6.08	90
	8	4.25/2.73/3.51	2.10/1.15/1.29	1.60/0.13/0.17	1.05/0.14/0.37	62.91	8.84	170
	16	3.77/2.03/2.81	5.54/0.00/0.33	1.48/0.14/0.20	1.40/0.10/0.34	89.89	15.37	330

Table 5: SMP Multi-client LAN Linpack Results

We also ran the Linpack benchmark on a SuperSPARC SMP server with 16 nodes (Figure 2 and Table 5.) For 1-PE versions, compared to the J90, its performance is more resilient to increase in c, and so are response time and waiting time. Also, CPU utilization still has not saturated even for $c=16$. On the other hand, recent preliminary measurements show that highly-multithreaded versions exhibit notable slowdown as c increases (e.g., when number of threads = 12). We speculate the reasons to be as follows: 1) SuperSPARC SMP is less I/O bound compared to J90 because of lower numerical performance, and 2) Solaris 2.5 on SPARC SMP is optimized for handling multiple client requests in server-client situations, and do not co-schedule multiple threads well, resulting in various thread-switching overhead, including cache and TLB misses.

4.2.2 Single-Site WAN Linpack Benchmark Results

Tables 6 and 7 show the WAN Linpack performance for 1-PE and 4-PE executions. As n increases, the ratio between computation vs. communication increases as well, improving the *Ninf_call* performance as in LAN. On the other hand, when c increases, the amount of data sent between the network connecting the two sites increases, causing severe drop in communication throughput. Because of this, the number of *Ninf_call*s effectively decrease, and as a result, server CPU utilization and load average remains low even for $c = 16$. Thus, under current Internet network throughput, it is difficult for the Ninf computing server to process numerous client requests emanating from the same site (or, at least sharing the same link to the backbone) for communication intensive tasks, requiring a good load balancing algorithm that not only takes into account server load such as is done for NetSolve [6], but also network traffic and global allocation of tasks.

Figure 8 shows the average client-observed performance of *Ninf_call* under WAN for task- (1-PE) and data-parallel (4-PE) executions. It exhibited almost the same characteristics as LAN; in fact, even when c is large, because the server performance has not saturated, the 4-PE versions exhibited better performance. The same is true for response times and waiting times. Thus, we observe that, it is preferable to use the optimized library versions for WAN clients as well.

n	c	Performance[Mflops] max/min/mean	response[sec] max/min/mean	wait[sec] max/min/mean	Throughput[MB/s] max/min/mean	CPU Utilization	Load Average	times
600	1	7.81/3.25/5.90	0.05/0.03/0.04	0.10/0.07/0.07	0.166/0.067/0.128	4.22	0.37	10
	2	5.93/3.63/4.69	0.27/0.00/0.14	0.21/0.07/0.14	0.122/0.074/0.096	8.03	0.49	24
	4	3.63/1.22/2.41	5.30/0.00/0.64	3.07/0.07/0.25	0.075/0.025/0.050	8.34	0.47	31
	8	1.47/0.59/1.14	5.25/0.27/2.26	0.46/0.08/0.19	0.030/0.012/0.023	7.8	0.46	59
	16	0.69/0.37/0.54	3.84/0.00/0.55	1.38/0.12/0.31	0.014/0.007/0.011	8.1	0.46	115
1000	1	12.05/4.41/9.28	1.78/0.01/0.20	0.10/0.07/0.08	0.164/0.054/0.123	5.26	0.39	10
	2	8.47/5.81/7.18	0.27/0.01/0.16	1.36/0.08/0.21	0.104/0.071/0.088	9	0.52	24
	4	4.38/1.84/3.66	2.17/0.03/0.23	1.38/0.07/0.24	0.055/0.022/0.045	8.83	0.52	31
	8	2.27/0.99/1.83	5.00/0.12/2.48	1.48/0.08/0.30	0.028/0.012/0.022	8.39	0.47	59
	16	1.18/0.54/0.90	5.21/0.71/1.97	1.59/0.07/0.32	0.014/0.006/0.011	8.22	0.47	115
1400	1	17.32/9.93/13.89	0.04/0.00/0.00	1.42/0.07/0.21	0.164/0.090/0.130	6.57	0.41	10
	2	11.22/7.91/9.29	2.28/0.00/0.18	0.35/0.08/0.15	0.099/0.069/0.082	12.75	0.66	24
	4	6.47/2.15/5.38	2.20/0.13/0.37	0.40/0.08/0.16	0.058/0.019/0.048	9.46	0.52	31
	8	3.03/1.31/2.50	7.29/0.00/2.87	26.71/0.07/0.89	0.027/0.011/0.022	8.89	0.5	59
	16	1.58/0.76/1.25	10.51/0.93/2.55	4.69/0.08/0.47	0.014/0.007/0.011	10.6	0.57	115

Table 6: Single-site, Multi-client 1-PE Linpack benchmark results for WAN

n	c	Performance[Mflops] max/min/mean	response[sec] max/min/mean	wait[sec] max/min/mean	Throughput[MB/s] max/min/mean	CPU Utilization	Load Average	times
600	1	8.00/7.24/7.68	0.15/0.14/0.14	0.08/0.07/0.08	0.166/0.153/0.161	6.85	0.4	10
	2	6.05/1.51/3.77	0.24/0.00/0.06	0.20/0.07/0.11	0.124/0.030/0.077	6.9	0.42	24
	4	3.57/0.92/2.46	2.61/0.00/0.23	0.57/0.10/0.19	0.073/0.019/0.051	8.94	0.5	31
	8	1.52/0.49/1.08	5.93/0.65/2.67	3.60/0.08/0.40	0.031/0.011/0.022	7.99	0.49	59
	16	0.90/0.29/0.54	10.24/0.00/1.30	17.54/0.10/0.78	0.018/0.006/0.011	8.17	0.49	115
1000	1	12.75/7.26/10.50	0.14/0.11/0.13	0.09/0.08/0.08	0.160/0.091/0.131	6.43	0.39	10
	2	8.22/5.89/7.15	0.36/0.07/0.18	0.20/0.08/0.16	0.101/0.072/0.088	9.23	0.52	24
	4	4.80/1.46/3.97	2.52/0.09/0.34	0.47/0.10/0.19	0.061/0.018/0.049	9.39	0.53	31
	8	2.42/0.97/1.82	6.74/0.47/3.18	7.51/0.08/0.47	0.029/0.012/0.022	8.71	0.53	59
	16	1.36/0.47/0.88	3.63/0.02/0.91	94.45/0.09/1.15	0.016/0.006/0.011	8.75	0.49	115
1400	1	18.46/11.87/16.42	1.94/0.06/0.27	0.09/0.08/0.08	0.166/0.104/0.147	8.15	0.47	10
	2	10.44/8.03/9.34	0.29/0.00/0.07	0.33/0.07/0.13	0.092/0.070/0.082	13.58	0.73	24
	4	7.00/2.00/5.50	2.62/0.00/0.34	0.34/0.12/0.19	0.061/0.017/0.048	15.61	0.81	31
	8	2.91/1.42/2.46	7.16/0.37/3.41	1.67/0.08/0.32	0.025/0.012/0.021	9.2	0.56	59
	16	1.59/0.79/1.25	9.64/0.39/1.50	1.77/0.08/0.27	0.014/0.007/0.011	9.35	0.54	115

Table 7: Single-site, Multi-client 4-PE Linpack benchmark results for WAN

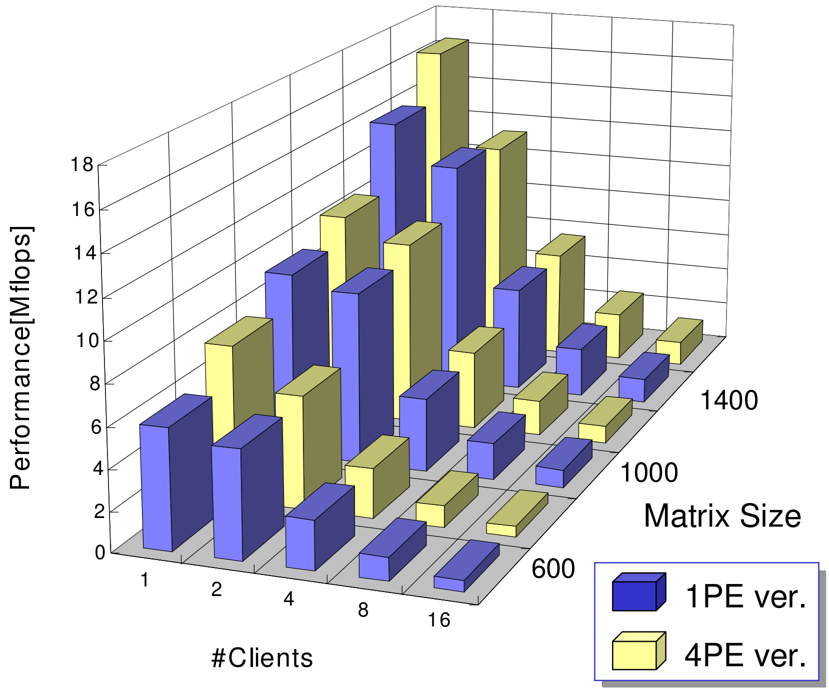

Figure 8: Average Performance of WAN Linpack ***Ninf_call***

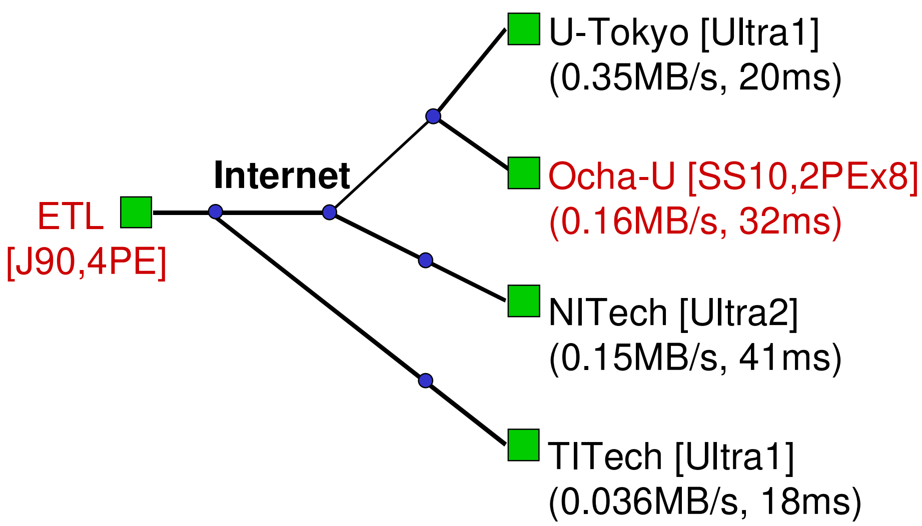

Figure 9: WAN Multi-client Benchmarking Environment

4.2.3 Multi-site WAN Linpack Benchmark Results

In global computing, in practice clients will be located at different sites in WAN. If simultaneous communication from multiple WAN sites could achieve their aggregate bandwidth so that server load could be maintained high, then global computing would be feasible for communication

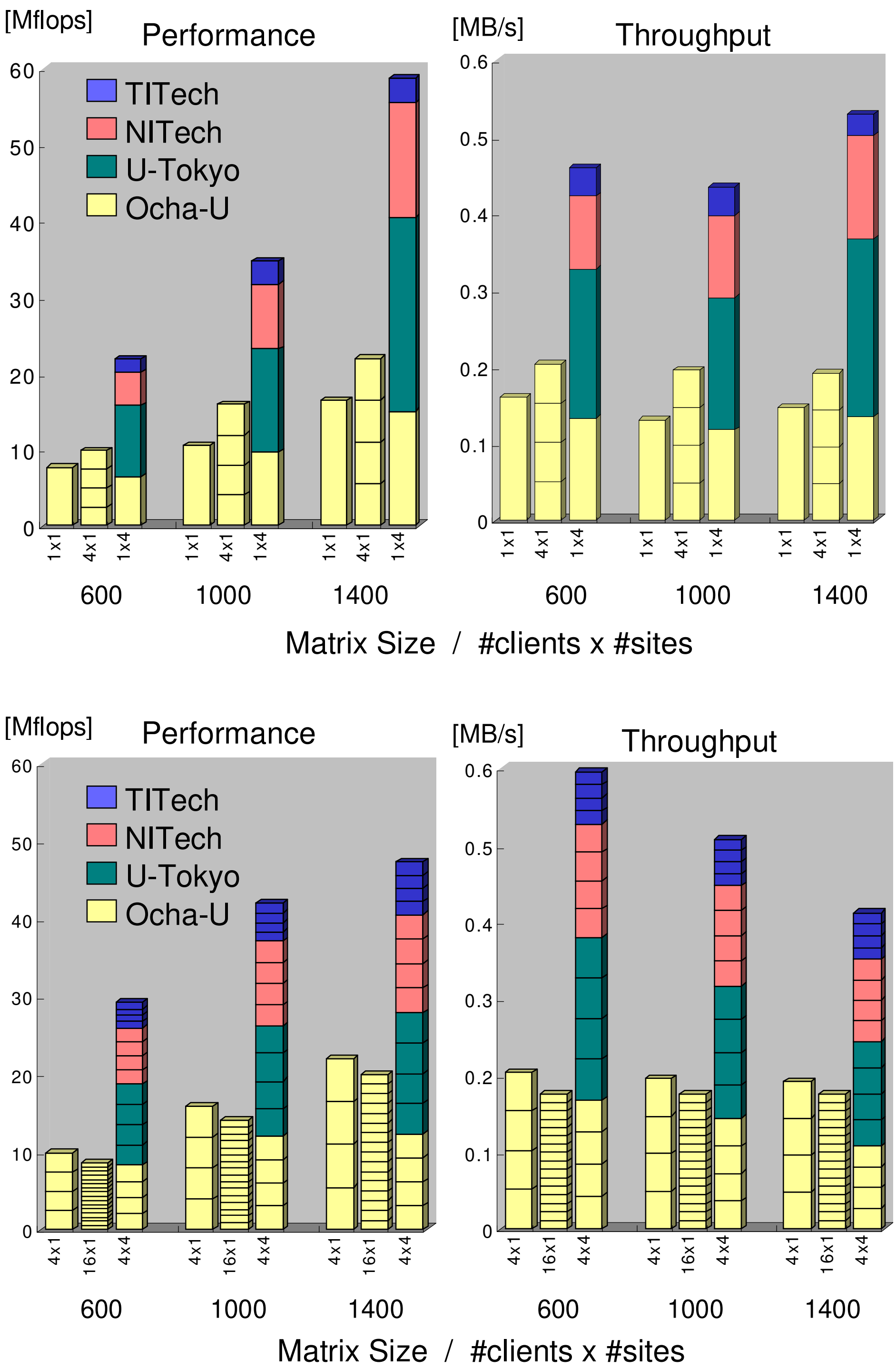

Figure 10: Multi-client, Multi-site WAN Linpack Benchmark Results

intensive applications; otherwise, if performance degrades just as the single-site WAN case, then global computing would have smaller market under the current Internet.

For multi-site WAN benchmark, we prepared clients at 4 different University sites in Japan, namely, Ochanomizu University (Ocha-U), University of Tokyo (U-Tokyo), Nagoya Institute of Technology (NITech), and Tokyo Institute of Technology (TITech), while the server is the J90 at ETL. The sites are connected to different backbones as we illustrate in Figure 9. The Linpack routines registered at the J90 Ninf server is the 4-PE version. Figure 10 shows the results.

We can observe that, aggregate communication throughput from multiple sites are substantially higher than that from a single site. In fact, between Ocha-U and ETL, multi-site communication performance deteriorated only by 9%~18% when $c = 1$ (total 4 clients in WAN), and 18%~44% when $c = 4$ (total 16 clients in WAN), maintaining substantially higher bandwidth than when the same number of clients are located at Ocha-U. As a result, CPU utilization and load average is substantially greater for multi-site compared to single-site. Performance-wise, we observe that the rate of performance increase for $c = 1$ is greater than when $c = 4$. Although one plausible explanation would be that the greater number of clients and larger problem size had saturated the computational power of J90, this is not the case as CPU utilization levels off even for $c = 4$ (approx. 27%~34%, not shown in the figure). Because throughput steadily declines for $c = 4$, we believe that network bandwidth saturation is again the cause.

Still, our initial conclusion that, 1) in WAN settings, for communication intensive tasks, point-to-point bandwidth between the client and the server is the dominant factor in determining client-observed performance (and not the current load average of the server), and 2) in general, client *Ninf_call*s should be distributed to multiple Ninf servers located at different network sites, avoiding concentration of tasks from a single site or those that share the same backbone, in order to maintain performance; otherwise, performance quickly deteriorates significantly as number of clients increase.

4.3 LAN/WAN EP Benchmark Results

EP(An embarrassingly parallel benchmark) is one of the kernel programs in the NAS Parallel Benchmark[7], performing (random-number) Monte-Carlo simulations.

For Ninf benchmarking, we execute the random number generation on the Ninf server. Communication complexity is O(1), namely, problem-size independent. Computational complexity is proportional to the number of random numbers generated, and becomes 2^{n+1} for 2^n trials. Therefore, The execution performance $P_{N\inf\_call}$ becomes:

$$P_{N\inf\_call} = \frac{2^{n+1}}{T_{N\inf\_call}}$$

where $T_{N\inf\_call}$ is the time taken for the entire *Ninf_call*. In each benchmark run, we execute 2^{24} trial samples for each PE, and execute them in a task-parallel manner on the 4-processor J90.

4.3.1 EP Benchmark Results

Table 8 shows the EP benchmark performance under LAN and single-site, multi-client WAN execution. Because of task-parallel execution, in both LAN and WAN cases *Ninf_call* performances are almost equivalent and are sustained up to $c = 4$ clients. The performances decline as expected when c further increases, but the server utilization remains approximately 100%. This is due to very low communication requirements of EP, and the *Ninf_call* performance mainly reflects server computing performance. We strongly expect that the result will hold for the multi-site, multi-client WAN case. Thus, for this class of applications such as parallel rendering/imaging, and parameter sensitivity analysis, global computing can now be considered quite feasible.

	c	Performance[Mops] max/min/mean	Response[sec] max/min/mean	Wait[sec] max/min/mean	Transmission[sec] max/min/mean	CPU Utilization	Load Average	Times
LAN	1	0.168/0.166/0.167	0.07/0.00/0.01	0.02/0.02/0.02	0.01/0.01/0.01	30.51	1.44	10
	2	0.169/0.166/0.168	0.03/0.00/0.01	0.03/0.02/0.02	0.01/0.01/0.01	53.86	2.39	17
	4	0.167/0.163/0.166	0.08/0.00/0.01	0.03/0.02/0.02	0.01/0.01/0.01	98.18	4.18	31
	8	0.086/0.080/0.084	4.91/0.00/0.09	0.04/0.02/0.02	0.01/0.01/0.01	100	8.9	59
	16	0.044/0.040/0.042	0.05/0.00/0.00	0.05/0.02/0.02	0.02/0.01/0.01	100	16.88	115
WAN	1	0.169/0.166/0.168	0.27/0.00/0.06	0.08/0.06/0.06	0.06/0.04/0.04	25.02	1.21	10
	2	0.168/0.164/0.168	0.13/0.00/0.03	1.26/0.06/0.13	0.06/0.04/0.04	49.16	2.19	17
	4	0.168/0.163/0.166	1.71/0.00/0.09	0.08/0.06/0.06	0.24/0.04/0.05	98.14	4.16	31
	8	0.087/0.081/0.084	0.31/0.00/0.06	1.21/0.06/0.08	0.05/0.04/0.04	100	8.91	59
	16	0.044/0.040/0.042	0.31/0.00/0.06	1.46/0.06/0.08	0.09/0.04/0.04	99.94	16.88	115

Table 8: Multi-client EP benchmark results for LAN and single-site WAN

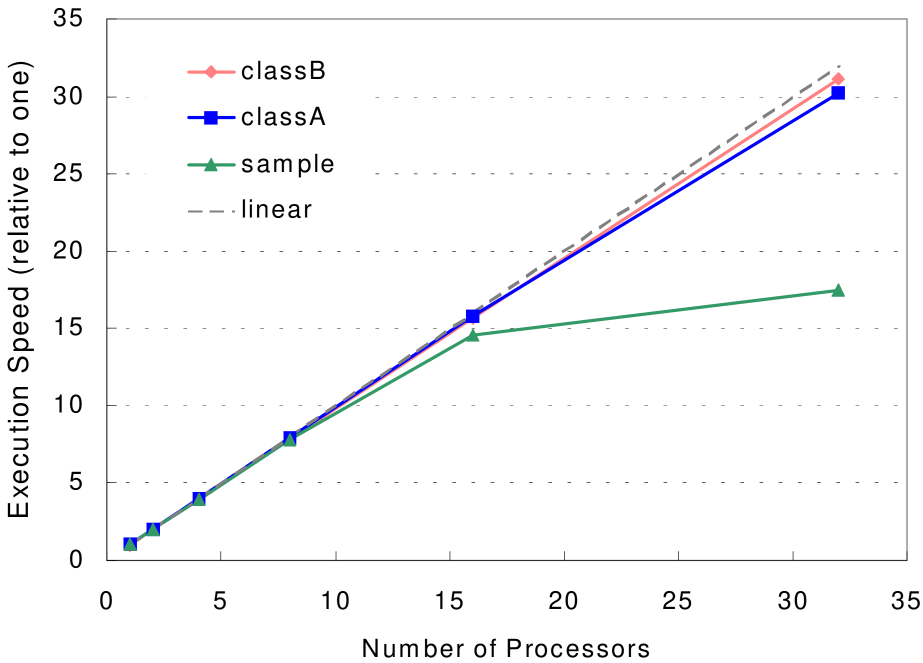

Figure 11: EP Metaserver Parallel Execution Benchmark

We also benchmarked automated load balancing using the Ninf metaserver, in order to evaluate its overhead. Because Ninf distributes per each *Ninf_call*, task-parallel execution of EP would be as follows:

```
Ninf_transaction_begin();
for (i = 1; i <= numprocs(); i++) {
        Ninf_call("ep", ...);
}
Ninf_transaction_end();
```

Because there are no data-dependencies between multiple *Ninf_call*s, they will all be scheduled and executed in a task-parallel manner. Given p processors, the effective *Ninf_call* performance would be:

$$P'_{N\inf\_call} = \frac{2^{n-\log_2 p+1}}{T_{N\inf\_call}}$$

In the benchmark, we employed a 32-processor Alpha cluster in LAN, with each node becoming a Ninf computing server. Figure 11 shows the results. For larger number of trials 2^{28} (class A) and 2^{30} (class B), we achieve almost linear speedup; however, for 2^{24} (sample), we observe significant slowdown; this is because the prototype Metaserver is written in Java, and the overhead of scheduling and distributing *Ninf_call* has become apparent compared to smaller problem size due to parallel distribution of the EP task. We expect that result will hold for WAN cases from the results of the previous benchmarks. We also conducted benchmarks with DOS (Density-Of-States) calculation, which is an EP-style practical application in computational chemistry, and came up with similar results.

5 Discussions

5.1 Guaranteeing Performance in Multi-Client Global Computing

Performance per each client under multi-client situation cannot be guaranteed irrespective of computing server performance. The same holds true when a single client makes multiple *Ninf_call*s to the same server. Although it is possible to restrict the number of remote clients, standard TCP-based RPC-protocols require clients and servers to stay connected, as is with current Ninf-RPC. In order to guarantee per-user performance, an alternative is to modify *Ninf_call* to become a two-phase transaction, where remote argument transfer takes place in the first phase, whereupon the communication is terminated, and after the server computation is over, the client is notified so that it may receive the results in the second phase. We have already implemented such a two-phase protocol for database queries in Ninf [8]. A similar approach is to submit jobs to an intermediate broker server as is with Javelin[9].

As another issue, with the current Internet lacking sufficient bandwidth or bandwidth reservation protocols, communication would become the bottleneck rather than the server's underutilized computing power as we have observed. To resolve this, we are planning to extensively employ the metaserver, which will effectively schedule computation utilizing various performance and communication parameters. In particular, IDL and server execution trace will give us effective information for predicting the communication transfer time versus computing time, making it possible to assign communication- and computation-intensive tasks to appropriate servers.

5.2 Server Job-handling Methodology

Under multi-client setting, we need to improve average response time for *Ninf_call*, and increase CPU utilization. However, the current Ninf server merely *fork* & *exec*s a Ninf executable in a First-Come-First-Served (FCFS) manner, causing longer response time and possibly lower CPU utilization. By predicting the computation and communication time of a *Ninf_call* task using IDL and server trace information, we could perform Shortest-Job-First (SJF) scheduling, improving the response time and utilization considerably. Also, as we have noted, for SMPs, co-scheduling of jobs would be effective in increasing overall system performance, while there would be some sacrifice in system response time.

5.3 Multi-Job Scheduling for MPP Servers

Our benchmark utilized 4 processors for J90 and 16 processors for SPARC SMP. As processor numbers increase, we expect the tradeoffs of various processor assignments to *Ninf_call* could become different. In particular, task switching of large SPMD tasks could be expensive; in such a case, task-parallel execution could be more efficient. To resolve this, we could extend Ninf IDL to call routines that utilize different number of PEs depending on the problem size.

Additionally, in such a case simple FCFS scheduling may not be the most effective scheduling policy, causing many processors to become idle. To overcome this drawback, we could employ more suitable algorithms such as Fit Processors First Served (FPFS) or Fit Processors Most Processors First Served (FPMPFS)[10]. Still, we need to investigate further which algorithm would be more appropriate for global computing systems such as Ninf.

6 Related Works

The Remote Computation System (RCS)[11] at ETH provides a RPC facility which unifies the interfaces to multiple supercomputers. Because the communication layer is based on PVM it is not well-suited for global computing, and the API lacks the flexibility provided by Ninf.

NetSolve [6] provides almost the same basic API as *Ninf_call*, and achieves load-balancing with a daemon process called Agents. There are various technical differences between Ninf and NetSolve; examples for Ninf are availability of parallel transactions for load distribution and fault tolerance, while NetSolve facilitates description of computational complexity in library IDL. Also, database server APIs are unique to Ninf. We do expect, however, the results of our benchmark work will also hold for NetSolve, and will have important implications for similar systems. For example, current NetSolve attempts to perform load balancing solely on server load average information; as we have seen, this might partially work for LAN situations, but would not scale to WAN settings.

Legion [12] is a powerful global computing system based on the model of distributed object. A client user distributes his programs across the network by programming with a parallel object-oriented language Mentat [13]. The difference between Mentat and other systems is that it assumes a relatively closed system while Ninf/NetSolve/RCS re-use existing programming languages (although it is possible in Legion to call external programs with wrappers.) Also, Legion assumes more finer-grained computing objects. Because of this, we have to be careful in observing whether our benchmark results will hold for Legion, but we expect that many of them will, as it is not possible to achieve good performance in global computing in general without being relatively coarse-grained.

There are other global computing projects that focus on different level of system architecture, such as Globus/Nexus[4], or whose focus is more on portability and other issues, such as Javelin[9].

7 Conclusion and Future Work

We performed an extensive performance analysis of Ninf, a global computing system for high-performance computing. We ran several benchmarks with different communication/computation characteristics on a variety of combinations of clients and servers, in their performance, architecture, etc., under LAN, single-site WAN, and multi-site WAN situations. We conclude with the followings:

- The current Ninf system (and, other systems such as NetSolve) would function in both LAN and WAN situations. The use of optimized parallel library would be sufficient for setting up numerous global computing servers all around the world, although for SMPs we do need to determine the optimal number of threads given the number of clients.
- On the other hand, there must be a variety of improvements that must be made, as we have indicated, both in communication architecture of the client-server, and task scheduling/load distribution, especially in WAN setting.
- In LAN setting, computing server performance dictates overall performance, while in WAN limitation of communication throughput is currently more significant. In particular, multiple client requests from a single site could quickly saturate the network for communication-intensive applications. However, we expect that in practice multiple client requests will be issued from different sites, in which case performance will depend of Internet network topologies. In fact, for EP-style applications, current Ninf or other systems such as NetSolve would be quite feasible.

Finally, although we have performed extensive benchmarks on two application core examples, we still do need to perform even more benchmarks using other types of practical applications, and different WAN configurations. However, on the Internet it is quite difficult to perform large-scale benchmarks with reproducible results. One current plan we have is to build a global computing simulator for Ninf, on which we could readily test different client network topologies under various communication and other parameters. Still, we do need to conduct further application benchmarks to determine parameters in the underlying global computation model for the Ninf simulator.

We also need to make our Ninf system more robust so that it could be readily distributed for global public usage, upon which many applications should be written, based on which we could gain more knowledge on the behavior of global computing systems in general. We also hope to collaborate with other efforts in global computing so that we could share as much of the technical results and infrastructures as possible.

Footnotes

1 Although the current Ninf client API only supports matrices, we plan to extend the IDL and the interpreter to handle arbitrary user-defined objects.

References

[1] http://phase.etl.go.jp/ninf/

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

[3] H. Nakada, M. Sato, and S. Sekiguchi. An overview of the RPC System Implementation for Ninf, Technical Report TR95-28, Electrotechnical Laboratory, 1995.

[4] I. Foster, and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications (to appear)*, 1997.

[5] Oguni, editor. *Matrix Computing Software*. Maruzen, 1991.

[6] H. Casanova, and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. In *Proceedings of Supercomputing '96*, 1996.

[7] http://science.nas.nasa.gov/Software/NPB/

[8] M. Iioka et al. Global Numerical Information Database Server System for Ninf. In *Proceedings of 13th Workshop on Object-Oriented Computing, Hakone, Japan*, 1997.

[9] P. Cappello, B. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Computing Using Java. In *1997 ACM Workshop on Java for Science and Engineering Computation (submitted)*, 1997.

[10] K. Aida, H. Kasahara, and S. Narita. Scheduling Scheme of Parallel Jobs on a Multiprocessor System, *Supercomputing '97 (submitted to Technical Paper)*, 1997.

[11] P. Arbenz, W. Gander, and M. Oettli. The Remote Computational System, High-Performance Computation and Network. Volume 1067 of *Lecture Note in Computer Science*, pages 662-667, Springer, 1996.

[12] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr. Legion: The Next Logical Step Toward a Natiowide Virtual Computer. Technical Report CS-94-21, University of Virginia, 1994.

[13] A. S. Grimshaw. Easy to Use Object-Oriented Parallel Programing with Mentat, *IEEE Computer*, pages 39-51, 1993.