ネットワーク数値情報システム Ninf: マルチクライアント環境での性能

竹房 あつ子[†1] 小川 宏高[†2] 松岡 聡[†3]
中田 秀基[†4] 佐藤 三久[†5]
関口 智嗣[†4] 長嶋 雲兵[†1]

ネットワーク数値情報システム Ninf は，広域ネットワークに分散した計算資源や情報資源を利用して高速・高性能・高品質な科学技術計算を実現するための基盤システムである．本稿では Ninf の全体構成について述べるとともに，Linpack benchmark を用いて，シングル / マルチクライアント環境での本システムの性能評価を実施した．この結果，現実的な広域分散利用条件での Ninf システムの実用性，堅牢性を確認した．また，既存のベクトルパラレルマシンを始めとする並列計算機が Ninf によってネットワーク計算資源として有効に活用されることを示した．

Network Numerical Information System *Ninf*: Performance for Multi-Clients

Atsuko Takefusa,[†1] Hirotaka Ogawa,[†2] Satoshi Matsuoka,[†3]
Hidemoto Nakada,[†4] Mitsuhisa Sato,[†5] Satoshi Sekiguchi[†4]
and Umpei Nagashima [†1]

To establish a basis for globally-distributed parallel computing in numerial computing, we are currently working on the Ninf (**N**etwork based **Inf**ormation Library towards a Globally High Performance Computing) software system. To evaluate the Ninf system, we perform Linpack Benchmark with the Ninf system on Cray J90 vector-parallel supercomputer and DEC Alpha cluster of workstations, and Sun workstations. Results show that the utility and robustness of the Ninf system, and multicomputers such as vector-parallel computers and MPPs can effectively support network information services via Ninf.

1. はじめに

ネットワーク数値情報システム **Ninf**☆ (**N**etwork **Inf**omation Library towards a Globally High Performance Computing) は科学技術計算分野における広域分散並列計算技術を支援する基盤システムである[1)]．Ninf を用いることにより，広域ネットワーク上に分散する超高性能計算機を利用した超高速なリモート数値計算ライブラリ等の計算資源や，各種データベースをはじめとする情報資源を統合して，高速かつ高性能・高品質な科学技術計算アプリケーションを実現できる．

広域分散並列処理は以下の前提の下に成立する:
(1) 通信バンド幅の増大は見込めるがレイテンシの短縮は難しい．信頼性を補うため冗長な通信手続きが必要．
(2) 計算資源として最新の超高性能計算機を利用するため，クライアントと比較して常に圧倒的に高速．

従って WS クラスタ + メッセージ通信ライブラリによって比較的細粒度で実現される分散並列処理とは異なる．一方，Ninf システムは通信の信頼性を保証する機構を備えたクライアント - サーバシステムベースであり，疎粒度のリモート数値計算ライブラリなどを提供する．しかし，このシステムが有効に機能することを実証するには実際の運用結果を待たねばならない．

Ninf サーバはクライアントからのアクセス集中にもプロセスハンドリングを破綻なく遂行する必要がある．Ninf がサーバの対象とするベクトル機や MPP など超高性能計算機の OS は，UNIX サーバ等と比較して，クライアント - サーバ環境で多数のプロセスの同時実行を考慮されていない．これらの OS と Ninf の組合せで十分な堅牢性を維持することが求められる．

また，超高性能計算機，特にベクトルパラレル，MPP 等の並列計算機で Ninf サーバを運用する際，順次到着する計算要求の処理指針として 2 つの選択があり得る．即ち，計算資源を分割して複数タスクを**並列に処理するか**，全資源を一つのタスクに割り当てて**直列に処理するか**である．この選択は提供するライブラリの設計段階に関わる．つまり，ネットワークサービスに特化さ

†1 お茶の水女子大学 Ochanomizu University
†2 東京大学 The University of Tokyo
†3 東京工業大学 Tokyo Institute of Technology
†4 電子技術総合研究所 Electrotechnical Laboratory
†5 新情報処理開発機構 Real World Computing Partnership
☆ http://phase.etl.go.jp/ninf/

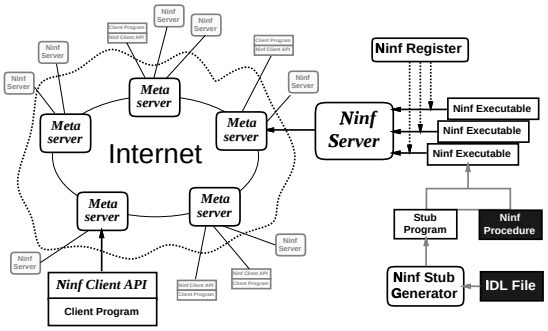

図 1 Ninf システムアーキテクチャ

れた計算ルーチンを用意する必要があるか，単にマシンの最高性能を達成する最適化されたルーチンで有効に運用できるかが問題である．

本稿では，Ninf システムの概要について述べるとともに Linpack Benchmark を用いて Ninf サーバの性能評価実験を行い，以下の結果を得た:

(1) ローカル実行より Ninf を用いた方が高速であることから，広域分散並列処理の有効性を実証．

(2) サーバにベクトル並列機 Cray J90 4PE を用い，複数クライアントが並行にアクセスする環境での実験結果から，並列計算機用 OS と Ninf システムの組合せで堅牢なネットワークサービスが提供できることを確認．

(3) 並列計算機ではピーク性能を達成する最適化されたライブラリで Ninf サービスを提供することで十分効率良く運用できることが判明．

2. Ninf システムの概要

Ninf システムはクライアント - サーバモデルに基づく．クライアントからは LAN/WAN に接続されたサーバが提供する計算ライブラリをはじめとする様々な資源を，サーバ上のデーモンプロセスを介して利用できる．**図 1**に Ninf システムアーキテクチャを示す．

Ninf システムは，Ninf のサービスを提供する **Ninf サーバ**，Ninf クライアントのインタフェースを提供する **Ninf クライアント API**，広域ネットワークに分散した Ninf サービスのスケジューリングを支援する**メタサーバ**等から構成される．これらの構成要素間の通信は Ninf RPC (Remote Procedure Call)[2] と称する手続きにより実現される．Ninf Stub Generator, Ninf Register はサービスの提供を半自動化するツールであり，サーバの保守を容易にする．

これらの枠組を利用することにより，ユーザは煩雑な手続きなしに Ninf のサービスを用いた広域分散並列処理を実現することができる．

2.1 Ninf サーバ

Ninf サーバは計算ライブラリやデータベース等の資源を提供するホスト上のデーモンプロセスで，クライアントとの通信やサービスの開始・終了の管理を行う．提供されるサービスは Ninf Executable と称する実行ファイル形式を採り，サーバはこれらを適宜 fork&exec

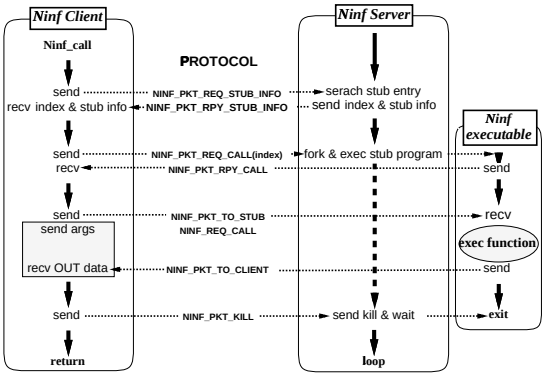

図 2 Ninf サーバ - クライアント間通信の概要

する．Ninf Executable は提供するライブラリと通信ランタイムを含んだ stub ルーチンとリンクして生成されるが，この処理は半自動化されている．

図 2に Ninf サーバとクライアント間の通信プロトコル Ninf RPC の概要を示す．Ninf RPC の実装は通信に TCP/IP，通信データ表現に Sun XDR を採用しており，多くのプラットフォームに移植容易である．

2.2 Ninf クライアント API

Ninf のクライアントは，C, C++, Fortran, Java, Lisp 等の言語と各言語用の簡易な Ninf クライアント API を用いて作成される．Ninf_call はこの API の一つで Ninf サービスの利用に用いる．Ninf_call の利用例として行列積の場合を示す．通常行列積ルーチンを呼び出す場合，

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

と記述するが，Ninf では以下のように書く．

```
Ninf_call("dmmul", n, A, B, C);
```

このようにローカルライブラリを呼び出す場合と類似した手続きで Ninf リモートライブラリが利用できる．Ninf_call は Ninf RPC を使って自動的に関数の引数の数・型を決定し，引数を使って Ninf サービスを利用した計算を行い，結果を適切な引数に格納するという一連の処理を行う．従ってユーザには自分のマシン上でリモートライブラリが実行されているように見える．

3. シングルクライアントによる Linpack を用いたサーバ性能

広域分散並列処理においてはデータの通信量と計算量とのバランスが実行性能を決定する最大の要因である．そこで Ninf の有効性を示すために，まずサーバに一つのクライアントが Ninf_call するという理想的な利用条件での評価実験を複数プラットフォーム上で行った．

3.1 Linpack Benchmark

評価には倍精度の Linpack Benchmark を実施した．Linpack Benchmark はガウスの消去法を用いた密行列の連立一次方程式の求解 (dgefa,dgesl) に要する時間を測定する．演算数が $\frac{2}{3}n^3 + 2n^2$ と一意に定まるため浮動小数点演算性能の標準としてよく引用される．

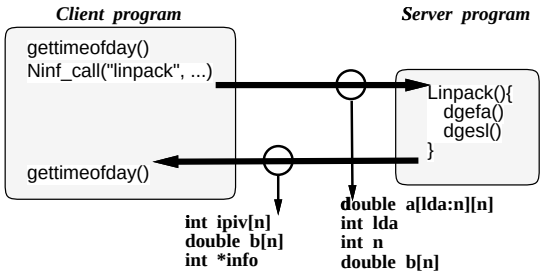

図 3 Linpack で用いたプログラムと授受データ

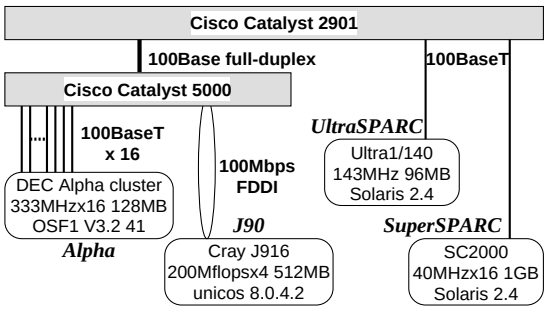

図 4 クライアント及びサーバに用いた計算機

Ninf を用いた Benchmark では， dgefa, dgesl の 2 つを Ninf サーバ上で実行する (**図 3**)．引数を含めた通信量は $8n^2+20n+O(1)$ bytes である．Ninf_call の実行時間 T_{Ninf_call} は通信時間 T_{comm} と計算時間 T_{comp} から成る．

$$T_{Ninf\_call} = T_{comm} + T_{comp} \quad (1)$$

Linpack の問題サイズを n とすると，

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

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

ここで T_{comm0}, T_{comp0} は通信と計算のセットアップにかかる時間を示す．B はクライアント – サーバ間の通信スループット，P_{calc} はサーバにおける Linpack の実行性能を表す．

また，Ninf_call の実行性能 P_{Ninf_call} は

$$P_{Ninf\_call} = \frac{2/3n^3+2n^2}{T_{Ninf\_call}} \quad (4)$$

と表せる．T_{comm} は $O(n^2)$，T_{comp} は $O(n^3)$ であるため，n が大きくなるにつれ通信のオーバーヘッドが隠蔽される．従って P_{Ninf_call} はクライアントマシンの実行性能より向上すると予測できる．

3.2 評価環境

クライアント及びサーバに用いた計算機とその OS は**図 4**に示す通りである．また，計測を行ったクライアントとサーバの組合せは**表 1**に示す．Local は Ninf を用いずにクライアント上で実行したものである．

Linpack の求解ルーチンとして，J90 では単精度の浮動小数点表現が 8 bytes であるため，libSci ライブラリの sgetrf, sgetrs を利用した．さらに 4PE を占有して高速に実行する 4PE 版と，1PE のみを用いて実行する 1PE 版を用意した．その他の計算機では LAPACK の dgefa, dgesl を用いた．問題サイズは 100 から 1600 を対象とした．

Client	Local	Ninf_call		
		Ultra	J90(1PE)	J90(4PE)
SuperSPARC	○	○	○	○
UltraSPARC	○	–	○	○
Alpha	○	–	○	○

表 1 評価に用いたサーバとクライアントの組み合わせ

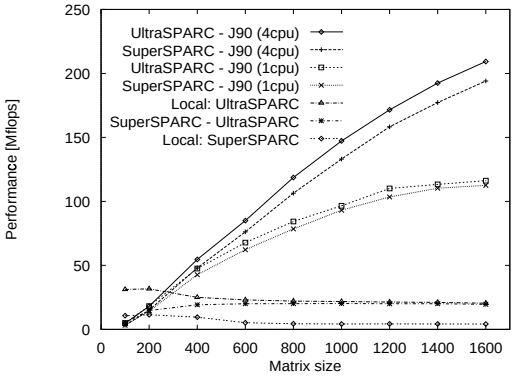

図 5 SPARC をクライアントとした Linpack の実行結果

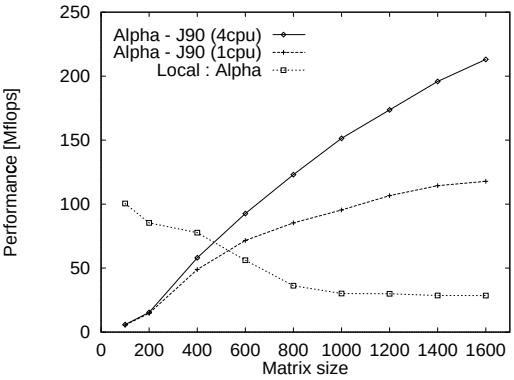

図 6 Alpha をクライアントとした Linpack の実行結果

3.3 測定結果

図 5に SuperSPARC, UltraSPARC をクライアントとした測定結果を示す．横軸は Linpack の問題サイズ，縦軸は Ninf_call と Local の実行性能を Mflops で示す．

SuperSPARC と UltraSPARC の Local は問題サイズ n によらずほぼ一定の性能を得た．一方 Ninf_call では，**3.1**で述べたように n が大きくなるにつれ性能が向上し，SuperSPARC, UltraSPARC とも $n = 200 \sim 400$ ですでに各のマシンの Local よりも高い性能が得られた．また UltraSPARC の Local と SuperSPARC から UltraSPARC への Ninf_call の測定結果を比較すると，Ninf_call は n が大きくなるにつれ，Local の性能に収束する．これは**式 (1)** において $T_{comm} \ll T_{comp}$ となり，通信オーバーヘッドが隠蔽されるためである．また，クライアントの性能が異なる場合でも Ninf_call ではほぼ同程度の性能を示した．

図 6に Alpha をクライアントとした測定結果を示す．

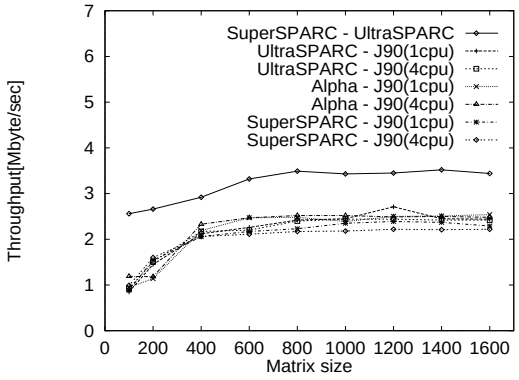

図 7 `Ninf_call` の通信スループット

Alpha の浮動小数点表現は IEEE/CRAY 形式と異なるため，J90 への Ninf_call ではクライアント，サーバ双方で通信用データ表現 XDR と計算機のデータ表現の間の変換に時間を要するが，SPARC と同様に問題サイズ 400 ～ 600 で Ninf_call が Local の性能を上回った．これはデータ表現の異なるプラットフォームでも Ninf_call の効率を著しく低下させないということに他ならない．

図 7 に Ninf_call の際の通信スループットの実測値を示す．Ninf ではパケット単位でデータ転送しており，クライアントとサーバでのデータ表現の変換，データの転送が並行して起こる．従って，このスループットにはデータ表現の変換に要する時間が含まれる．

図 7 において，2MB/sec を超えた付近でほぼ飽和している 6 本のグラフは SPARC 及び Alpha と J90 間のスループットを示す．Alpha が両 SPARC よりマシン性能が高いため，やや高いスループットを示した．一方，サーバとクライアントに SPARC を用いた場合が平均約 3.2 MB/sec と最も高速になった．これは SPARC 間であればデータ表現の変換のオーバーヘッドが小さいためと考えられる．

4. マルチクライアントによるサーバ性能

通常の Ninf サーバの運用では，一つのサーバが多数のクライアントから同時に Ninf_call されることが前提となる．Ninf システムが有効に機能するには，このような場合にも平均処理時間の著しい増大を招かず，かつサーバマシンの稼働率を高く維持することが必要である．そこで，複数のクライアントを用いたサーバの性能評価を行った．

4.1 評価環境

前節の評価で用いた J90 をサーバ，Alpha クラスタをクライアントとした (**図 4**)．Linpack の求解ルーチンは前節同様，4PE 版と 1PE 版を用いる．前者は 4PE を占有して高速実行するが同時に 1task しか実行できず，後者は高速ではないが 4task 並列に実行できる．

評価には Ninf クライアントプログラムのモデルとして，Linpack Benchmark ルーチンを繰り返し呼び出すものを用意した．このモデルでは Ninf_call はステップ (s sec) 毎に一定の確率 p で発生するものとし，クライアント数は c, 問題サイズは試行の間一定で n とする．測定は，$s = 3$，$p = 1/2$, $c = 1, 4, 8, 12, 16$, $n = 600, 1000, 1400$ という条件ですべての組合せを実施した．

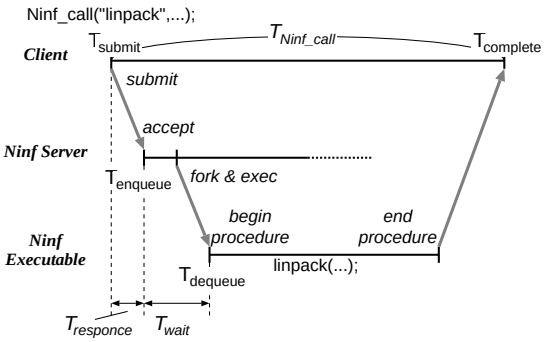

図 8 `Ninf_call` の測定のタイミング

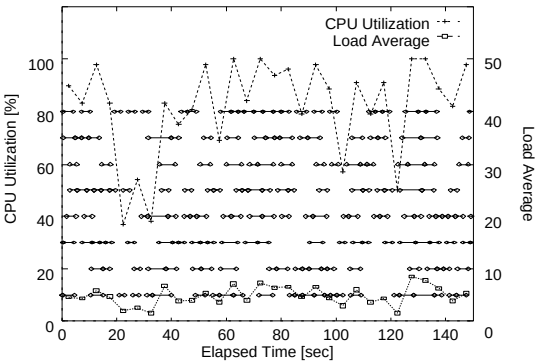

図 9 1PE 版, n=600, c=8 の実行結果

実験ではサーバの稼働率，負荷，及び各 Ninf_call について，クライアントで Ninf_call を開始した時刻 T_{submit}, サーバで Ninf_call を受け付けた時刻 $T_{enqueue}$, サーバで Ninf Excutable を起動した時刻 $T_{dequeue}$, クライアントで Ninf_call を完了した時刻 $T_{complete}$, 通信スループットを測定した (**図 8**)．ここで Ninf_call の応答時間 $T_{response}$, 待ち時間 T_{wait} を以下のように定義する．

$$T_{response} = T_{enqueue} - T_{submit} \quad (5)$$

$$T_{wait} = T_{dequeue} - T_{enqueue} \quad (6)$$

4.2 測定結果

図 9に 1PE 版で $(n, c) = (600, 8)$，**図 10,11,12**に 4PE 版で $(n, c) = (600, 8), (600, 12), (1400, 8)$ での実行状況を示す☆．横軸は経過時間 [sec]，縦軸は 5 秒毎の CPU の稼働率 [%](左) 及び負荷平均値 (右) を示す．グラフ上の水平線は各クライアントで Ninf_call を実施中の状態で，線分の長さは Ninf_call の経過時間を表す．**表 2，3**に実行結果の詳細を示す．

図 9,10を比較すると，4PE 版は負荷が高いが CPU の稼働率が高い．従ってサーバでの処理開始は遅れる

☆ 他の図表は
http://phase.etl.go.jp/~atakefu/hokke97/index.html .

n	c	Performance[Mflops] max/min/mean	$T_{response}$ [sec] max/min/mean	T_{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
	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	82.20	4.90	184
	12	66.37/9.13/32.15	5.04/0.02/0.13	0.14/0.02/0.04	2.38/0.35/1.30	97.31	8.90	272
	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	21.40	1.06	30
	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
	12	48.74/19.98/29.22	5.05/0.00/2.31	0.16/0.02/0.04	2.44/0.51/0.92	98.89	10.68	272
	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	100.00	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
	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	88.40	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
	12	44.21/22.94/32.57	5.02/0.00/0.36	0.05/0.02/0.03	1.29/0.47/0.72	100.00	12.64	152
	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	100.00	16.64	174

表 2 1PE 版のマルチクライアントによる実行結果

n	c	Performance[Mflops] max/min/mean	$T_{response}$ [sec] max/min/mean	T_{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
	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	90.00	9.98	184
	12	71.83/11.26/29.83	5.04/0.01/0.29	1.46/0.03/0.08	2.48/0.26/0.77	98.15	13.96	272
	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
	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	87.90	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
	12	42.50/19.63/28.11	5.03/0.00/0.40	0.72/0.03/0.06	0.77/0.29/0.43	99.93	17.58	272
	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
	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	96.80	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
	12	42.81/25.94/31.71	5.04/0.00/1.14	0.53/0.03/0.05	0.55/0.30/0.39	99.82	21.69	152
	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	100.00	30.29	200

表 3 4PE 版のマルチクライアントによる実行結果

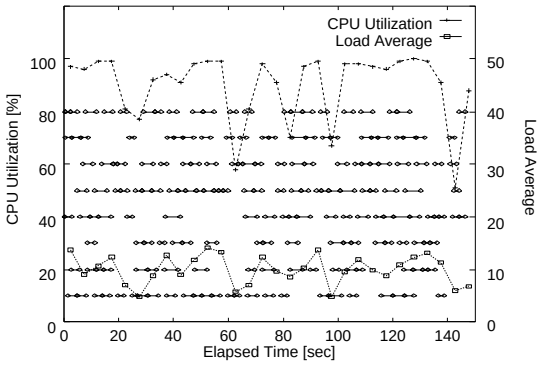

図 10 4PE 版, n=600, c=8 の実行結果

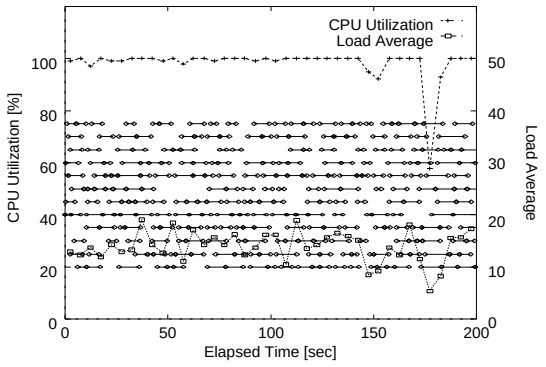

図 11 4PE 版, n=600, c=12 の実行結果

可能性があるが， n, c が小さい場合には 4CPU 版の方が良い性能が得られる． n, c が増加すると稼働率は飽和し，その後サーバの負荷が次第に増加していく．**図 11,12**を比較すると，**表 3**よりどちらもほぼ稼働率は飽和し，負荷はほぼ等しいが， c が増加していく場合は負荷が均一に上がり， n が増加していく場合は負荷の最大・最小値に差が生じる．これは比較的粒度が小さい場合は全 PE が占有される時間が短いが，粒度が大きい場合は長いことによる． 1PE 版でも同様の結果を得たが， 4task 同時に処理できるため， 4PE 版で見られる負荷の差は認められなかった．

Ninf_call の応答時間及び待ち時間は Ninf_call のタイミングにより長くなることもあるが, 平均値では n, c 及び 1PE/4PE 版による影響はあまり見られなかった．

Ninf_call の実行性能に関しては， c が少ない場合に 4PE 版では非常に高い性能が得られた． 4PE 版は計算

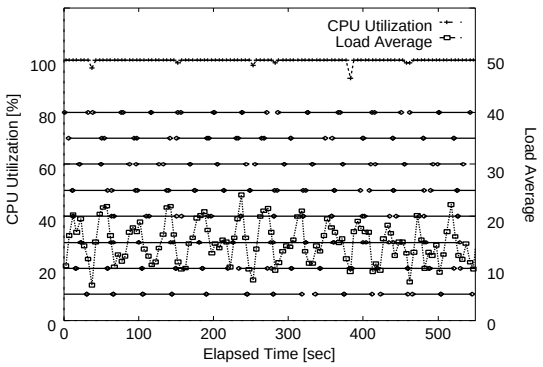

図 12 4PE 版, n=1400, c=8 の実行結果

コアの処理速度が高速なため，一時的に 4PE が占有されても Ninf_call の平均実行時間が低下しない．また c が増加すると Ninf_call の増加により通信スループットが低下し，1PE/4PE ともに実行性能が低下する．この際，4PE 版ではスケジューリングのオーバーヘッドにより 1PE 版よりも大幅に性能が低下すると予測されたが，計測では実行性能の差はあまり見られなかった．これは 4PE が占有されている場合でも通信が同時に行われていること，計算コアの実行速度の違いによるものと考えられる．

5. 関 連 研 究

ETH の Remote Computation System (RCS)[3] は複数のスーパーコンピュータを統一したインタフェースで利用するための RPC を提供する．通信レイヤに PVM を用いており広域分散に適さず，クライアント API が Ninf とは異なり拡張性に欠ける．

テネシー大の NetSolve[4] は Ninf_call に類似の API を提供するシステムであり，Agent と称するプロセスを通して負荷分散を実現する．しかし，インタフェース記述機能を備えていないため，広域分散したサーバの運用に適さない．

Legion[5] は広域分散環境での分散プログラミングを支援するオブジェクト指向言語 Mentat[6] を使って，多くの計算資源を統合した仮想計算機を実現する．Mentat を用いることで分散システム特有の最適化や機能追加が容易であるが，Ninf のように既存システムとの連続性が維持できない．

6. まとめと今後の課題

本稿では，科学技術計算分野における広域分散並列処理を支援する基盤システムを目指した Ninf システムの概説を行うとともに，システムの核となる Ninf サーバの，シングル / マルチクライアント環境での Linpack Benchmark による性能評価を行った．

3章の評価実験では，問題サイズが比較的小さい場合でも, Ninf を用いた方がローカル実行より高速化された．また, クライアントのマシン性能やプラットフォームが異なる場合においても同様の性能向上が見られ, Ninf による広域分散並列処理の現実的な有効性を実証することができた．

4章の評価実験では，最大 16 クライアントから Ninf_call を行い, サーバの負荷が最大 30 に達したが, 破綻することなく機能した．これにより実際的な運用条件での堅牢性，特にベクトルパラレルマシンに代表される，従来「計算専用マシン」として使われてきた計算機での堅牢性を確認することができた．

また，問題サイズに関わらず，クライアント数が 1 ～ 8 の閑散な状態では 4PE 版が 1PE 版を大幅に上回り, 8 ～ 16 までの繁忙な状態では 1PE 版が 4PE 版を多少上回る性能を示した．さらに, いずれの場合も応答時間及び待ち時間に関して 4PE 版が著しく不利になることはなかった．従って並列計算機システムを Ninf サーバとして利用する場合，提供するライブラリはネットワークサービスに特化する必要はなく，単体で最高性能を発揮するものを利用することが望ましい, すなわち既存の高性能ライブラリの再利用性が高いと判断できた．

一方でサーバのジョブスケジューリング，メタサーバによる Network-Wide スケジューリング等を含めた Ninf システムの有効性を検証するためには，少数の実機での実験では不十分である．今後の課題はより複雑な事象を扱えるネットワークシミュレータを作成して解析を行うことである．

謝辞 本研究を行うにあたり，ご指導並びにご討論頂いた，お茶の水女子大学細矢治夫教授，富士総合研究所西川宜孝研究員，名古屋工業大学 高木浩光助手，日本クレイ (株) つくば・東北事業所および計測環境を提供して頂いた電子技術総合研究所に深く感謝致します．

参 考 文 献

1) 佐藤ほか: Ninf: World-Wide Computing 指向のネットワーク数値情報ライブラリ, インターネットコンファレンス'96 論文集, pp. 73–80 (1996).
2) 中田, 佐藤, 関口: ネットワーク数値情報ライブラリ Ninf のための RPC システムの概要, 技術報告 TR95-28, 電子技術総合研究所 (1995).
3) Arbenz, P., Gander, W. and Oettli, M.: *The Remote Computational System, High-Performance Computation and Network*, Lecture Note in Computer Science, Vol. 1067, Springer, pp. 662–667 (1996).
4) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proceedings of Supercomputing '96* (1996).
5) Grimshaw, A. S. et al.: Legion: The Next Logical Step Toward a Nationalwide Virtual Computer, Technical Report CS-94-21, University of Virginia (1994).
6) Grimshaw, A.S.: Easy to Use Object-Oriented Paralel Programing with Mentat, *IEEE Computer*, pp. 39–51 (1993).