DO NOT DISTRIBUTE

マルチクライアントによるネットワーク数値情報システム *Ninf* の性能

概要

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た超分散並列処理の基盤を提供するソフトウェアシステムであり，高速・高性能・高品質な科学技術計算を支援することを目的に設計された．本稿では Ninf システム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両環境でのコア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マルチプロセッサシステムが 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．

Abstract

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 for High Performance Computing) software system. Basically, the Ninf is based on the server-client model. Thus, servers, distributed in the Internetwork, handle information resources either its numerical libraries or scientific constants databases, and clients are programmed by users or generated by semi-automated tools with the Ninf client API which establishes RPC connections to servers. To evaluate the Ninf system, we perform Linpack Benchmark with the Ninf RPC on Cray J90 vector-parallel supercomputer and DEC Alpha cluster of workstations, and Sun workstations. Results show that the utility and robustness of the network computing with the Ninf, and multicomputers such as vector-parallel computers and MPPs can effectively support network information services via the Ninf.

1 はじめに

科学技術計算の分野においては，より複雑な事象の解明のため，高精度かつ高信頼性のある計算をより高速で高性能・高品質に実行すること，すなわちハイパフォーマンスコンピューティング (HPC) が強く求められてきている．また近年のコンピュータネットワーク技術の発展に伴い，E-mail, FTP, World Wide Web 等で実現されるデータ資源の共有が可能となった．このようなネットワークによる情報資源へのアクセスは，物理的，論理的，時間的に透明であることが重要な要素である．一方，高速ネットワーク技術の発展を前提とすれば，CPU やストレージといった計算資源を積極的に共有することで，多くの大規模計算が可能となる．この実現には，広域ネットワーク上に展開された超広域分散に対応した計算技術の成熟を要する．こうした超分散並列計算技術を Global Computing や World Wide Computing と呼ぶ．

我々はネットワーク数値情報システム **Ninf**[1] (**N**etwork based **Inf**omation Library for Global World-Wide Computing Infrastructure) を科学技術計算分野における Global Computing を実現する基盤システムとして提案している [5, 7, 8]． Ninf は広域ネットワーク上に分散された計算資源や情報資源へのアクセスを容易に実現し， HPC を支援することを目標として設計された．Ninf はクライアント - サーバモデルに基づいており，ネットワーク上に設定された複数のサーバに対し，遠隔地のユーザは既存の言語と簡易な API を使って Ninf の機能を呼び出すだけでサービスを享受できる． Ninf が提供するサービスは高性能・高精度・高速な数値ライブラリを実行して結果を返すリモート計算ライブラリ，物理・化学・数学等の各分野の数値定数の検索，統計数値データや文献への URL Link 等のデータベース等である．

広域ネットワーク分散並列処理の前提となるのは，

通信性能 バンド幅は今後も健調な伸びを見込めるが，レイテンシの大幅な短縮は望めず，また信頼性に欠けるため冗長な通信手続きが必要．

計算性能 スーパーコンピュータや MPP のようにクライアントと比較すると常に圧倒的に高速．

ということであり，例えばワークステーションクラスタ環境で， PVM, MPI 等のメッセージパッシングライブラリによって「比較的」細粒度で実現される LAN 内での分散並列処理とは一線を画する．このような前提に即して， Ninf システムは疎粒度のリモート計算ライブラリやそれらと協調して利用するに足る付加価値を持つ情報資源を提供するとともに信頼性を保証する枠組としてメタサーバ [10] を用意する．しかし，これらが有効に機能することを実証するには実際の運用の結果を待たねばならない．

Ninf のサーバは UNIX システム上で動作しており，クライアントからのアクセスが集中しても多数のプロセスのハンドリングを破綻なく遂行する必要がある．一方， Ninf がサーバの対象としているスーパーコンピュータや MPP などの超高性能計算機の OS は，UNIX ワークステーションまたは UNIX サーバと比較して，多数のプロセスを同時に実行することを考慮されていない．これらの計算機の OS と Ninf の組合せで十

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

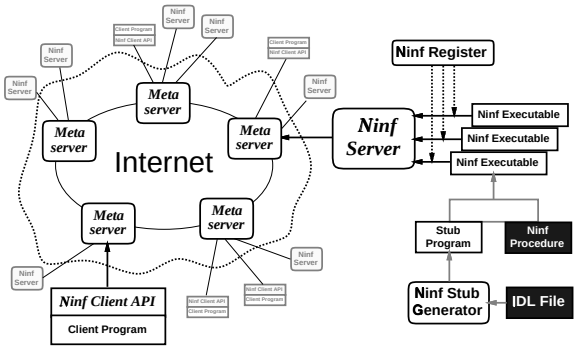

図 1 Ninf システムアーキテクチャ

分な堅牢性を維持するシステムが要求される．

また，超高性能計算機，特にベクトルパラレル計算機，MPP 等の並列計算機で Ninf サーバシステムを運用する際，順次到着する計算要求の処理方法として，2 つの選択[2]があり得る．

タスクパラレル 計算資源を空間方向に分割し，それぞれのタスクに割り当てて並列に処理．

ベクトルパラレル・超並列 すべての資源を一つのタスクに割り当てて直列に処理．

この選択は多くの場合，提供するライブラリの設計段階に関わる問題である．つまり，ネットワークサービスに特化された計算ルーチンを用意する必要があるか，あるいは単にマシンのピーク性能を実現するように最適化された計算ルーチンで有効に運用できるかが問題となる．

本稿では，Ninf システムの概要について述べるとともに，これらの諸問題に対する解答を得ることを目的として Linpack Benchmark を用いて Ninf サーバの性能評価実験を行った．本システムを用いた方がローカル実行より高速化されることにより，Ninf による広域ネットワーク分散並列処理の有効性を実証した．また，サーバマシンとしてベクトルパラレル計算機 Cray J90 4PE や Sun SMP ワークステーションを用い，複数の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した．さらに，ベクトルパラレル計算機，MPP ではピーク性能を実現するように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との判断に至った．

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 や WAN に接続されたサーバが提供するライブラリ関数を始めとする様々な資源を，サーバ上で動作するデーモンプロセスを介して利用することができる．**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

[2]計算主体の Ninf では時分割については考慮するに値しない．

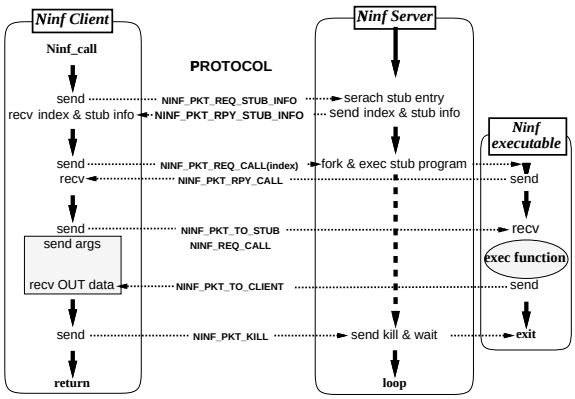

図 2 Ninf サーバ - クライアント間通信の概要

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**などから構成される．これらの構成要素間の通信は Ninf RPC (Remote Procedure Call)[9] と称する手続きによって実現される．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．以下では Ninf を実現する要素技術について述べる．

2.1 Ninf サーバ

Ninf サーバはリモート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で動作する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．提供される資源は，Ninf Executable と称するサーバ上で実行できるファイル形式を採っている．LAPACK などの著名なライブラリやホスト用に最適化された数値ライブラリを，通信ランタイムを含む stub ルーチンとリンクするだけで，容易にこれらのライブラリをリモート計算ライブラリとして提供できる．Ninf Executable は `ninf_register` コマンドで登録することでサーバデーモンの管理下に置かれ，適宜 fork&exec される．

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容易に移植できる．

2.2 Ninf クライアント API

Ninf のクライアントは，C, C++, Fortran, Java, Lisp 等の既存の言語と，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 クライアント API は Ninf_call という唯一のインターフェースから成り，Ninf の全サービスの利用に用いられる．Ninf_call の利用例として行列積を取り上げる．通常行列積のルーチンを呼び出す場合，

```
double A[n][n], B[n][n], C[n][n];
/* declaration */
dmmul(n, A, B, C);
/* calls matrix multiply, C=A*B */
```

のように記述する．dmmul が Ninf サーバで利用可能な場合は，以下のように書き換える．

```
Ninf_call("dmmul", n, A, B, C);
/* call remote Ninf library on server */
```

このようにローカルライブラリを呼び出す場合と非常に類似した手続きで Ninf の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．このため，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かのように見える．

2.3 Ninf IDL

2.1で述べたように Ninf RPC での Ninf Executable の仕様 (インタフェース情報) に関するクライアント-サーバ間の情報共有は，クライアントが最初にサーバに問い合わせることで実現される．

Ninf IDL (Interface Description Language) はこのインターフェース情報を記述する手段を提供するもので，先の行列積の例では下のようになる．

```
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 */
"dmmul is matrix multiply for double precision",
/* specify library required by this routine. */
Required "libxxx.o"
/* Use C calling convention */
Calls "C" mmul(n,A,B,C);
```

必須のインタフェース情報として引数の数，型，アクセスモード，配列の大きさが指定できる他，機能の説明，build に必要なモジュールやライブラリ，リンク時オプション，別名等が記述できる．

サービス提供者は機能コアとインタフェースを記述した Ninf IDL を用意しさえすれば，以下の手順で半自動的に Ninf サービスを追加できる (**図 3**)．

1. Ninf IDL を記述
2. Ninf IDL コンパイラ `ninf_gen` により， Makefile 及び Stub コード (インタフェース情報と通信ランタイムを含む) を生成
3. makefile を用いて Ninf Executable を生成
4. `ninf_register` コマンドで Ninf Executable を Ninf サーバに登録

2.4 メタサーバ

メタサーバは複数の Ninf サーバ情報をモニタし，クライアントからのリクエストのスケジューリングを適切に行う．メタサーバを介することでユーザは必要なサービスを提供している Ninf サーバの所在を意識せずに利用できる．複数のメタサーバ間の情報共有も可能であり，新しい Ninf サーバを用意する場合は最寄りのメタサーバに登録を行えばよい．また，適切な Ninf サーバ

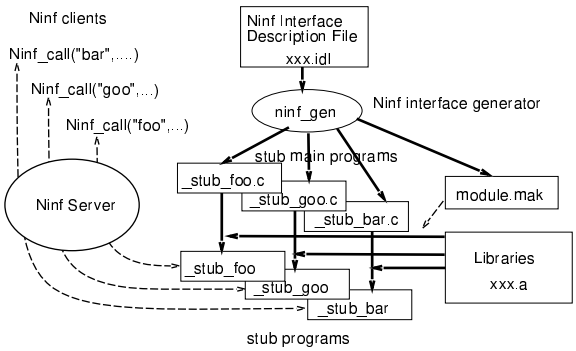

図 3 インターフェースジェネレータ

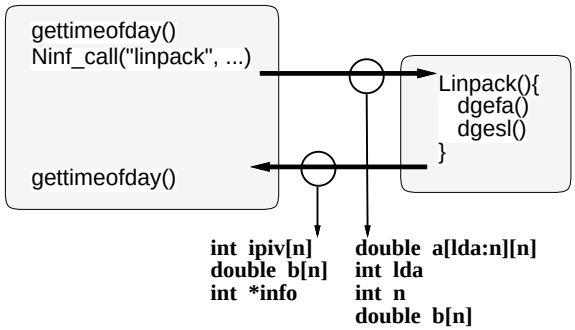

図 4 Linpack で用いたプログラムと授受データ

をクライアントに割り当てることにより， Ninf_call の処理時間を最小にし， Network-Wide な Ninf サーバ間の負荷分散を実現する．また，クライアント API の拡張により， `Ninf_check_in`, `Ninf_check_out` の間に書かれた Ninf_call を並列実行する．

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システムの有効性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もので，演算数が $\frac{2}{3}n^3 + 2n^2$ と一意に定まることから浮動小数点演算性能の標準としてよく引用されている．

Ninf を用いた Benchmark では， dgefa, dgesl の 2 つを Ninf サーバ上で実行する．プログラムの例とその間の授受データを**図 4**に示す．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．

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

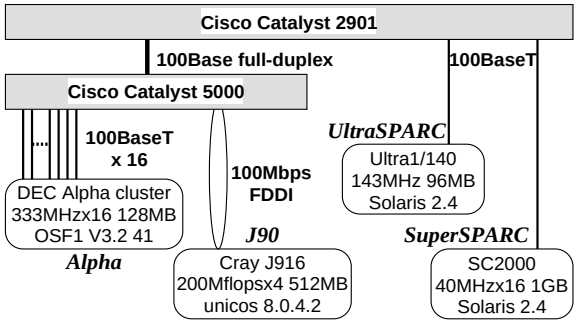

図 5 クライアント及びサーバに用いた計算機

表 1 評価に用いたサーバとクライアントの組み合わせ

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

Linpack の問題サイズを n とすると,

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

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

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. また, 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 の実行性能を示す.

また, Ninf_call の実行性能 P_{Ninf_call} は以下のように表すことができる.

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

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て通信のオーバーヘッドが隠蔽される. したがって P_{Ninf_call} はクライアントマシンの実行性能より向上することが予測できる.

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 5**に示す通りである. これらの計算機は電子技術総合研究所のものを利用した.

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 sgetfa, sgetsl を利用した. さらに 4PE を占有して高速に実行するもの (4PE 版) と, 1PE のみを用いて実行するもの (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.

計測を行ったクライアントとサーバの組合せを**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.

3.3 測定結果

図 6に SuperSPARC と UltraSPARC をクライアントとした測定結果を示す. 横軸は対象とする行列の大きさ, 即ち Linpack の問題サイズを示す. 縦軸は Ninf_call と Local の実行性能を Mflops で示している.

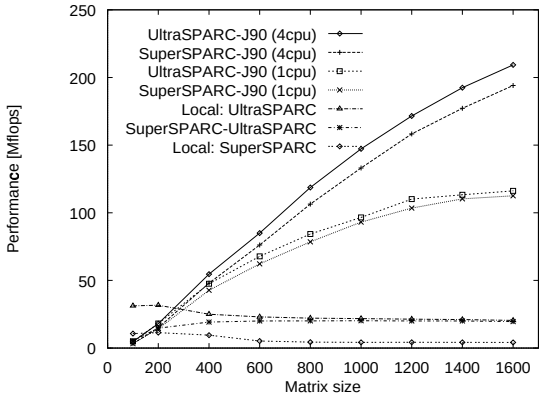

図 6 SPARC をクライアントとした Linpack の実行結果

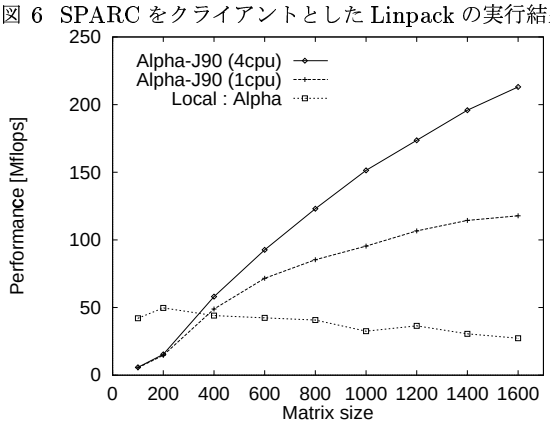

図 7 Alpha をクライアントとした Linpack の実行結果

SuperSPARC と UltraSPARC の Local はそれぞれ多少キャッシュの効果があるものの, 問題サイズによらずほぼ一定の性能を得ている. これに対し, Ninf_call では, **3.1**で述べたように問題サイズが大きくなるにつれて性能が向上し, SuperSPARC, UltraSPARC とも問題サイズが 200 ～ 400 ですでにそれぞれのマシンの Local よりも高い性能が得られている. 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 は問題サイズが大きくなるにつれ, Local の性能に収束している. 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も Ninf_call ではほぼ同程度の性能向上が観測された.

図 7に Alpha をクライアントとした測定結果を示す. Alpha の浮動小数点表現は IEEE 形式であり, CRAY 形式と異なるため, 通信用データ表現の XDR と計算機のデータ表現の間の変換に時間を要する. SPARC と同様に問題サイズ 400 前後で Ninf_call が Local の性能を上回った.

図 8 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を転送しており, クライアントでのデータ表現の変換, データの転送, サーバでのデータ表現の変換が並行して起こっている. したがって, このスループットにはデータ表現の変

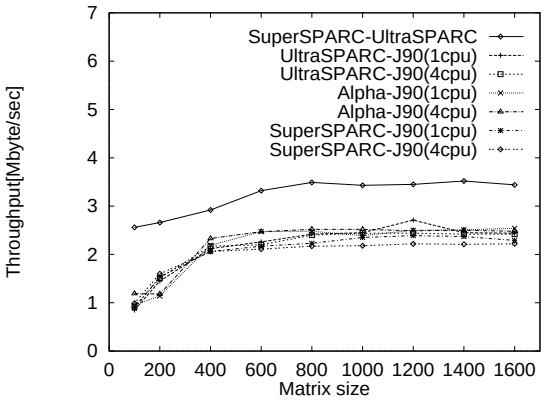

図 8 Ninf_call の通信スループット

換に要する時間が含まれる.

図 8 では，スループットの値によって 2 つのグループに分類できる．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 の間のスループットを示す．この中では 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す．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 されることが前提である．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 Cray J90 をサーバ，Alpha クラスタをクライアントとして用いた (**図 5**)．Linpack の求解ルーチンとして前節同様，4PE 版と 1PE 版を用いる．前者は 4PE を占有して高速に実行する代わりに同時に 1task しか実行できず，後者は高速ではない代わりに 4task 並列に実行できる.

評価には現実に近い Ninf クライアントプログラムのモデルとして，前節で述べた 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 とする．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.

実験ではサーバの稼働率，負荷，及び各 Ninf_call について，**図 9**に示すようにクライアントで 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 $T_{complete}$，通信及びデータ表現の変換のスループットを

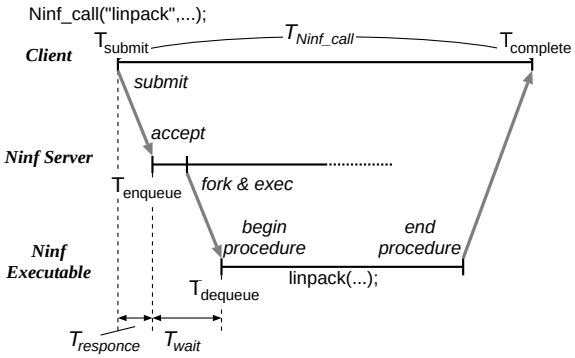

図 9 Ninf_call の測定のタイミング

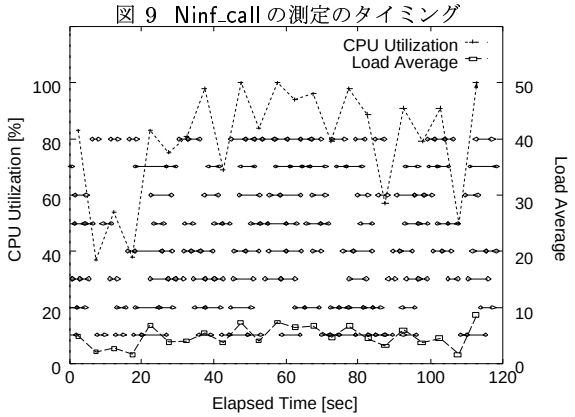

図 10 1PE 版, n=600, c=8 の実行結果

測定した．ここで Ninf_call の応答時間 $T_{responce}$，待ち時間 T_{wait} を以下のように定義する.

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

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

4.2 測定結果

図 10に 1PE 版で $(n, c) = (600, 8)$，**図 11,12,13**に 4PE 版で $(n, c) = (600, 8), (600, 12), (1400, 8)$ での実行状況を示す[3]．横軸は経過時間 [sec]，縦軸は CPU の稼働率 [%](左), 5 秒毎の負荷平均値 (右) を示す．グラフ上の水平線は各クライアントで Ninf_call を行っている状態を表す．**表 2**，**表 3**に実行結果の詳細を示す.

図 10と**図 11**を比較すると，4PE 版は負荷が高いものの CPU の稼働率が高い．したがってサーバでの処理開始は遅れる可能性があるが，n, c があまり大きくない場合には 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，その後はサーバの負荷が次第に増加していく．**図 12**と**図 13**を比較すると，**表 3**よりどちらもほぼ稼働率は飽和していき，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.

[3]他の図表は http://phase.etl.go.jp/~atakefu/jspp97/ 参照.

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	72.71/69.90/71.17	0.03/ 0.02/ 0.02	0.03/ 0.02/ 0.03	2.57/ 2.42/ 2.48	12.63	0.68	29
	4	72.01/43.85/67.06	1.01/ 0.02/ 0.06	0.04/ 0.02/ 0.03	2.53/ 2.01/ 2.36	42.71	2.01	60
	8	72.04/17.44/49.35	5.04/ 0.02/ 0.16	0.05/ 0.02/ 0.03	2.55/ 0.64/ 1.89	79.88	4.81	109
	12	66.37/ 9.13/31.26	5.02/ 0.02/ 0.18	0.14/ 0.02/ 0.04	2.38/ 0.35/ 1.23	98.20	8.96	146
	16	64.13/ 8.12/20.89	5.03/ 0.01/ 0.35	0.74/ 0.02/ 0.04	2.23/ 0.24/ 0.87	97.74	12.96	204
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	93.64/59.92/81.49	0.20/ 0.02/ 0.03	0.05/ 0.02/ 0.03	2.54/ 1.47/ 2.10	76.76	3.55	62
	8	83.91/32.63/48.43	0.40/ 0.01/ 0.03	0.05/ 0.02/ 0.03	2.39/ 0.79/ 1.39	98.92	7.35	95
	12	48.74/19.98/29.56	5.05/ 0.00/ 3.08	0.07/ 0.02/ 0.03	2.44/ 0.51/ 0.98	97.89	10.09	158
	16	27.29/15.61/20.78	5.04/ 0.01/ 0.52	0.08/ 0.03/ 0.04	0.77/ 0.35/ 0.53	100.00	16.32	58
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	109.42/75.41/93.42	5.02/ 0.01/ 0.24	0.04/ 0.02/ 0.03	2.44/ 1.80/ 2.20	88.81	4.19	67
	8	60.49/43.20/49.79	1.85/ 0.01/ 0.07	0.04/ 0.02/ 0.03	1.72/ 0.92/ 1.18	100.00	8.55	36
	12	37.12/26.07/31.98	0.22/ 0.00/ 0.02	0.04/ 0.02/ 0.03	0.96/ 0.53/ 0.73	100.00	12.81	42
	16	27.71/20.01/23.83	5.02/ 0.01/ 0.29	0.05/ 0.03/ 0.03	0.74/ 0.40/ 0.50	100.00	16.92	20

表 3 4PE 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	92.18/21.70/76.74	5.01/ 0.02/ 0.19	0.05/ 0.03/ 0.04	2.50/ 1.52/ 2.16	56.80	4.04	64
	8	89.35/26.92/52.22	0.66/ 0.01/ 0.05	0.12/ 0.03/ 0.05	2.50/ 0.60/ 1.38	87.71	9.75	111
	12	63.96/12.95/29.65	1.55/ 0.01/ 0.06	0.65/ 0.04/ 0.08	1.69/ 0.27/ 0.74	99.67	14.77	153
	16	56.73/ 9.43/18.92	5.02/ 0.01/ 0.07	1.09/ 0.04/ 0.11	1.66/ 0.21/ 0.47	100.00	19.75	200
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	134.26/61.01/93.89	0.04/ 0.01/ 0.02	0.08/ 0.03/ 0.04	2.32/ 0.98/ 1.55	88.34	5.87	66
	8	67.14/27.76/44.76	5.01/ 0.01/ 0.27	0.20/ 0.03/ 0.05	1.21/ 0.42/ 0.68	99.90	11.42	112
	12	42.50/20.22/27.85	5.02/ 0.01/ 0.12	0.72/ 0.03/ 0.06	0.66/ 0.29/ 0.41	99.99	18.04	164
	16	32.22/14.22/20.33	1.30/ 0.01/ 0.06	1.42/ 0.03/ 0.09	0.50/ 0.21/ 0.30	99.97	24.59	79
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	119.26/77.95/97.69	5.03/ 0.01/ 0.53	0.11/ 0.03/ 0.05	2.16/ 0.94/ 1.29	95.65	7.39	57
	8	59.39/34.23/47.74	0.91/ 0.01/ 0.05	0.06/ 0.03/ 0.04	0.81/ 0.43/ 0.58	99.96	15.56	32
	12	38.81/26.99/31.90	5.02/ 0.00/ 0.37	0.08/ 0.03/ 0.04	0.55/ 0.30/ 0.39	100.00	22.67	37
	16	28.06/19.27/23.21	0.63/ 0.01/ 0.03	0.89/ 0.03/ 0.06	0.34/ 0.21/ 0.27	100.00	28.97	51

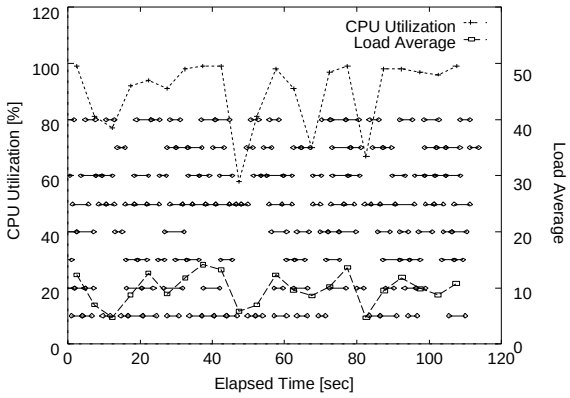

図 11 4PE 版, n=600, c=8 の実行結果

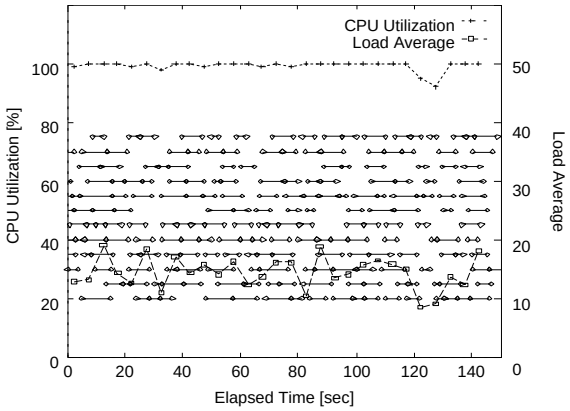

図 12 4PE 版, n=600, c=12 の実行結果

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ものがわずかにあるものの，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．これはスタブ情報の提供と Ninf Executable の fork&exec には複数のクライアントから要求がある場合にもクライアントを待たせず処理することを示す．

問題サイズ n による実行性能の傾向は，c が小さい場合は n が大きい方が通信と XDR の変換の割合が小さくなるために高い性能が得られるが，c が大きくなるにつれ 4PE が占有される時間が長くなり，通信スループットが低下する結果，全体の性能が低下する．

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 の増加による通信スループットの低下により，1PE/4PE ともに大幅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ことが予測されるが，実行性能の差はあまり見られな

表 4 SMP 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	4	4.38/3.17/3.81	1.36/1.09/1.20	0.21/0.13/0.16	1.82/0.21/0.46	20.10	6.08	67
	8	4.25/2.73/3.52	2.10/1.19/1.32	0.24/0.13/0.16	1.05/0.14/0.38	28.92	8.84	114
	16	3.77/2.08/2.83	5.54/0.00/0.35	1.23/0.14/0.20	1.40/0.11/0.35	47.36	15.37	211

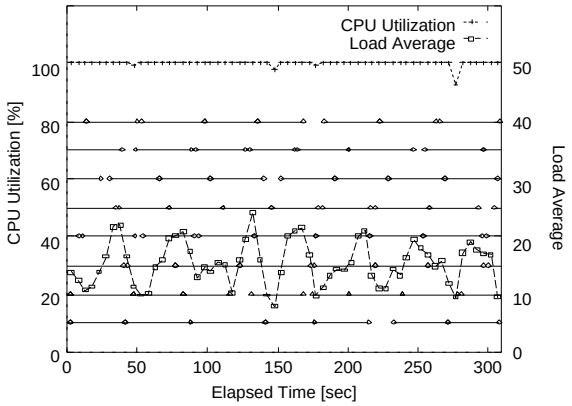

図 13 4PE 版, n=1400, c=8 の実行結果

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．

参考までに**表 4**にサーバとして Cray J90 の代わりに SuperSPARC の 16 台構成 SMP(**図 5**参照) を用いた場合の結果を示す．クライアント数の増加に伴う性能の低下が J90 に比べて小さく，応答時間及び待ち時間も増大しない．また，$c = 16$ でも CPU 稼働率に余裕を残している．これは計算性能が非常に低いために I/O ネックが顕在化せず，また複数プロセスハンドリングのオーバーヘッドが小さい UNIX サーバシステムであるためと考えられる．

5 議論

5.1 Ninf による広域分散並列処理の可能性

広域分散並列処理では，クライアント - サーバ間の通信は低速で信頼性に欠けるが，サーバとして用いる超高性能計算機の性能はクライアントを圧倒的に上回っている．Ninf はこのような環境に適した分散並列処理のために疎粒度のリモート計算ライブラリやそれらと協調して利用するに足る付加価値を持つ情報資源を提供し，また信頼性を保証する枠組も用意する．

3章の評価実験では，Ninf サーバにとって理想的な条件を与えるシングルクライアント環境において，いずれのプラットフォームでも比較的小さい問題サイズ以上であれば，Ninf を用いた方がローカル実行より高速化されるという結果を得た．しかも，クライアントのマシン性能は Ninf_call の性能にほとんど影響を与えなかった．これらの結果は，現時点で 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ものである．

5.2 Ninf サーバの実際的な運用上の堅牢性

Ninf サーバマシンとして用いられるスーパーコンピュータや MPP などの超高性能計算機の OS は，UNIX ワークステーション等の OS と比較して，多数のプロセスを同時に実行することを考慮されていないが，クライアントからの Ninf_call が集中しても多数のプロセスのハンドリングを破綻なく遂行する必要がある．

4章の評価実験では，最大 16 個のクライアントから Ninf_call を行い，サーバの負荷は最大 30 に達したが，破綻することなく機能した．これは，商用のスーパーコンピュータや MPP 上の UNIX システムに既にマルチプロセッシングを支援するに十分な信頼性があり，かつ Ninf システムがエラーに対して堅牢に作られているためである．また，当然ながら SMP システムでも堅牢かつ効率良く機能する．

5.3 タスクパラレルかベクトルパラレル・超並列か?

ベクトルパラレル機，MPP で Ninf サーバシステムを運用する際，順次到着する Ninf_call の処理方法として，計算資源を空間方向に分割し，それぞれのタスクに割り当てて並列に処理 (タスクパラレル) するか，すべての資源を一つのタスクに割り当てて直列に処理 (ベクトルパラレル・超並列) するか選択できる．

4章の評価実験の結果では，問題サイズの大小に関わらず，クライアント数が 1 ～ 8 個までの閑散な状態では 4PE 版，8 ～ 16 までの繁忙な状態では 1PE 版 の方が高い平均実行性能を示した．また，閑散時には 4PE 版の方が大幅に 1PE 版に比べて性能が高く，逆に繁忙時の 1PE 版のアドバンテージは決して大きくなかった．また，いずれの場合で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．

以上から，ベクトルパラレル計算機・超並列計算機ではピーク性能を実現するように最適化されたライブラリで Ninf サービスを提供すればよく，ネットワークサービス用に特化する必要はないと判断できる．これは既に作られたライブラリの再利用性が高いということに他ならない．

5.4 サーバのジョブハンドリングメソッド

Ninf システムが **3章**のように理想的状態で利用できない場合に要求されることは次の 2 点である．すなわち，応答時間の平均を短くすることと，CPU の稼働率を向上させることである．

4章の実験ではどの条件でも応答時間が著しく増大することはなかった．したがって，サーバの混雑状態に合わせて他サーバへ re-submit を行うなどの戦略を採用した場合にも効率を損なわないと判断できる．

Ninf_call のジョブは I/O を基準に考えれば十分疎粒度であるため，I/O ネックが顕在化しにくい．また，計算主体であることを考えると小さいタイムスライスで時分割処理されるよりは，一旦スケジュールされたジョ

ブは他のジョブに優先して最後まで遂行される方が望ましい．このような場合，広く知られているようにサーバでのジョブスケジューリングは Shortest Job First を採用するのが有効である．

現状の Ninf サーバは単純に要求が来ると即座に fork&exec するのみで First-Come, First-Serve で処理されるため，CPU に余力のある場合にそれ以上稼働率を向上させることができない．一方，Ninf_call は通常のプロセスとは異なり，例えば Ninf IDL にアルゴリズムの複雑さを記述しておくなどすることで，ジョブの runlength が比較的正確に予想できるため，サーバ内でジョブをキューイングしておき，Shortest Job First で fork&exec することは容易に実現できる上に，どのプラットフォームでも共通に採用できる．

6 関連研究

ETH の Remote Computation System (RCS)[1] 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しておらず，計算資源間の負荷分散を行うがメタサーバのように並列計算支援は行わない．また，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[2] は Ninf_call に類似の API を提供するシステムであり，Agent と称するプロセス通してメタサーバと同様の機能を提供する．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[4] プロジェクト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[3] 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 はプログラミング言語であるため，分散システム特有の最適化や機能が実現が容易である．これに対し，Ninf は特定の言語を仮定しないため，ユーザにとっては既存のシステムとの連続性が維持でき，サーバ運用上の安全性は高いと言える．

早稲田大のメタコンピュータシステム [6] は，自動並列化 FORTRAN コンパイラを用いて問題を分割し，ネットワーク上のサーバにコード輸送し，サーバ上でコードから実行形式を生成して実行するシステムである．コードの安全性，広域分散実行に適した粒度の抽出に疑問が残る．

7 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ネットワーク上の分散並列処理を実現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．その結果，Ninf による広域分散並列処理の可能性を実証するとともに，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，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のが望ましいと判断できた．つまり，既に作られたライブラリの再利用性が高いことに他ならない．

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

参考文献

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

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

[3] A. S. Grimshaw. Easy to Use Object-Oriented Paralel Programing with Mentat. *IEEE Computer*, pp. 39–51, 1993.

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

[5] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. — Ninf —: Network based Information Library for Globally High Performance Computing. *Parallel Object-Oriented Methods and Applications, Santa Fe*, 1996.

[6] Kazuyuki Shudoh and Youichi Muraoka. http://www.muraoka.info.waseda.ac.jp/sc96/, 1996.

[7] 関口智嗣, 中田秀基, 佐藤三久, 長嶋雲兵, 松岡聡. ネットワーク数値情報ライブラリ Ninf - システム実装と評価. 情報処理学会研究報告 96-HPC-62, pp. 153–158, 1996.

[8] 佐藤三久, 中田秀基, 関口智嗣, 松岡聡, 長嶋雲兵, 高木浩光. 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.

[9] 中田秀基, 佐藤三久, 関口智嗣. 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. Technical Report TR95-28, 電子技術総合研究所, 1995.

[10] 中田秀基, 草野貴之, 松岡聡, 関口智嗣, 佐藤三久. ネットワーク数値情報ライブラリ Ninf におけるメタサーバアーキテクチャ. 情報処理学会研究報告 96-HPC-60, 1996.