ハイパフォーマンス
コンピューティング 67－15
（1997. 8. 20）

高性能広域計算システム Ninf のスケジューリングに関する予備的考察

小川宏高[†1] 竹房あつ子[†2] 中田秀基[†3]
合田憲人[†4] 松岡聡[†4]

広域ネットワークの整備につれ，分散する多数の超高性能計算機を利用して大規模計算を実現する試みが我々の Ninf を含めてなされつつある．しかし，このような広域計算システムの，特に WAN 上で複数のクライアントが複数のサイトに分散している状況下での性能特性に関する議論は十分になされていない．それゆえクライアントに一定性能を保証するような広域スケジューリングの有効な手法は確立できていない．本稿では，広域計算システムのジョブスケジューリングのための予備的考察として，WAN において単一サイト / 複数サイトに複数のクライアントを設置して評価実験を行い，通信主体の計算においては通信が性能を支配すること，設置条件によって通信が性能に与える影響に一定の傾向があることを確認した．また，この結果を元にして待ち行列理論に基づいたシミュレーションモデルを提案した．さらに現在実装を進めているスケジューリングサーバ Metaserver による評価結果を示した．

Preliminary Study of Global Job Scheduling for Ninf: a High-Performance Global Computing System

HIROTAKA OGAWA,[†1] ATSUKO TAKEFUSA,[†2] HIDEMOTO NAKADA,[†3]
KENTO AIDA[†4] and SATOSHI MATSUOKA [†4]

Rapid increase in speed and availability of global-network may make global supercomputing possible, including our *Ninf* system. However, performance characteristics of these systems have been little investigated, especially under multi-clients, multi-sites situation. In order to establish methodology to schedule multiple job requests to multiple computational servers effectively and guarantee performance per each client, we conducted benchmarks under various WAN environments. And we observed communication bandwidth dominated performance for communication-intensive applications such as Linpack, and aggregate bandwidth could be sustained for multi-clients located at diffirent internet sites. Furthermore, according to these observations, we proposed simulation model based on queuing theory. And we also performed preliminary benchmarks using our scheduling server named *Metaserver*.

1. はじめに

ネットワーク技術の発展を前提とすれば，広域に分散した超高性能計算機や記憶装置等の計算資源や多様なデータ資源を活用して大規模計算の要求に応えることが可能になる．近年これを目的とした広域計算システムが Ninf[1] を筆頭に複数提案されてきており，多数のリモート計算サーバやデータベースサーバをネットワーク透過に利用したアプリケーションを Fortran, C, Java 等の言語を使って作成できる環境が整いつつある．

また，我々は Ninf 計算サーバの性能や広域計算システムの有効性に対する検討を進めている．2) では，計算性能が通信性能を圧倒的に上回る現在のネットワーク環境での基本性能を調べて広域計算システムの有用性を示すとともに，リモート計算ライブラリの設計方針や既存の高性能ライブラリの再利用性，Ninf システム自体の robustness について議論した．一方で，計算サーバはネットワーク上に分散する多数のクライアントによって共有利用されるが，各クライアントに一定の要求性能を満たすためにどの計算サーバを選択して利用させればよいのかという明確な基準がない．それゆえ計算資源の広域スケジューリングに関する有効な手法の確立には至っていない．

本稿では広域計算システムのジョブスケジューリングのための予備的考察として以下を実施した:

- WAN において，単一サイト / 複数サイトに設置された複数のクライアントから，計算サーバに対して通信パターンの異なる計算要求を行った場合の性能評価を行った．クライアントの実効性能は，計算主体 (EP) の計算の場合にはサーバの計算能力のみによるが，通信主体 (Linpack) の場合には，サーバおよびクライアントのネットワーク上の設置条件によって通信負荷の影響が支配的になり，サーバの計算能力を十分に発揮できなくなる場合もあることが確認された．

†1 東京大学 University of Tokyo
†2 お茶の水女子大学 Ochanomizu University
†3 電子技術総合研究所 Electrotechnical Laboratory
†4 東京工業大学 Tokyo Institute of Technology

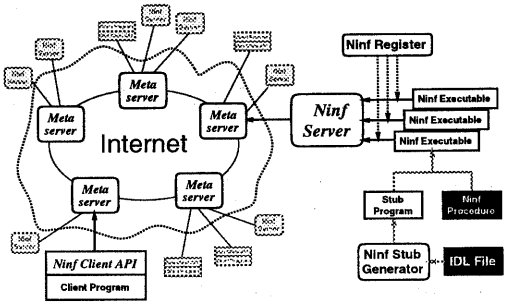

図1 Ninf システムアーキテクチャ

- 上記の考察を元にして，より一般的な設定での性能予測シミュレーションのための待ち行列モデルを提案した．
- また，現在実装を進めている広域スケジューリングサーバ メタサーバの動的負荷分散機能に関する評価実験を行い，静的なサイクリック分割に対するアドバンテージを確認した．

これらとは別にテネシー大などで開発されている広域計算システム NetSolve との「相互乗り入れ」化が計画されている．本稿では Ninf 側から NetSolve 側の計算資源を利用するためのアダプタの概要と予備実験結果についても報告する．

2. Ninf システムの概要

図1に Ninf システムアーキテクチャを示す．Ninf システムは，計算サービスを提供する **Ninf 計算サーバ**，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**，クライアントの計算要求を広域ネットワーク上の複数の計算サーバへスケジューリング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 Remote Procedure Call の拡張で Ninf RPC[3] と称する手続きで実現される．各構成要素の概要を示す:

Ninf 計算サーバ Ninf 計算サーバは計算ライブラリ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 と称する実行形式を採り，サーバはこれらを適宜 fork&exec 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，半自動的に生成される．

Ninf クライアント API Ninf のクライアントは，Fortran, C, Java 等の言語と各言語用の 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 サーバに計算要求を行う．行列積ルーチンを呼び出す例を示す:

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

このようにローカル関数呼出しと類似した手続きで Ninf 計算サーバを利用できる．Ninf_call は内部的に Ninf RPC を用いて自動的に関数の引数の数・型を決定し，Ninf Executable と相互に通信して引数や計算結果のやり取りを行う．従ってユーザには手元の計算機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．他には非同期呼出し Ninf_call_async，並行に実行するトランザクション区間の指定 Ninf_begin_transaction, Ninf_end_transaction 等の API が用意されている．

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，クライアントの計算要求のスケジューリング，動的負荷分散を行う．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．また，メタサーバはトランザクションによる並行実行を支援する．Ninf_transaction_begin,Ninf_transaction_end の間に記述された Ninf_call の列に対して引数のデータ依存グラフを生成し，必要な同期を取りながら，独立な Ninf_call を並行に実行する．

3. WAN に分散するクライアントによる計算サーバの評価

Ninf 計算サーバの一般的な運用では，一つのサーバは WAN に分散する多数のクライアントから同時に Ninf_call され，共有利用される．このような条件下で各クライアントに一定の要求性能を満たすために，どの計算サーバを選択して利用させればよいのかという明確な基準が現状ではない．従って計算資源の広域スケジューリングに関する有効な手法の確立に至っていない．

そこで，WAN において，単一サイト / 複数サイトに設置された複数のクライアントから，計算サーバに対して通信パターンの異なる計算要求を行った場合の性能評価を行った．評価には Linpack Benchmark と NAS Parallel Benchmark の EP を用いた．Linpack は通信のオーバーヘッドが (比較的) 顕在化しやすいという性質を持つ．一方，EP は通信量一定で計算量を任意に設定できるため，通信の影響はほとんど現れない．

3.1 評価方法

Ninf クライアントプログラムのモデルとして，Linpack および EP の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，問題サイズは試行中一定とする．測定は，$s = 3$, $p = 1/2$, $c = 1, 2, 4, 8, 16$ という条件で実施した．

実験ではサーバの稼働率，負荷，通信スループットを測定した．

3.2 Linpack Benchmark による評価

Linpack 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プログラムである．Benchmark では，LU 分解 (dgefa)，後退代入 (dgesl) を計算サーバで実行する．引数を含む通信量は $8n^2 + 20n + O(1)$ bytes，演算数は $2/3n^3 + 2n^2$ である．従って Ninf_call の所要時間を T_{Ninf_call} とすると Ninf_call の実効性能は $P_{Ninf\_call} = \frac{2/3n^3+2n^2}{T_{Ninf\_call}}$[flops] と表せる．

評価では問題サイズを $n = 600, 1000, 1400$ とした．

3.2.1 単一サイトを用いた WAN での測定結果

単一サイトを用いた評価では，サーバには電総研の Cray J90 (200Mflops×4, 512MB) を用い，クライアントにはお茶大の SuperSPARC(50MHz×2, 96MB) 8台構成の WS クラスタを利用した．クライアント - サー

n	c	性能 [Mflops] max/min/mean	スループット [MB/s] max/min/mean	サーバ稼働率	サーバ負荷
600	1	8.00/7.24/7.68	0.166/0.153/0.161	6.85	0.40
	2	6.05/1.51/3.77	0.124/0.030/0.077	6.90	0.42
	4	3.57/0.92/2.46	0.073/0.019/0.051	8.94	0.50
	8	1.52/0.49/1.08	0.031/0.011/0.022	7.99	0.49
	16	0.90/0.29/0.54	0.018/0.006/0.011	8.17	0.49
1000	1	12.75/7.26/10.50	0.160/0.091/0.131	6.43	0.39
	2	8.22/5.89/7.15	0.101/0.072/0.088	9.23	0.52
	4	4.80/1.46/3.97	0.061/0.018/0.049	9.39	0.53
	8	2.42/0.97/1.82	0.029/0.012/0.022	8.71	0.53
	16	1.36/0.47/0.88	0.016/0.006/0.011	8.75	0.49
1400	1	18.46/11.87/16.42	0.166/0.104/0.147	8.15	0.47
	2	10.44/8.03/9.34	0.092/0.070/0.082	13.58	0.73
	4	7.00/2.00/5.50	0.061/0.017/0.048	15.61	0.81
	8	2.91/1.42/2.46	0.025/0.012/0.021	9.20	0.56
	16	1.59/0.79/1.25	0.014/0.007/0.011	9.35	0.54

表 1　WAN での単一サイト，マルチクライアントによる実行結果

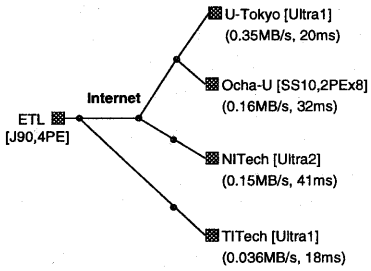

図 2　WAN の計測環境

バ間の FTP 通信スループットは 0.17MB/s である．

表 1 に実行結果を示す．n が大きくなると，通信量に対して計算規模が比較的小さいため，WAN においても Ninf_call の性能は向上する．

一方 c が増加する場合は，クライアント - サーバ間の通信スループットの低下が著しい．これはクライアント数が 1〜2 と少ないときにクライアント - サーバ間のバンド幅が飽和してしまい，時間あたりに到着する Ninf_call の数が抑制されてしまうためである．このことは c=16 に達しても稼働率，負荷ともに余裕を残したことからも見て取れる．

3.2.2　複数サイトを用いた WAN での測定結果

複数サイトを用いた評価では，クライアントとしてお茶大 (Ocha-U)，東大 (U-Tokyo)，名工大 (NITech)，東工大 (TITech) の 4 サイトのマシンを利用した．サーバには電総研の J90 を用いた．各サイトで用いたマシン，クライアント - サーバ間の ping のレイテンシ，FTP 通信スループット，WAN の接続の状態を図 2 に示す．各サイトでのクライアント数は 1,4 とした．

図 3 に 4 つのサイトを用い，$c = 1$(上), $c = 4$(下) としたときの実行結果を示す．比較のため，単一サイト (ETL-Ocha-U 間) での実行結果も示した．横軸には問題サイズとクライアント数 cx サイト数を示し，縦軸にはそれぞれ，Ninf_call の実効性能と通信スループットの総和を示している．

複数サイトを用いた場合の ETL-Ocha-U 間の通信スループットは単一サイトのみの場合と比較すると，$c = 1$ のときで 9%〜18%，$c = 4$ のときで 18〜44% しか低下していなかった．よって，通信スループットの総和は複数サイトからジョブが投入される場合，各クライアント - サーバ間で 1 対 1 のときに観測される通信スループットより大幅に上回ることが分かる．サーバの CPU の稼働率，負荷平均値についても複数サイトの方が高い値となっていた．

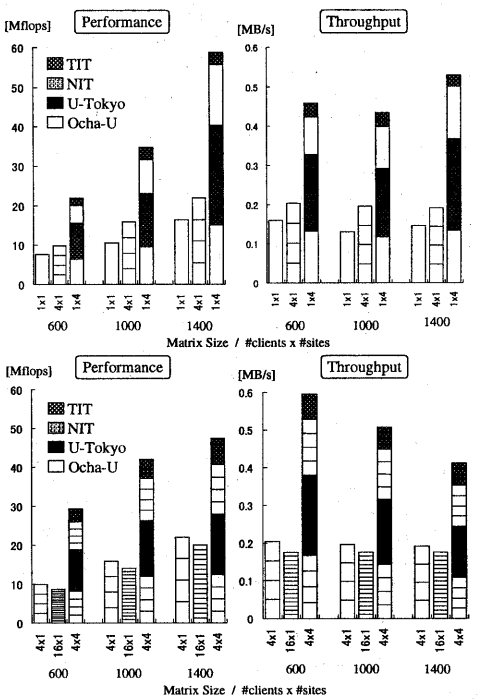

図 3　WAN での複数サイト，マルチクライアントによる実行結果

複数サイトで $c = 1$ と $c = 4$ の場合を比較すると，通信スループットのばらつきによる影響があるものの，各クライアントでの総合性能の差は少なく，WAN においては 1 対 1 の通信性能により性能が制限されることが分かった．

また，一般にクライアント数と問題サイズの増加によりサーバの計算性能が飽和してしまうと予測されるが，CPU の稼働率は $c = 1$(計 4) のときで 20.2%〜30.3%，$c = 4$(計 16) のときで 27.1%〜34.2% に留まっており，負荷平均値についても同様であった．従って，通信量の多いジョブではネットワークの負荷により性能が決まるため，サーバの稼働率，負荷平均値のみを考慮しても適切な負荷分散が実現できないことが分かる．

3.3　EP による評価

EP(An embarrassingly parallel becshmark) は NAS Parallel Benchmark のカーネルプログラムの 1 つで，乗算合同法により一様乱数及び正規乱数を生成する典型的なモンテカルロシミュレーションである．

Ninf を用いた Benchmark では，乱数生成の試行をサーバ上で行う．通信量は問題サイズによらず一定で $O(1)$bytes である．演算量は乱数を生成した回数とし，試行回数が 2^n のとき 2^{n+1} となる．従って Ninf_call の所要時間を T_{Ninf_call} とすると Ninf_call の実効性能は $P_{Ninf\_call} = \frac{2^{n+1}}{T_{Ninf\_call}}$[ops] と表せる．

評価では 2^{24} 回 (Sample) の試行を行った．EP ルー

	c	性能 [Mops] max/min/mean	通信時間 [sec] max/min/mean	サーバ稼働率	サーバ負荷
LAN	1	0.17/0.17/0.17	0.01/0.01/0.01	30.51	1.44
	2	0.17/0.17/0.17	0.01/0.01/0.01	53.86	2.39
	4	0.17/0.16/0.17	0.01/0.01/0.01	98.18	4.18
	8	0.09/0.08/0.08	0.01/0.01/0.01	100.00	8.90
	16	0.04/0.04/0.04	0.02/0.01/0.01	100.00	16.88
WAN	1	0.17/0.17/0.17	0.06/0.04/0.04	25.02	1.21
	2	0.17/0.16/0.17	0.06/0.04/0.04	49.16	2.19
	4	0.17/0.16/0.17	0.24/0.04/0.05	98.14	4.16
	8	0.09/0.08/0.08	0.05/0.04/0.04	100.00	8.91
	16	0.04/0.04/0.04	0.09/0.04/0.04	99.94	16.88

表 2 EP の LAN/WAN でのマルチクライアントによる実行結果

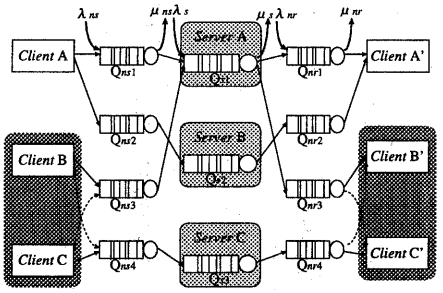

図 4 広域計算システムのモデル

チンは 1PE で実行し，J90 では 4 ジョブ並行に処理できる．

3.3.1 測定結果

表 2 に EP の LAN/WAN でのマルチクライアントによる実行結果を示す．4PE 構成の J90 に対し，各 Ninf_call は 1PE で処理されるため，クライアント数 c が 4 までは Ninf_call の性能が維持され，$c = 4$ のときにサーバの稼働率がほぼ 100% に達した．一方，c=8, 16 では Ninf_call の性能が 1/2, 1/4 となっていた．これは EP では通信量が少ないため，LAN/WAN ともに Ninf_call の性能はほとんどサーバの稼働状態のみに依存するためである．また，複数サイトを用いた場合でも同様の結果が得られると考えられる．

4. 待ち行列による広域計算システムのモデル化

前節では広域計算システムの特定の利用条件での評価を行った．しかし，広域ジョブスケジューリング手法の実現に必要とされるような，より一般的な設定条件での性能予測のためには，実機によるベンチマークのみでは不十分である．また今後の広域ネットワークの整備を前提とした議論も困難であると言わざるを得ない．

これに対し，我々は広域計算システムの単純なモデル化を行って，シミュレーションによるシステムの挙動予測を試みている．以下では待ち行列を用いてネットワークや計算サーバを抽象した単純なモデルの説明を行う．

4.1 シミュレーションモデルの概要

図 4 に我々が提案する広域計算システムのシミュレーションモデルの概要を示す．図中の Client A と A', B と B', C と C' は同じクライアントを表している．

このモデルではクライアントから計算サーバへの通信，サーバでの処理，サーバからクライアントへの通信をそれぞれ待ち行列 $Q_{ns_i}, Q_{nr_i}, Q_{s_i}$ とする．

$\lambda_{ns}, \lambda_{nr}$ はネットワークを流れる Ninf_call 以外の要求の到着率，即ちネットワークの混雑度や遅延時間を表し，ポアソン到着であると仮定する．μ_{ns}, μ_{nr} は待ち行列の要求のサービス率，即ちネットワークのバンド幅を表し，$1/\mu_{ns}, 1/\mu_{nr}$ は指数分布に従うものとする．

λ_s, μ_s についても同様で，それぞれサーバの混雑度，サーバの処理性能を定める．

また，図 4 において，Client B と C は同一サイト内にあるものとする．前節での結果から，同一サイト内にある複数のクライアントは通信路を共有するため，クライアント数に反比例するハンド幅しか利用できない．従って，Client B から Server A に対して投じられる要求は，Client C から Server C の間の待ち行列にも投じられて通信に遅延をもたらす．逆も同様である．

現状はこのモデルを市販のシミュレーションパッケージ上での実現を試みている段階である．

5. メタサーバによる動的負荷分散機能の評価

並列処理に関する知見として，高速化のために計算負荷を PE に対して均等に分散し，特に負荷が静的に予測不能な場合は動的な負荷分散を行う，というものがある．我々はこの手法が広域計算システムにおいても有効であることを確認するために，Ninf メタサーバに動的負荷分散機能を実装して評価実験を行った．

5.1 動的負荷分散機能の概要

以下ではトランザクションによる並列実行の機構とそれを利用したメタサーバによる動的負荷分散の実現方法について説明を試みる．

5.1.1 トランザクションによる並列実行

トランザクションとは一連の計算をまとめて並列実行を行うための枠組である．ユーザによって明示的に指定されたトランザクション区間内の Ninf_call の列に対して，引数のデータ依存関係が自動的に解析され，この関係を満たしながら独立な Ninf_call を並行実行する．

行列 A, B, C, D の加算の例をを以下に示す:

```
Ninf_transaction_begin();
Ninf_call("madd", n, A, B, E);
Ninf_call("madd", n, C, D, F);
Ninf_call("madd", n, E, F, G);
Ninf_transaction_end();
```

A と B，C と D の加算は同時にできるので，自動的に 2 つの加算は並行に実行され，両方が終了した時点で E と F の加算が実行される．

クライアントにとっては，1 つのトランザクションは 1 つの Aggregate された Ninf_call と見なせる．トランザクションの内部では Ninf_call のセマンティクスは変化しており，実際の計算要求は行わず，ただデータ依存グラフの構築のみを行う．

Ninf_transaction_end に到達した時点でデータ依存グラフの構築を終え，データ依存グラフとトランザクション内の Ninf_call の列をメタサーバに送る．メタサーバはデータ依存グラフに従って Ninf_call をスケジューリングして Ninf 計算サーバに対して計算要求を行う．すべての計算が完了するとメタサーバはクライアントに必

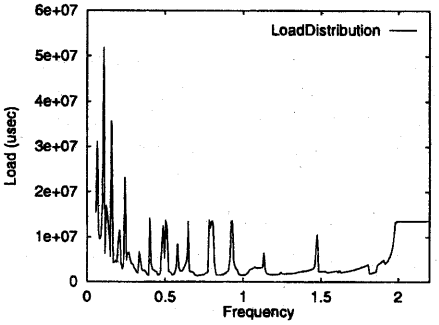

図5 状態密度計算の負荷分布

要な値を書き戻す．

5.1.2 動的負荷分散機能の実現方法

メタサーバは複数の Ninf 計算サーバを管理しており，それぞれの計算サーバで現在実行中のジョブ数も把握している．動的負荷分散機能は，この情報を用いて各計算サーバで実行するジョブの数を制限し，一定以上の発行を抑制することで実現されている．抑制されたジョブはメタサーバ上の待ち行列にキューイングされる．

ある計算サーバで実行中のジョブが完了すると，待ち行列の中からそのサーバで実行可能な最初のジョブが取り出され，その計算サーバに計算要求を行う．現在のメタサーバは Java で記述されており，この機能は Java の wait/notify を用いて実装されている．

5.2 評価環境

実験環境としては，DEC Alpha (333 MHz) 33 台の WS クラスタを用いた．各 WS は，100Base/T のスイッチで結合されており，1 台をメタサーバホストに，残りの 32 台を計算サーバとして利用した．

対象プログラムとしては状態密度計算 (density of states)[4] を用いた．この計算は巨大分子の状態密度を，行列の対角化を行うことなく，求めるもので，巨大行列をバネのつながった力学モデルに見立て，周期的な外力を与えてバネの共鳴の度合いを状態数とするものである．この問題は外力の周波数に対して並列化を行うことができるため並列計算に適しているが，各周波数における計算負荷が一定でないため，均等に負荷を分散することは困難である．

図5 に負荷の分布を示す．横軸は周波数，縦軸はその周波数の共鳴度合いを計算するための計算時間である．負荷が周波数によって著しく異なるため，周波数に対して単純にブロックやサイクリック分割しても負荷は均等に分割されず，高速化が期待できない．さらに，各周波数における負荷は事前には予測不可能である．今回の実験では 256 の周波数に対して計算を行った．

5.3 計測結果

表3 に実行時間を示す．分割数は 256 の周期を幾つの Ninf_call に分割したかを示している．32 分割の場合は各々 8 つの周期に対して一括して計算する Ninf_call を 32 回実行する．この分割はサイクリックに行った．

参考のため，単純にサイクリックに 32 分割して静的に負荷分割を行った場合とコールバックを用いて動的負荷分割を行った場合[5] の結果も併せて示す．

1 台での実行をみると，分割数が増大するにつれ実行時間が大きくなっている．これはメタサーバで複数の Ninf_call を一まとめにするためにかかるオーバーヘッドが分割数に比例して増大するためである．

計算サーバ数		1	8	16	32
分割数	32	1394.8	195.5	111.9	109.7
	64	1408.5	196.1	107.9	78.2
	128	1430.0	206.1	113.1	69.2
	256	1489.4	234.6	131.9	85.9
単体実行		1339.0	—	—	—
サイクリック		1380.4	233.8	165.3	94.4
コールバック		1385.9	179.8	91.3	52.9

表3 各種負荷分散手法による状態密度計算の実行時間 [秒] の比較

台数が増大するにつれ，実行時間は単調に減少している．同じ台数で分割数を変化させた場合には，台数に応じてそれぞれ最適な分割数が存在する．8 台で 32 分割，16 台で 64 分割，32 台で 128 分割が最適である．分割数を増やすと粒度が小さくなってより適切な負荷分散が可能になるが，オーバヘッドも増大するため，実行時間が短縮されるとは限らない．1 台あたりの分割数がある程度あれば負荷分散には十分であり，それ以上に増やしてもオーバヘッドが増えるばかりで速度の向上には寄与しない．今回の結果では，1 台あたり 4 分割程度が負荷分散の効果とオーバーヘッドの均衡点となった．

また静的なサイクリック分割との比較では，負荷分散のオーバーヘッドを上回るアドバンテージを得た．

コールバックによる動的負荷分散と比較すると，メタサーバを用いる手法はコストが大きい．これには以下の原因が考えられる:

- コールバックを用いる場合は台数分の Ninf_call しか行われないのに対し，メタサーバを用いる場合には分割数分の Ninf_call が行われる．
- メタサーバを用いる場合，全ての Ninf_call のデータがメタサーバでバッファリングされ，データ転送が完了するまで個々の Ninf_call が発行されない．
- 終了時も同様に、すべての Ninf_call が終了するまでクライアントへのデータ転送が行われない．

このうちの 1 番目は本質的で不可避であるが，残りの 2 つは今後の改良によって回避できる性質のものである．

またプログラミングの容易さから言えば，コールバックによる負荷分散を行うには特殊なプログラミング手法に従って記述する必要があるのに対し，メタサーバを用いる手法は負担がなく，望ましいと言える．

6. Ninf-NetSolve アダプタ

NetSolve[6] はテネシー大などで Jack Dongarra らによって開発されている広域計算システムである．NetSolve の設計目的は Ninf と非常に近く，システムとしての構成 / 機構もまた類似している．従って，相互のシステムの用意した計算ライブラリを利用することができれば，双方のシステムのユーザのメリットになる．これを可能にするためにはプロトコル / データを変換するアダプタを実現する必要がある．

6.1 NetSolve の概要

NetSolve の構成を図6 に示す．Ninf のメタサーバに相当する，Agent と呼ばれるサブシステムがサーバの情報を管理している．メタサーバが代理サーバとして

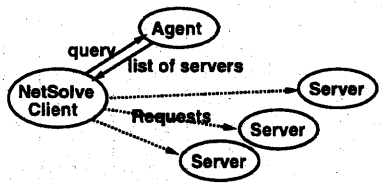

図6 NetSolve の構成

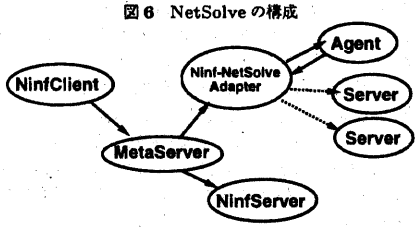

図7 Ninf-NetSolve アダプタの構成

実行時間 [秒]	50	100	150	200
ネイティブクライアント	8.2	18.4	33.3	72.1
アダプタ	8.5	21.7	38.8	82.4
オーバーヘッド	0.3	3.3	5.5	10.3

表4 Ninf-NetSolve アダプタの性能

機能するのに対し，Agent はクライアントの要求に応じてその計算を実行可能なサーバのリストを提供する紹介者として機能する．クライアントはリストの中から適当なサーバを選択し，実際の計算を依頼する．

6.2 Ninf→Netsolve アダプタの実現

Ninf と NetSolve の相互運用を行うためには双方向のアダプタが必要であるが、今回は Ninf クライアントから NetSolve の計算ルーチンを呼び出すためのアダプタを開発した．このアダプタは，図7に示すように，Ninf においては計算サーバの一つとして，NetSolve においてはクライアントの一つとして機能する．

本アダプタは Java で記述されている．アダプタは以下の役割を担う．

- NetSolve の計算ルーチンのインターフェイス情報を Ninf の情報へ変換
  NetSolve のインターフェイス情報は Ninf のものとほぼ等価だが，構造が全く異なるので変換が必要．
- 転送されるデータの構造変換
  NetSolve は主なクライアント記述言語として Fortran を，Ninf では C を想定しているため他次元配列の格納順が異なる．このため，データを一時的にキャッシングし並べ替えて呼び出すことが必要．

6.3 予備評価

アダプタの性能を評価するために予備的な実験を行った．NetSolve の Agent としては comet.cs.utk.edu に設置されているパブリックな Agent を用い，NetSolve ネイティブのクライアントを用いる場合と Ninf クライアントからアダプタを介して実行する場合を比較した．NetSolve では実際に計算が行われる計算機を直接指定することはできないが，今回の実験では双方とも dancer.cs.utk.edu で実際の計算が行われた．

計算ルーチンとしては LAPACK の一部で，Linpack benchmark とほぼ等価な degsv を用いた．問題サイズは 50,100,150,200 とした．

アダプタを用いた場合のオーバーヘッドが対象となる行列サイズに従って大きく増加することが分かる．このコストの多くは引数となる配列の並べ換えに費やされていると思われる．並べ換えは Ninf と NetSolve の本質的な差異から来るもので不可避であるが，実装の工夫によってコストを低減できると考えられる．

7. まとめと今後の課題

本稿では広域計算システムのジョブスケジューリングのための予備的考察として，WAN において単一サイト / 複数サイトに設置された複数のクライアントから，計算サーバに対して計算要求を行った場合の性能評価を行った．その結果，通信主体の計算では，クライアントの設置条件によって通信負荷の影響が支配的になり，サーバの計算能力を発揮できないことが分かった．従って，ジョブスケジューリングにおいてサーバの負荷や稼働率だけを考慮すると，重大な資源の枯渇を招く．また，より一般的な設定での性能予測シミュレーションのための待ち行列モデルを提案した．

さらに，現在実装を進めている広域スケジューリングサーバ メタサーバの動的負荷分散機能に関する評価実験を行い，静的なサイクリック分割に対するアドバンテージを確認した．

最後に Ninf からテネシー大などで開発中の NetSolve の計算資源を利用するためのアダプタの概要と予備実験結果についても報告した．

今後の課題は性能予測シミュレーションを実際に行うこと，メタサーバ，Ninf-NetSolve アダプタの完成度を向上することである．

謝辞 本研究を行うにあたり，ご指導並びにご討論頂いた，高木浩光助手 (名工大)，関口智嗣氏，建部修見氏 (電総研)，佐藤三久氏 (RWC)，長嶋雲兵教授 (お茶大) に深く感謝致します．

参考文献

1) Sato, M. et al.: Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture, *Proceedings of HPCN'97 (LNCS-1225)*, pp. 491–502 (1997).
2) 竹房ほか: 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, *Proceedings of JSPP'97*, pp. 273–280 (1997).
3) 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).
4) 日向寺ほか: 大規模実対称行列の状態密度の計算とその並列化, 情報処理学会研究報告 93-HPC-48, pp. 57–64 (1993).
5) 中田ほか: Ninf による広域分散並列計算, *Proceedings of JSPP'97*, pp. 281–288 (1997).
6) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proceedings of Supercomputing '96* (1996).