2009－AL－122 (6)
2009／1／30

Ninf-G 上の分散並列計算システムの開発

内藤 昌彦† 築地 立家†
†東京電機大学 理工学研究科

Ninf-G はネットワーク環境下における分散処理システムを構築する API モジュールで，Version5.0 になり単体で動作することが可能になり，以前インストールする必要があった Globus Tool Kit が不要となった．Ninf-G を使用したシステムの多くは分散処理起動時に与えた静的パラメータで動作するオープンループ形式の処理で，動的パラメータには対応していなかった．今回，分散コンピューティングで Grover の量子アルゴリズムを古典コンピュータでシミュレートするため，Linux の API と Ninf-G の API を使用することで Client-Server 間通信を実現し，動的なパラメータを与えることに対応したシステムを開発した．

Development of Distributed Parallel Computing Systems with Dynamic Parameters on Ninf-G

Masahiko Naito† Tatsuie Tsukiji†
†Tokyo Denki University. The Science and Engineering graduate course

Ninf-G is an API module for building distributed computing systems over open global networks. It has worked as a stand-alone product after the Version5.0; before, it required Globas Tool Kits. Most of the previous Ninf-G applications assumed an open-loop distributed computation, where each computation has only starting parameters, sharing no dynamic parameters passed between them. In this paper, in order to simulate a Grover's quantum algorithm by many classical computers connected over open networks, we develop a system on Ninf-G sharing dynamic parameters between different computers. To achieve it, we realize a communication between Client and Server by developing a Client-Server system combining Ninf-G API with Linux API.

1. はじめに

分散コンピューティングは，ネットワーク環境に配置された遊休中の PC さえあれば，ソフトウェアのみで構成可能な計算環境である．これにより，スーパーコンピュータの様な新たなハードウェアを用意せずとも，それと同等の計算環境を安価かつ高信頼性で実現出来るため，科学技術計算等への応用が徐々に増えつつある．実際，SETI@Home (http://setiathome.berkeley.edu/) では，100 万台近い PC ユーザーの協力による分散コンピューティング環境で 300TFLOPS に上る計算能力を確保している．また，DNA 解析のための分散コンピューティングソフトウェアが，多数開発・販売されている．ところで，これらの分散システムは，スタート時に与えた静的パラメータだけで計算を実行し，結果を得た段階で呼び出し元へ返却する，いわゆるオープンループ形式の分散計算処理となっている．

本稿では，分散コンピューティング環境における量子コンピュータシミュレータを提案する．一般に，量子コンピュ

ータを古典コンピュータでシミュレーションする場合，N ビット量子コンピュータのメモリは 2^N 個のインデックスを持つため，N が大きくなると演算のため使用するメモリが指数的に増大し，メモリオーバを発生してしまう．すなわち，ある程度大きなNに対して，量子コンピュータを1台のPCでシミュレートすることは不可能である．そこで，量子コンピュータのインデックス空間を，ネットワークを介した複数のPC に分散させることにより，量子コンピュータのシミュレーションを行う．その際，スタート時の静的パラメータだけでなく，各 PC において計算途中で生じる動的パラメータも相互参照しながら，分散計算を進める必要が生じる．そこで，計算途中で各 PC に発生する動的パラメータを収集・配布することが可能な，新しい分散計算処理システムを提案することにより，量子コンピュータシミュレータを実現する．

現在，分散計算環境を構築するめに提供されている主だったミドルウエアは，以下の通りである．

Globus Toolkit：Globus Alliance が管理するグリッドコンピューティング環境構築用のミドルウェア．現在では事実上の標準となっている．但し Globus Toolkit のみでは分散コンピューティング環境を構築出来ない．(http://www.globus.org)

Sun Grid Engine：Sun Microsystems が開発したオープンソースの Toolkit．(http://www.sun.com/software//gridware/)

Xgrid：Apple 社が開発した Mac OSX 用ツールキット．(http://www.apple.com/server/macosx/technology/xgrid.html)

SCore：日本で開発されたグリッドコンピューティングツールキット．開発団体である新情報処理開発機構は既に解散．(http://www.pccluster.org/)

AD-POWERs：大日本印刷が開発した Windows 用グリッドコンピューティングミドルウェア．LAN 環境下で動作する．(http://www.dnp.co.jp/cmc/ad-powers/)

PromotionalGRID：ブランドダイアログ社が独自に開発したグリッドコンピューティングミドルウェア．(http://branddialog.co.jp/service/pgs/)

Ninf-G：本論文の次章参照．(http://ninf.apgrid.org/)

本稿では，Version5.0 になった Ninf-G を使用して，分散コンピューティング実験環境を構築する．他のミドルウェアと同様，Ninf-G が用意する API も，起動時パラメータのみを受け付ける関数が主であり，実行中の動的パラメータを継続的に監視・処理するような関数はほとんど見受けられない．そこで，Client-Server ネットワーク環境の下で，同期・非同期呼び出し等の GridRPC を行う API 関数を巧みに利用することにより，動的パラメータを常時監視し，必要に応じて処理することが出来る分散計算処理を新たに実現する．

2. Ninf-G の概要

Ninf-G は，当初，Globus Toolkit の元で動作する分散コンピューティングミドルウェアとして，産業総合研究所及び東京工業大学により開発されたが，最近，Version5.0 になり，Globus Toolkit なしで動作するようになった．Ninf-G の主な特徴及び機能を以下に示す．

2.1 Ninf-G の特徴

- Open Source であり，誰でも入手可能である．
- 学習が容易であり，短時間で分散コンピューティング環境を構築可能である．
- 分散された処理の再利用が可能．
- 各 Server に別々の処理を割り当てることが可能．

2.2 Ninf-G の機能(Version 5.0 の場合)

- SSH 等を使用したセキュリティ管理．
- RPC を同期，非同期で実行．
- デバッグ等による Server 側の出力指示を Client 側のコンソールに出力可能．

3. モンテカルロ法による円周率演算処理

本節では，Client-Server ネットワーク環境下で Ninf-G Version5.0 の API 関数を利用した分散計算システムを実際に構築して，PC の台数を増やすことによる処理能力の

向上を確認する. 実行させる計算は, モンテカルロ法による円周率演算処理とする.

3.1 Client 及び Server の構築

表 3.1.2 に示される性能を持つ 5 台の Server PC と 1 台の Client PC を, 図 3.1.1 のようにローカルネットワーク環境で接続することにより, Client-Server システムを構築した. Server 間の負荷分散の均等化を図るため, 5 台の Server PC の計算能力を均一にした. さらに, Ninf-G Version5.0 は, Client-Server 間の通信に SSH を使用するため, Client PC 側の SSH 暗号化処理は, 各 Server PC 側のそれと比較して5倍の負荷が発生する. そこで, Client の PC には性能の高い PC を割り当て, SSH 暗号化処理および Server 側からの受信データ復号化処理に対応出来ることとした.

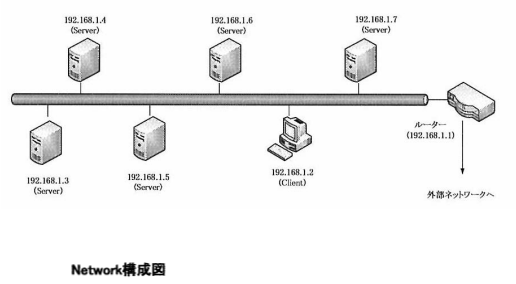

図 3.1.1

表 3.1.2

機種名	Dell Precision 390	Dell Optiplex GX520	Dell Optiplex GX620
CPU	Intel Core2Duo	Intel Pentium4	Intel Pentium4
クロック	1.86GHz	2.8GHz	3.0GHz
L2キャッシュ	2MByte	1MByte	2MByte
Memory	1.0GByte	1.0GByte	2GByte
HDD	Serial ATA 320GByte	Serial ATA 320GByte	Serial ATA 320GByte
IPアドレス	192.168.1.2	192.168.1.3	192.168.1.4

機種名	Dell Optiplex GX520	Dell Optiplex GX520	Dell Optiplex GX520
CPU	Intel Pentium4	Intel Pentium4	Intel Pentium4
クロック	2.8GHz	2.8GHz	2.8GHz
L2キャッシュ	1MByte	1MByte	1MByte
Memory	1.0GByte	1.0GByte	1.0GByte
HDD	Serial ATA 500GByte	Serial ATA 500GByte	Serial ATA 500GByte
IPアドレス	192.168.1.5	192.168.1.6	192.168.1.7

3.2 処理結果(表 3.1.1 および図 3.1.2)

モンテカルロ法による円周率の計算においては, 乱数を使用した試行回数が増えるにつれて, 円周率の精度が向上する. そこで, 試行回数を増やしながら, Server 2 台による処理時間と, Server 5 台による処理時間を計測して比較した. その結果, 処理回数が小さい内は, Server 2 台の処理の方が早く終了することが観測された. これは, 計算処理より通信処理の負荷の方が相対的に高いためと思われる. 一方, 処理回数が大きくなるにつれて, 計算処理の負荷が高まるため, Server 5 台の処理時間は Server 2 台の処理時間の半分以下で済むことが分かった.

表 3.3.1

PC名 / 処理回数	Server2台	Server 5台
1000	8553.6	17026.8
10000	8525.8	17672.6
100000	8111.4	17193
1000000	7652.4	16838.8
10000000	9495.8	17468.4
100000000	37679.4	27107.4
1000000000	395724.2	173541.8
2100000000	787373.4	341338.8

(単位:ms)

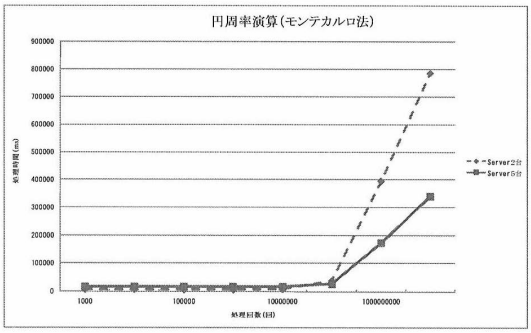

図 3.3.2

4. 量子コンピュータシミュレーション

本節では, 3節で提示した Client-Server 分散コンピューティング環境の上で, データベース探索に用いられる Grover の量子アルゴリズムをシミュレートするための分散計算ソフトウェアを開発する.

4.1 Grover のアルゴリズム

Grover のアルゴリズム[1]とはランダムに配置された N 個

のファイルの中から特定のファイルを探し出すための量子アルゴリズムである．この問題の古典的なアルゴリズムの平均ステップ数は N/2 である．一方，Grover のアルゴリズムは，$N^{1/2}$ ステップ以内で，高確率に検索に成功することが証明されている．

Grover アルゴリズムを3節の分散計算環境でシミュレートするための手続きを以下に述べる．説明の便宜上，量子ビットの桁数は 4 ビットとする．

ステップ1（初期状態の準備）． 全てのインデックス（量子ビット状態）に同じ振幅を与える．

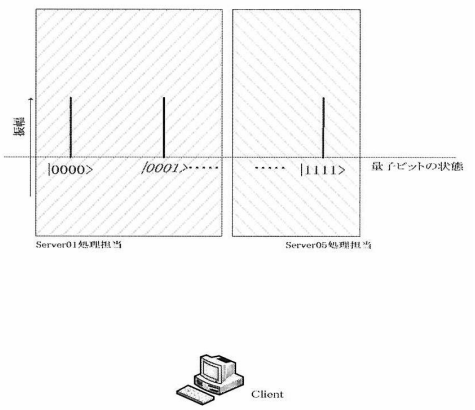

図 4.2.1

ステップ 2． 各 Server において，キーインデックスの振幅だけを反転（＋→－）する．

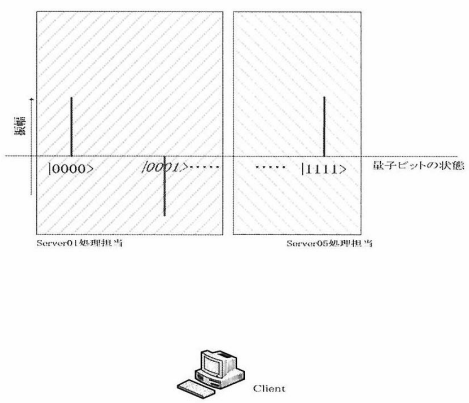

図 4.2.2

ステップ 3． 各 Server において担当するインデックス領域の振幅の小計を求めて，Client に送付する．

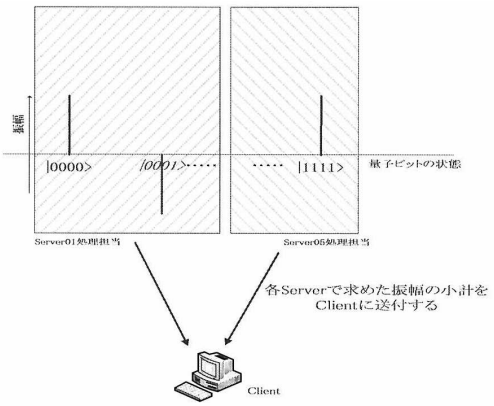

図 4.2.3

ステップ 4． 各 Server から送付された振幅の小計を全て受信した後に，全てのインデックスに渡る振幅の平均値を求めて，各 Server に送付する．

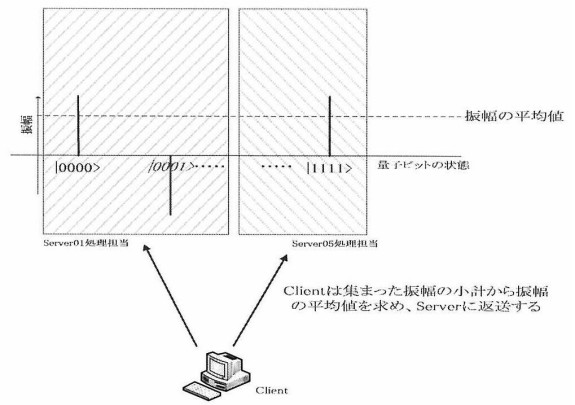

図 4.2.4

ステップ 5． 各 Server において，振幅の平均値を中心にして，インデックスの振幅値を折り返す．

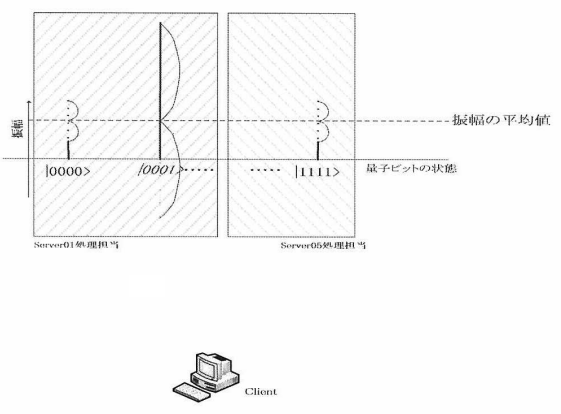

図 4.2.5

ステップ 6. (2)～(5)の処理を繰り返し，キーインデックスの振幅の平方和が 0.99 以上になったときに，終了する．

5.動的パラメータに対応した分散計算ソフトウェアの開発

4節で提示したシミュレーションの各ステップを3節の Client-Server 分散計算環境下で実現するに当たり，以下の問題点を克服する必要がある．

問題点1：シミュレーションのステップ 3 において，Server は Client から呼び出された関数の処理を継続したままで，Client に小計を送付する必要がある．しかしながら，Ninf-G においては，小計を Client に送付した瞬間に，その関数の処理が終了してしまう．

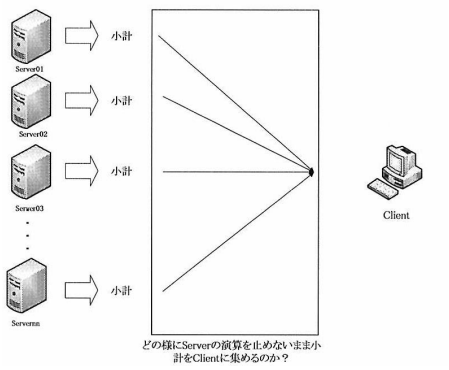

図 5.1.1

問題点2：シミュレーションのステップ4において，Client は，Server で現在稼働中の関数に対して平均値を通知する必要がある．しかしながら，Ninf-G においては，1 台の Server に対して呼び出せる関数は1種類であり，かつ既に稼働中の関数に対して新たなデータを渡すこともできない．

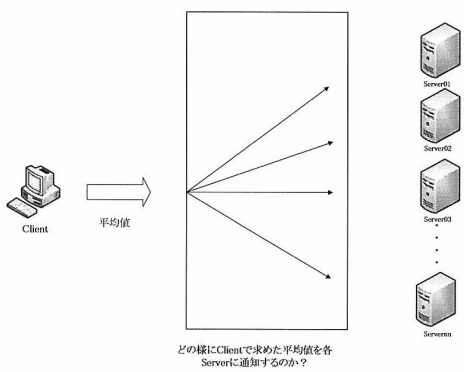

図 5.1.2

5.1 Client 経由の Server 間通信の実現

前小節の問題点1および2を克服するために，例えば，Linux API の中の socket 関数を使用すれば，Server 間通信は実現可能である．しかし，socket 関数を利用するとなると，socket 関数用のインターフェースなどを別途開発して組み込むとになり，開発工程やエラー処理が増大するばかりでなく，Ninf-G 単独のソフトウェアが持つ機動性の良さも失われてしまう．そこで，本稿では，Ninf-G 単独で，問題点1と2の解決をすることを試みる．

残念ながら，Ninf-G は，socket 関数のような，PC 間通信を行うための API 関数を持たない．そこで，各 Server に対して，計算処理用の関数と通信処理用の関数を Client が別途呼び出して起動することにより，動的に発生するパラメータを処理する仕組みが考えられる．実際は，問題点2で述べたように，Ninf-G 上では，異なる関数を同一 Server 上に同時に起動させることは出来ないことが判明した(図 5.1.1)．幸いなことに，同一の関数ならば，同一 Server 上に同時に起動させることが可能であった(図 5.1.2)．そこで，同一の関数に対して異なる入力パラメータ値を与えて 2 回呼び出すことにより，通信処理担当と演算処理担当の分岐呼び出しを実現することにした．最初の呼び出しで，Client-Server 間の通信処理を担当する関数を起動し，2 回目の呼び出しで，Server 上での演算処理を担当する関数を起動した (図 5.1.3)．

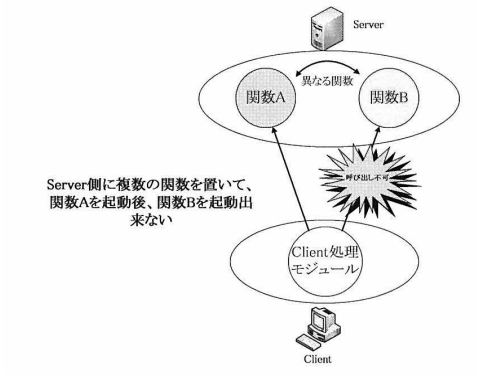

図 5.1.1

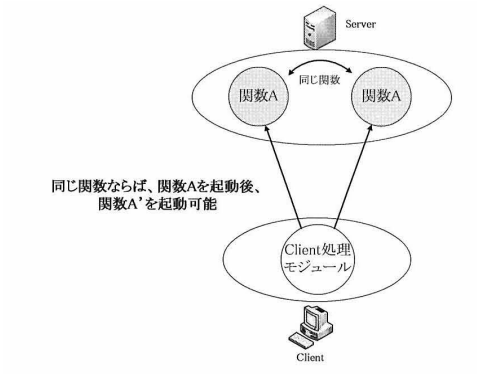

図 5.1.2

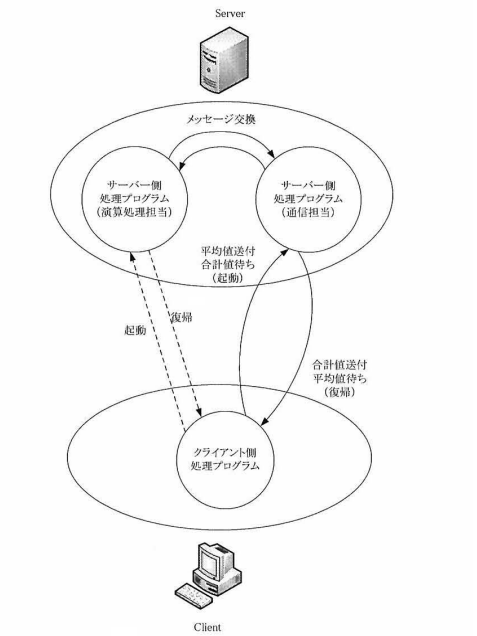

図 5.1.3

以下のステップに従って，図 5.1.3 のシステム上で 4 節の量子コンピュータシミュレーションを実行した．

ステップ1． Client PC から各 Server PC に対して通信処理関数を起動し，Message Queue の作成を指示する．

ステップ2． 各 Server は Message Queue を作成し，Client PC に復帰する．

ステップ3． Client PC は，各 Server が Message Queue の作成を完了した順に，演算処理関数を起動する．

ステップ4． 各 Server の演算処理関数は，Grover のアルゴリズムに従って，担当するインデックス領域部分の中にキーインデックスがあればその振幅の符号を反転させてから，振幅の小計を求める．小計が出たら，Linux の API である Message 関数を使用して，通信処理関数に小計を通知する．通信処理関数は，小計を戻り値として Client に復帰して終了する．

ステップ5． Client PC は，全ての Server の通信処理関数が復帰・終了したら，全ての小計を加算して平均値を求める．また，キーインデックスの振幅の平方和を計算して，それが 0.99 以上であれば，終了フラグを立てる．

ステップ6． Client PC は，各 Server に対して，新たに通信処理関数を起動し，その入力パラメータとして，求めた平均値を送付する．

ステップ7． 各 Server の通信処理関数は，Client PC から入力パラメータ値で受領した平均値を，Message 関数を通じて演算処理関数に送付する．

ステップ8． 各 Server は，Client から受領した平均値を元にして，担当するインデックス領域の各振幅を，平均値の周りで折り返す．

ステップ9． 終了フラグが OFF の間，ステップ4～8を繰り返す．

6. 処理時間測定

モンテカルロ法による円周率演算時と同様の計算機環境で図 5.1.3 の分散システムを構築して，前節のステップに従って，N = 25 桁で量子コンピュータシミュレーションを実行した．従って，インデックス数は 2^{25} 個である．このインデックス領域を各 Server に均等に割り振った．また，キーインデックスは 4 個として，ランダムに発生させた．

このような状態の下で，Server の台数が 1 台，3台，5台の各場合についての計測をおこなった(表 6.1.1)．Server の台数が 1 台の場合の処理時間を比較すると，3台の場合は約 0.43 倍，5台の場合は約 0.33 倍となっている．

表 6.1.1

量子コンピュータシミュレーション PC=5台

回数	1	2	3	4	5	平均
処理時間	746835	598803	810578	682429	825503	732829.6

(単位:ms)

量子コンピュータシミュレーション PC=3台

回数	1	2	3	4	5	平均
処理時間	891724	1225319	1284257	1141615	1065472	1121677.4

(単位:ms)

量子コンピュータシミュレーション PC=1台

回数	1	2	3	4	5	平均
処理時間	2534094	2631779	2628339	2658910	2640034	2618631.2

(単位:ms)

7. 各関数の処理時間の分析

表 6.1.2 は，ステップ 4 で Server が通信処理関数呼び出しを復帰・終了させた時刻から，ステップ 5 で Client がその復帰・終了を全ての Server について確認する迄の時刻の差分を，それぞれの Server 毎に計測した結果である．ただし，Server は，ステップ6で通信処理関数を呼び出す順番に表記してある．先に呼び出した通信処理関数程，復帰・終了する時刻が早いため，他の Server が復帰・終了する迄長く待たされていることが分かる．

そこで，ステップ6で，通信処理関数を非同期で呼び出した時刻から，その制御が戻ってくる迄の時刻の差分を，それぞれの Server 毎に測ったところ，80～90ms 近く掛っていた(表 7.1.2)．さらに，非同期で呼び出した時刻から，Server 側にその制御が渡る迄の時刻の差分を測ったところ，10ms～20ms 程度余計に時間が掛っていた(表 7.1.2)．

表 7.1.1

各Serverの復帰開始からすべてのServerの復帰が完了するまでの時間

回数 / Server IP	1	2	3	4	5	平均
192.168.1.3	502.00	425.00	408.00	410.00	741.00	497.20
192.168.1.4	314.00	341.00	322.00	326.00	323.00	325.20
192.168.1.5	230.00	249.00	237.00	239.00	235.00	238.00
192.168.1.6	160.00	171.00	164.00	166.00	160.00	164.20
192.168.1.7	78.00	79.00	80.00	83.00	76.00	79.20

(単位:ms)

表 7.1.2

Client側 非同期関数呼び出し処理時間

回数 / Server IF	1	2	3	4	5	平均
192.168.1.3	91	89	88	88	90	89.2
192.168.1.4	83	88	89	88	88	87.2
192.168.1.5	89	88	86	88	88	87.8
192.168.1.6	89	88	88	89	89	88.6
192.168.1.7	82	84	85	90	91	86.4

(単位:ms)

Server側関数実行開始迄の時間

回数 / Server IF	1	2	3	4	5	平均
192.168.1.3	101	99	98	98	100	99.2
192.168.1.4	114	119	120	119	119	118.2
192.168.1.5	93	92	90	92	92	91.8
192.168.1.6	93	94	93	94	94	93.6
192.168.1.7	106	108	109	114	115	110.4

(単位:ms)

この結果から，Client から Server への非同期呼び出し処理を完了するための時間は，Server1 台当たりについて 90ms～110ms 掛かることが分かった．そこで，改めて表 7.1.1 を眺めると，最初の Server の平均値と最後の Server の平均値の差分は 400ms 程度であり，非同期呼び出し処理を 4 回行うのに掛かる時間の分だけ待たされていることが分かる．

このことから，Ninf-G を利用した分散システムにおいては，各 Server PC の演算処理時間が 100ms 程度よりも短い場合は，Server PC の台数を増やすことによるスケールメリットは得られないことが判明した．

8. 負荷分散処理

Client は, Server への非同期呼び出しが全て完了した時点で, Server からの復帰受け付けを開始する. 前節の解析から, Server への非同期呼び出しを開始してから, Server からの復帰受付を開始する迄に, 500ms 程度の時間が掛る. そこで, 復帰受付開始時刻から, 各 Server が実際に復帰する迄時刻の差分である待ち時間を計測した表が, 表 8.1.1 である. 最初の方で呼び出された 3 台の Server は, 復帰受付開始時刻迄に演算処理を終了して復帰を完了しているため, 待ち時間はほぼ 0 である. 一方, 最後に呼び出した 2 台は, 復帰受付開始しばらくしてから, 復帰している.

そこで, 最後の2台の Server に掛る負荷を最初の3台の Server に分散させることにより, 全体の処理時間の短縮を図った. 具体的には, 2^{25} 個のインデックスの Server への均等分配を若干ゆがめて, Server01～03 に全体の数%分の負荷を余計に均等分配した. 表 8.1.2 から分かる通り, 全体の 1%程度の負荷を分配すると, 僅かではあるが, 処理時間の短縮が観測された.

表 8.1.1

Server通信処理→Client復帰待ち時間

回数 / Server IP	1	2	3	4	5	平均
192.168.1.3	0	0	0	0	0	0
192.168.1.4	0	0	1	0	0	0.2
192.168.1.5	1	0	12	0	0	2.6
192.168.1.6	105	91	44	6	83	65.8
192.168.1.7	255	117	81	392	266	222.2

（単位:ms）

表 8.1.2

◎全体の1%をServer01～03で負担
(Server01～03 = 20.33･･･%,Server04 = 20%, Server05 = 19%)

回数	1	2	3	4	5	平均
処理時間	710666	707591	754317	694454	772217	727849

（単位:ms）

◎全体の3%をServer01～03で負担
(Server01～03 = 21%,Server04 = 20%, Server05 = 17%)

回数	1	2	3	4	5	平均
処理時間	894652	912827	990148	840487	772817	882186.2

（単位:ms）

◎全体の5%をServer01～03で負担
(Server01～03 = 21.66･･･%,Server04 = 20%, Server05 = 15%)

回数	1	2	3	4	5	平均
処理時間	821905	801612	799224	826152	842537	818286

（単位:ms）

9. まとめ

今回は, 量子コンピュータの分散並列シミュレーションを行うために, 動的パラメータを処理する Client-Server 分散計算環境を Ninf-G 上で作成して, シミュレーション実験を行った. その結果, 各 Server 処理の負荷が高ければ, Server の台数に比例して, 処理時間の短縮が見込めることが分かった. ただし, 各 Server の処理時間が 100ms 程度以下の場合は, スケールメリットが得られないことも分かった. また, 各 Server の処理時間が 100ms ×Server の台数以下の場合は, Server の呼び出し順に応じた負荷分散が有効であることも分かった. 今回開発した, Client を経由した Server 間通信プログラムは, フィードバック制御のシミュレーション, リモート監視・操作可能な制御プログラムとして利用出来る. 今後の課題として, Ninf-G の非同期呼び出しを並列処理するような仕組みを考案して, 各 Server の処理が軽くてもスケールメリットが得られるようにしたい. また, 高いセキュリティが求められる現在のインターネット環境下においても利用出来るように, 今回の分散計算システムを拡張したい.

参考文献:

[1] L. Grover, A fast quantum mechanical algorithm for database search, STOC'96, 212–219, 1996