未知のノードを含むネットワーク分散協調演算処理システム

Network Cooperative Calculation System with Unknown Nodes

阿部　一博[1)]
Kazuhiro Abe

1)日本無線(株)　モバイル研究所（〒181-0013 東京都三鷹市下連雀五丁目１番１号
E-mail: abe@lab.jrc.co.jp）

ABSTRACT.　Recently, high speed processing computers are inevitably required for many kinds of scientific and engineering applications, for example, graphics and simulations. The solution ways for this demand are, for example, cooperative controlled multi-nodes system and centrally controlled multi-nodes system. In this paper, we propose an idea to diffuse these two systems into one system to make much more effective and speedy system. In this system, even if the communication speed between nodes each other is relatively slow, the total processing speed is rather higher compared with two systems. The dominance is certified using simulation results. And design based on Java2 concept will establish the independence system for machines or operating system.

１．背景

近年グラフィックスおよびシミュレーション等の分野において、計算機は必要不可欠なものとなってきている。これに伴い計算機の処理速度の向上が求められるようになってきた。具体的な手法としては、プロセッサの並列化およびプロセッサの動作周波数の上昇化とさまざまあるが、今プロセッサの並列化に着目する。プロセッサの並列化は大型の計算機に従来より多く用いられてきた手法であり、中心的な位置にある。計算機内部のプロセッサが行なう処理として、タスクの処理およびプロセッサ間通信の処理が挙げられ、これら両者間の関係が、しばしば大型の計算機を大別する為の指数となっている。[1][2]

一般に各プロセッサにおいて、タスクの処理速度が通信の処理速度を上回る場合、並列化システム全体の処理速度の向上を期待することができる。従ってプロセッサ間の通信速度が低速であったとしても、単位時間に受信可能なタスクの処理量が大きければ問題ないことが理解できる。

一方プロセッサの並列化を考えた場合、ネットワークに内包される計算機を並列化することもほぼ同様であると考えることができる。ネットワークは近年普及し現在一般家庭まで深く浸透している。従ってこれに内包される計算機の数を考えたとき、この数は膨大であり、この資源を有効に活用することを考えることは重要であると思われる。これに関しては従来より多くの研究[3]～[5]が試みられているが、大規模なネットワークでの試みはあまりなされていない。

今回示すシステムは、ネットワーク内の各計算機に処理を分散させ効率よく命令を実行するものである。ネットワーク内の通信速度は確かに低速ではあるが、先までの考察により実現可能であることに期待がもてる。しかし本システム内には、未知の計算機を考慮する為に、従来考案されたシステムではほとんど重要視されない問題が発生する。

ここではシステムを実現する上での基盤となるアルゴリズムの考察および評価、すなわち分割し易い構造を有する処理を各計算機に分散し実行させることおよび問題に対する対処を考慮することを目的としている。また実際に本システムを具現化する際、"Java2"を用いる予定である。これにより、機種依存およびオペレーティング・システム依存がなく、またネットワークのソフトウェアに関して容易に完成度の高いものを構築することが可能であることが期待される。[6]～[11]

本論文では第 2 章にて本システムの全体構想を示す。次に第 3 章にて全体構想の内部詳細を示す。そこでは主として処理の分散方式および本システム構成の際の各問題点に対する対処方法についてのアルゴリズムについて示す。またこの際に行った確認の為のシミュレーション結果および検討について第 4 章にて示す。最後に第 5 章にて結論および今後の方針について述べる。

２．目的

従来よりタスクを分割し、ネットワーク上の各ノードに分散させ処理速度を向上させることは行われてきた。その代表的手法とし集中分散系および自律分散系がある。集中分散系は、ある端末がホスト(ノードを集中的に管理するサーバを意味する)となり、ホストに接続されるノードを集中的に管理および制御し分散処理を行う構造を有するものである。しかしこの場合一括したノード管理に伴うホストの管理能力およびネットワークの稼働率がノードの数の上昇に従い 比例的に悪化し、管理可能な限界が早々に発生する。さらにホストが何らかの要因で停止した場合、これを補うシステムの必要性および復旧にかかる問題が大きい。

他方自律分散系は集中分散系に挙げた集中的な管理を行わない為、ホストの処理限界および停止に伴う問題は大き

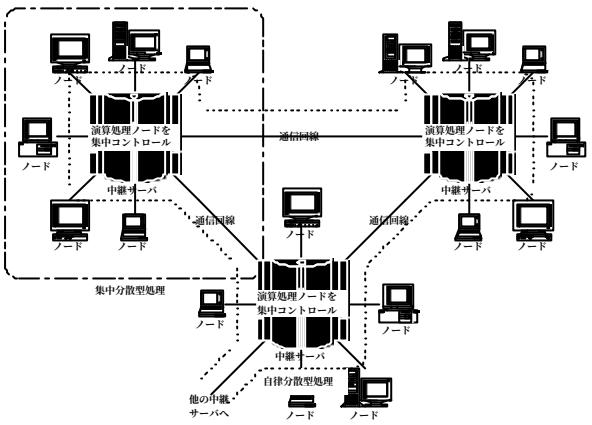


図1 システム構成

Figure1 System Construction

くはない。しかし一方各端末にて同様の処理が繰り返し行われ、タスクの分散が行われる為、低い処理能力のノードは通信処理および通信自体に大きな負荷がかかる為、処理速度はあまり上昇せず分散速度も低下する可能性がある。

図1に示す通り本システムではこれら2つの系の利点を生かし、中継サーバとこれに接続されるノードは集中分散系とし、中継サーバ同士では自律分散系としている。本方式を用いることにより集中分散系および自律分散系に見られる各問題点自体も分散される為、ノード数の増加に対し比例的な速度の向上が期待できる。

これらをまとめて表1に示した。

また本システムではユーザにとって未知のノードにおいてユーザ送信の処理が実行される。この為システムの頑健性が重要となる。次にその一部を列挙し検討をおこなう。

・未知のノードの実行結果が適切なものか否かを判断するための処理(検算)
送信した処理が、未知のノードにて正しく実行されたか否かの確認。

・ノード停止したときの対処(ノード停止)
ノードが停止し、処理が中断された為結果を中継サーバに送信できない場合の対処。

・処理の強制終了命令の実行(緊急停止)
送信した処理が、ユーザにより誤りと判断され、その処理を停止することが必要な場合の対処。

これら各検討事項の確認に関し、検算およびノード停止については各中継サーバが保有する処理負荷の斑として扱い、その斑が高速に吸収されるか否かについてのシミュレーションをおこなう。緊急停止に関しては。別途シミュレーションをおこない、緊急停止命令が各中継サーバに対し高速に伝播される否かを検証する。

表1 システム概要

Table1 System Concept

	ネットワーク稼働率	サーバの負荷	ントヘデータを送信可能な量
集中分散系	小	大	大
自律分散系	大	小	小
合成系	大	小	中

3. システム詳細

次にシステム概要で挙げた各機能の詳細について述べる。はじめに(1)にて、集中分散系の解析モデルについて述べる。つづいて(2)にて、自律分散系の解析モデルおよびシミュレーションをおこなう際の系について示す。次に(3)、(4)および(5)にてシステムの頑健性を補償するための一部機能、検算、緊急停止およびノード停止についての解析モデルおよびシミュレーションの系について示す。

ただし、検算およびノード停止については解析のみをおこないシミュレーションに関しては集中分散系を含め、各中継サーバが保有する処理負荷の斑とし現象を扱うこととしている。

また、頑健性の検討項目については、実機を用いての確認を行う。

(1) 集中分散系

はじめに集中分散系について示す。図2に示すとおり集中分散系は中継サーバおよびノード間にて用いられている分散方式である。

中継サーバは、自身に接続される各ノードの単位時間あたりに処理できる量、すなわち処理能力の値に比例した量に処理を分割し、送信の処理をおこなう。処理能力はベンチマーク値とし、これは予め算出され中継サーバに送信されている。次に上記分割に関する理論モデルを示す。

今N_iをノードiへ一度に送信される処理量とし、

$$N_i = \frac{W}{D\sum_{i=1}^{N} J_i} J_i \qquad (1)$$

ただし、Wを中継サーバが保有している総処理負荷、Nをノード総数、J_iをあらかじめベンチマーク等で調査したノードiの単位時間におこなうことのできる処理負荷の値およびDを各ノードに対し処理負荷を分散させる際の回数としている。

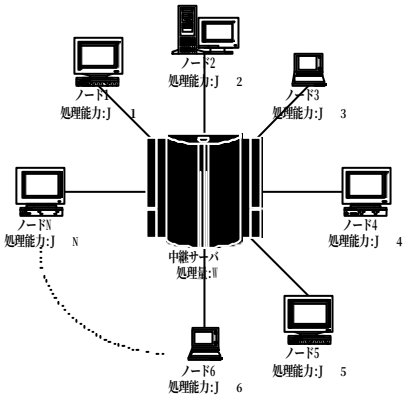


図2 集中分散系

Figure2 Centrally Controlled System

(2) 自律分散系[12]～[14]

次に自律分散系について示す。図 1 に示すとおり自律分散系は中継サーバ間で用いられている分散方式となっている。

今中継サーバが、図2 に示すとおりに接続されているものとする。また S(x,y) は S(x+1,y)、S(x,y+1)、S(x-1,y) および S(x,y-1)に対し 0.25 の確率でアクセスすることが可能であるとする。この系における単位時間あたりの中継サーバ間の処理の移動量を解析する。

仮に S(x,y)が S(x+1,y)とお互いアクセスしたものとすると、S(x,y)と等しい量の処理負荷を S(x,y+1)、S(x-1,y) および S(x,y-1)が有していると考えることが可能である。この時 S(x,y)と S(x-1,y)間での処理の移動量 J(x,y)は処理数を P(x,y)、拡散速度定数を D として、

$$J(x,y) = D \cdot \nabla_{x,y} P \tag{2}$$

と表現できる。一方単位時間当りに中継サーバ S(x,y)より S(x+1,y)に送信される処理負荷を、

$$P(x,y) = D \cdot \nabla_{x,y} W \tag{3}$$

とすれば、

$$J(x,y) = D \cdot \nabla_t^2 W \tag{4}$$

となる。(4)式は一般に拡散方程式として知られている。したがってある中継サーバに対して大きな処理負荷がかけられた時、図 4 に示すとおり、その近傍の中継サーバとの平均を常時とることによって、その処理負荷はすべての中継サーバに均等に分散される。

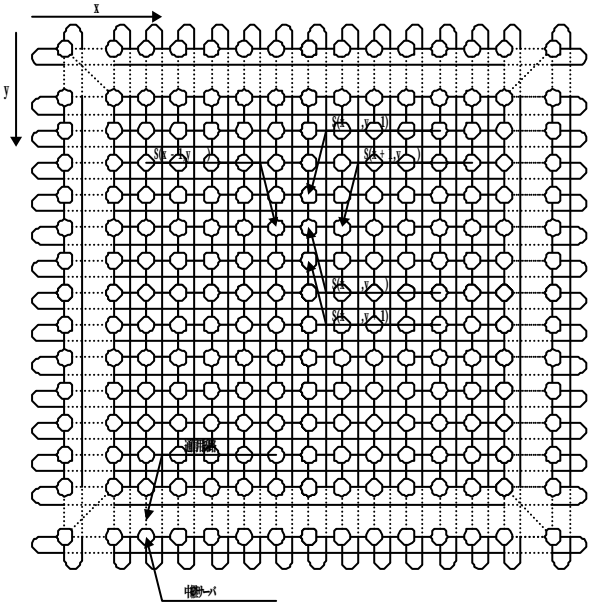


図 3 自律分散系
Figure3 Cooperative Controlled System

図 4 自律分散シミュレーション
Figure4 Cooperative Simulation

本方式は文献[15][16]において、これら詳細および応用例についてよく論じられているが、ここでは特に拡散速度定数の変化および各々の中継サーバに存在する動的な処理分布に伴う分散速度に着目したい。

(3) 検算

次に検算の手法について述べる。本論文に示すようなシステム管理者にとって未知のノードが含まれる場合、そのノードより送信されるデータが妥当なものか否かを判断する機能を有することは重要である。ここでは図 5 に示すとおり、数台に同一の処理を送信し、そのデータの一部をランダム抽出し比較をおこなう。

この場合のデータの妥当性を示す指数 P_s は、

$$P_s = \left\{ 1 - \left(\frac{1}{2} \right)^{mk} \middle| n \geq k \right\} \tag{5}$$
$$othewise \cdots 1$$

となる。ここでの誤算確率は 0.5 としている。

ただし、m を同一の処理をおこなうノード総数、n をデータの総数、k を検査用のサンプル数としている。

よって、検査用のサンプル数はさほど多く必要とされない為、解析的には、同一の処理をおこなうノード数に比例した速度の低下が起こると考えることができる。

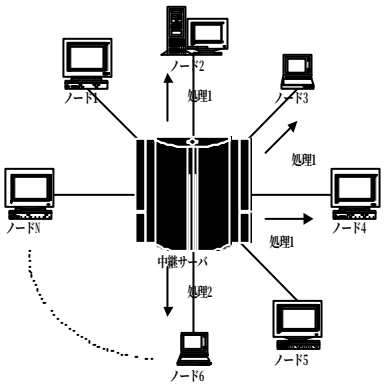


図 5 検算
Figure5 Check Sysytem

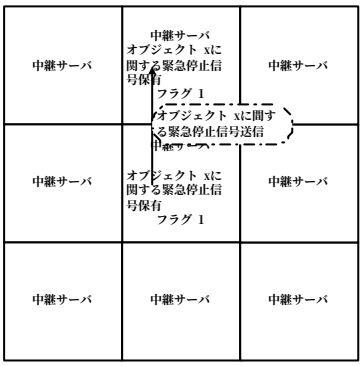


図 6 緊急停止 0

Figure6 Emergency Halt 0

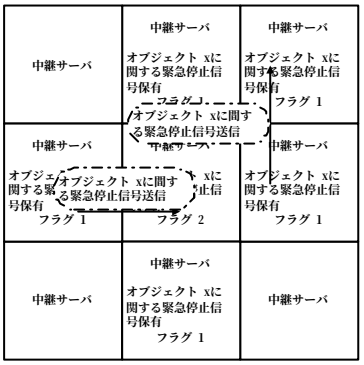


図 7 緊急停止 1

Figure7 Emergency Halt 1

図 8 緊急停止 2

Figure8 Emergency Halt 2

(4) 緊急停止

はじめに図 6 の状態にあるものとする。中心の中継サーバに対し緊急停止命令を送信した場合、この信号を受信した中継サーバはフラグ 1 をたてた後、接続されているすべての中継サーバに対し緊急停止命令を送信する。これを受信した中継サーバは同様にフラグ 1 をたてる。

次に図 7 に示すとおり、送信後はフラグ 2 をたて、以後接続されている中継サーバより送信される緊急停止命令の受信のみをおこなう。

次に図 8 に示すとおり送信したすべての中継サーバより同一の緊急停止命令を受信した場合、フラグおよび対象オブジェクト消去する。

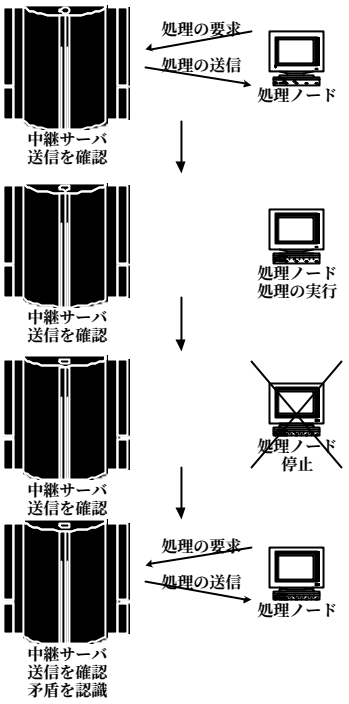


図 9 ノード停止

Figure9 Node Down

(5) ノード停止

次にノードが停止したときの対処について示す。

図 9 に示す通りノードが通常動作をしている間、ノードは自身に実行すべき処理がなき場合、中継サーバに対して処理を要求する。処理要求を受信した中継サーバは自らが管理している処理の一部をノードに対し送信する。ノードは処理を受信すると直ちにその処理を実行、中継サーバに結果を送信する。中継サーバは結果を受信、ノードが正常に動作していることを確認する。

他方ノードが処理途中で停止した場合結果を中継サーバに送信できない。したがって再起動後ノードは処理を保有していないため、中継サーバに対し処理を要求する。中継サーバはノードより結果を受信してないことを確認、結果の受信なしに処理の再送を要求された矛盾を認識し、ノードが停止前におこなっていた処理を再送する。

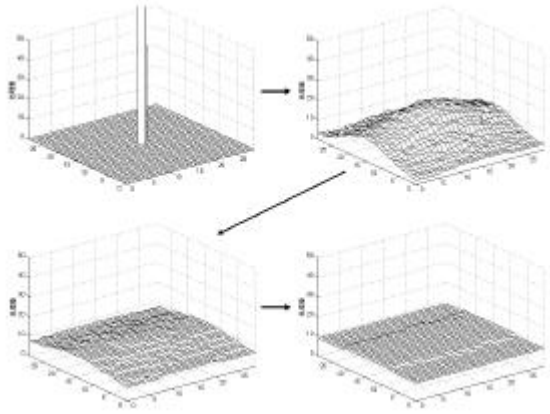
本動作は解析的に、処理が均等配分された後の処理負荷の斑が発生したことを意味する。この斑は先の自律分散系において高速に吸収されることが期待される。次にシミュレーション結果について述べ上記の現象について確認する。

4. シミュレーション[17]

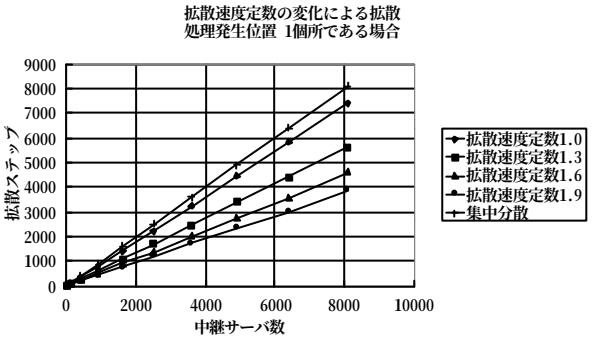
次に自律分散系に関する検証、ノード停止等の各中継サーバの動的な処理負荷変動への対応に関する検証および緊急停止命令の伝播に関する検証にかかるシミュレーション結果について示す。

(1) 自律分散系

図 10 に自律分散系のシミュレーション結果を示した。はじめに単一の中継サーバの処理負荷が分散される様子、お



900 台の中継サーバが図 3 に示すとおり接続されている場合において、中央の 1 台の中継サーバに処理負荷 9000 が与えられた場合の分散の様子を示す。



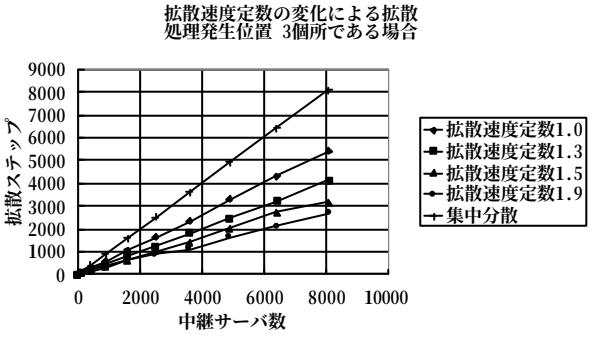


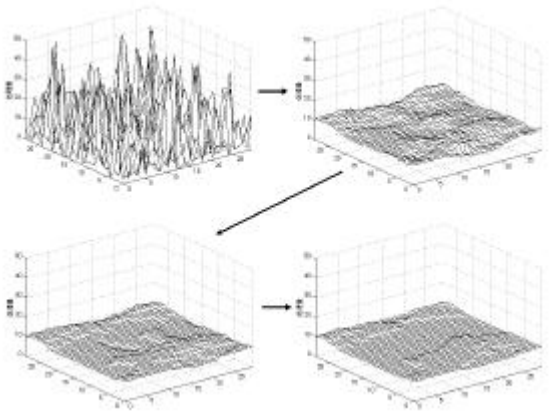
図 10 自律分散経過

Figure10 Cooperative Flow

よび拡散速度定数の上昇にしたがい処理の分散速度が向上している様子がうかがえる。しかし一方集中的に処理を分散させた場合に比較し分散の速度はさほど効果がない。だが他方 3 台の中継サーバの処理負荷が分散される場合、集中的に処理を分散させる手法に比較し、分散の速度が上昇している様子がうかがえる。

また図 3 に示した中継サーバ同士の接続形式外の接続方法をとった場合についても同様にシミュレーションをおこない、前者と同様に処理負荷が各中継サーバに対し均等に分散されていく様子が確認されている。

これより、中継サーバの不慮の停止に対してもシステム全体の停止は免れることがうかがえる。



900 台の中継サーバが図 3 に示すとおり接続されている場合において、すべての中継サーバにランダムな処理負荷が与えられた場合の分散の様子を示す。

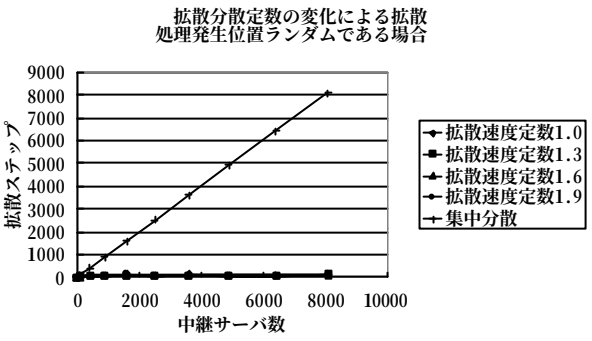


図 11 動的な処理負荷の変化

Figure11 Dynamical Weight

(2) 動的な処理負荷の変化

図 11 にノード停止、各中継サーバにおける単位時間あたりの処理量および検算の際の安全指数斑に伴う処理負荷の動的な変化が高速に吸収されるか否かについてのシミュレーション結果を示す。図 11 に示すとおり、処理負荷の斑がすべての中継サーバに分散され高速に吸収されていく様子がうかがえる。

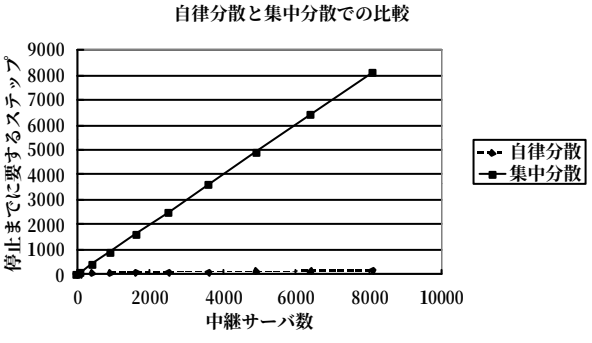


図 12 緊急停止

Figure12 Emergency Halt

(3) 緊急停止

図 12 に緊急停止に関するシミュレーション結果を示した。本シミュレーションは単一の中継サーバより緊急停止信号を送信し、システム全体に伝播するまでの速度を計測した物である。図 12 に示すとおり集中分散的に伝播させたものに対して、より高速に伝播されている様子がうかがわれる。

また、(1)自律分散系、(2)動的な処理負荷の変化および(3)緊急停止のいずれのシミュレーションにおいても、デットロック等の異常事態が発生していないため、これらの点より処理の分散および実行の速度のみならず、安全性の見地からも本方式を適用したシステムの有用性に期待がもたれる。

5．結論および今後の方針

今回示すシステムはネットワーク内の各計算機に処理を分散させ、効率よく処理を実行させるものである。ネットワーク内の通信速度は確かに低速であるが実現可能であることに期待がもてる。しかし本システム内部には、未知の計算機が存在する為、従来考案されたシステムでは重要視されない次の問題が発生する。

・未知のノードの実行結果が適切なものか否かを判断するための処理(検算)
　送信した処理が、未知のノードにて正しく実行されたか否かの確認。
・ノード停止したときの対処(ノード停止)
　ノードが停止し、処理が中断された為結果を中継サーバに送信できない場合の対処。
・処理の強制終了命令の実行(強制終了)
　送信した処理が、ユーザにより誤りと判断され、その処理を停止することが必要な場合の対処。

ここでは、本システムを実現するための基盤とされるソフトウェアの構想、理論の検討およびそれら各シミュレーションの結果について示すと共にその妥当性について検討をおこなった。

結果より今回検討した範囲についてはシステム的に問題なく動作することがうかがわれる。現在、本論文に挙げた解析結果を元に、実際にシステムの具現化を試みる予定であり、シミュレーションでは結果を判断しがたい頑健性の補償については実機にて評価されている。

また今回のシステム作成検討にあたり、問題を多く発見した。一つはセキュリティー問題である。本システムはユーザもしくは中継サーバにとって未知の部分にて動作する部分が多い。すなわち未知のユーザの作成した処理が未知のノードにて実行されるため、処理の結果の補償およびノードの安全性の補償に関する検討は非常に重要であると思われる。

他方で、システムを開発するにあたり、そのシステムを制御できるある種の柔軟性をもたせることも必要であると思われる。この柔軟性についても規則を必要とし例えばそれを言語として体系化させることがその一つとして考えられる。

以上、考慮すべき点はまだ多くあり実際の起動にいたるには多くの時間を必要としている。

6. 参加企業及び機関

1. ネイチャーランドジャパン株式会社
　斎藤　昌義代表・難波　　薫様
2. 日本無線(株)
　馬場　満徳様
3. アイアンドエル・ソフトウェア株式会社
　廣澤　　稔様・近藤　充弘様

また、本案件をご採択いただいた東京大学　松島　克守教授および本論文提出にあたり多くのご助言いただいた東京電機大学　柿倉　正義教授に対しこの場を借りて謝意を表す。

7．参考文献

[1] 飯塚　　肇・緑川　博子:　並列プログラミング入門, 丸善株式会社, (2000)
[2] 大森　健児 訳:　並列プログラミングの基礎, 丸善, (1990)
[3] 山添　博史・田中　慎司・伊達　新哉・五島　正裕・森　眞一郎・富田　眞治: 並列アプリケーションを指向した分散システムコンピュータ・コロニーの構想, 情処研報 97-OS-76(SWoPP 阿蘇'97), pp.55-60
[4] 山名　早人・佐藤　三久・児玉　祐悦・坂根　広吏・坂井　修一・山口　喜教:　並列列計算機 EM-4 におけるマクロタスク間投機的実行の分散制御方式, 情報処理学会誌, Vol36, No.07-010
[5] B.ウィルキンソン・高橋　義造・渡辺　　尚・小林　真也・長谷川　誠: 計算機設計技法, 凸版, (1998)
[6] 日本サン・マイクロシステムズ株式会社: JAVA RMI,サイエンス社, (1998)
[7] Troy Bryan Downing著・夏目　　大 訳: JAVA RMI IDG Books JAVA RMI(JAVA Remote Method Invocation)ハンドブック, (1997)
[8] Jon Meyer and Troy Downing 著・鷲見　　豊 訳, JAVA バーチャルマシン,オライリージャパン, (2000)
[9] ダグリー・松野　良蔵: JAVA スレッドプログラミング, 翔泳社, (2000)
[10] 池田　　誠: Handy Reference JAVA 言語ハンドブック, ナツメ, (1997)
[11] 小高　和宏: TCP/IP JAVA ネットワークプログラミング, オーム, (1999)
[12] 登坂　宣好・大西　和榮: 微分方程式のシミュレーション, 東大, (1998)
[14] ブロム・ホスト・サンデル・森　　誠: 確率問題ゼミ, シュプリンガー・フェアラーク社, (1997)
[15] 寺田　文行・樋口　裸一: 高校数学解法事典, 旺文社, (2000)
[16] 新　　誠一・池田　建司・湯浅　秀男・藤田　博之: 自律分散システム, 朝倉書店, (1995)
[17] 石田　哲郎・清水　　東: 改版　半導体素子, コロナ, (1994)
[18] 上坂　吉則: MATLAB プログラミング入門, 牧野書店, (2000)