PG Lab ディスカッションペーパーシリーズ
第16号　2007年6月

ISSN 1880-9146

Discussion Paper Series
No.16 June, 2007

エージェント間結合を考慮したMAS分散処理の実装

田中 義久，蟻川　浩，前田 太陽，村田 忠彦

文部科学省私立大学社会連携研究推進拠点
関西大学政策グリッドコンピューティング実験センター

Policy Grid Computing Laboratory,
Kansai University
Suita, Osaka 564-8680 Japan
URL : https://www.pglab.kansai-u.ac.jp/
e-mail : pglab@jm.kansai-u.ac.jp
tel. 06-6368-1228
fax. 06-6330-3304

関西大学政策グリッドコンピューティング実験センターからのお願い

本ディスカッションペーパーシリーズを転載，引用，参照されたい場合には、
ご面倒ですが、弊センター（pglab@jm.kansai-u.ac.jp）宛に
ご連絡いただきますようお願い申しあげます。

Attention from Policy Grid Computing Laboratory, Kansai University

Please reprint, cite or quote WITH consulting Kansai University Policy Grid Computing Laboratory (pglab@jm.kansai-u.ac.jp).

エージェント間結合を考慮した MAS 分散処理の実装

田中義久 [1]，蟻川浩 [2]，前田太陽 [2]，村田忠彦 [1, 2]

Implement of Distributed MAS Considering Relations among Agents

Yoshihisa Tanaka[1], Hiroshi Arikawa[2], Taiyo Maeda[2], Tadahiko Murata[1, 2]

概要

本稿では，MAS のグリッドへの展開を実現するために，MAS のエージェント相互間の結合度に応じた分散化について議論し，計算機の信頼性を考慮した分散システムの構築を行う．適用事例として，医療機関選択に関わる社会シミュレーションを提案システム上で実行し，その有効性を示す.

Abstract

To realize computational grid for multi-agent simulation, distributed computing system considers reliability on each nodes, was developed. It had method we called clonning relay task method, and it’s applied selection of medical services. Using the system, it obtained high reliability and identical results.

キーワード：分散処理，耐故障性，MAS

Keyword: Distributed Computing, Fault-Tolerance, MAS

1 関西大学総合情報学部

2 関西大学政策グリッドコンピューティング実験センター

1. はじめに

グリッドの登場以来，大学や企業では，高いコストパフォーマンスを持ちコモディティ化が進む計算機を複数用いて，大規模な問題解決環境を構築したいというニーズが高まっている．一方で，マルチエージェントシミュレーション（Multi-Agent Simulation，以下 MAS）を用いることで社会科学の解明を目指した研究においても，計算グリッドへの期待が高まっている．中でも，企業や大学，自宅にある計算機群を遊休資源として利用するニーズが高い（本稿ではデスクトップグリッドと呼ぶ）．この先行研究として Condor[1], Globus[2], BOINC[3], OmniRPC[4] など多くのプロジェクトが存在する．

大規模計算や高速化，高スループットを目指し MAS をデスクトップグリッドに展開するためには，適切な方法で MAS を実装する必要があるが，この実現には以下の 2 つの課題が存在する．1 つは，MAS の分散計算や計算機の相互通信が必要な並列計算には，どちらの場合も計算機の信頼性を考慮することが必要不可欠であること，もう 1 つは，信頼性を考慮したシステム上で乱数や通信などを考慮した MAS のための計算手法を確立することである．

本稿では，社会シミュレーションに代表される MAS の計算グリッド化を実現することを目指し，エージェント相互が密に結合した MAS を並列化する際に，実際にタスクを処理する各ノードに注目することで，計算の信頼性を確保する分散処理システムを構築するとともに，MAS の分散処理の実現方法とその実装について報告する．提案システム上で簡略化した医療機関選択シミュレーションを実行し，提案システムの有効性と，計算結果の再現性・同一性について議論する．

2. グリッドと MAS

デスクトップグリッドを利用して効果的に MAS の並列化・分散化を行うには，MAS の特性を十分に考慮し，適切な方法で計算を分割し実装する必要がある．本節では，エージェント間結合に応じた MAS の分割方法と，計算機の信頼性について述べる．

2.1 MAS の計算分割

MAS では，多数のエージェントが相互に影響を及ぼしながらシミュレーション全体が遷移する．一般に，MAS にはエージェントの相互関係があり，デスクトップグリッド上で並列化・分散化を実現するために個々の計算機間で通信処理を行う必要がある．ここでエージェント間の結合度に着目すると，対象とするモデルにより，1）全体が密な結合，2）全体が疎な結合，3）部分的に密な結合，に分類することができる．しかし，これらの実装には通信量と計算負荷について考慮する必要がある．

通信量の観点から見た場合，結合度が密な程，通信の頻度は高くなり，ネットワーク性能が異なるグリッド環境ではボトルネックとなりやすい．また，個々のエージェントの計

算負荷が極めて小さい場合，特に注意が必要となる．エージェントが環境全体を走査することで意思決定を行うモデルでは，環境を管理するノードを個別に用意することで計算を分割することが可能である．しかし，環境情報を常に最新に保つためには多くの通信が発生し，結果として環境管理ノードに負荷が集中する．そのため，効率的な環境情報更新アルゴリズムが必要である．

次に計算負荷の観点から MAS を見た場合，計算の負荷が不均一であることが多い．エージェントが移動するモデルや生存競争を行うモデルは，分割したタスクの計算負荷がシミュレーション実行中に動的に変動するという特性を持っている．このため，ある情報を通信するために同期を取ると計算負荷の小さいタスクは待ち状態に入る．この特性については，必要に応じてロールバックし非同期で計算する手法[5] が有効であった．

2.2 計算機の信頼性

一般に，アプリケーションの実行が長時間に及び，参加する計算機の台数が増えれば増えるほど，計算機が故障する可能性が高くなる．また，デスクトップグリッドでは所有者の利用を妨げてはならないため，計算機はいつ利用できなくなるか不明である．このためユーザから見れば計算機の信頼性は高くないといえる．

デスクトップグリッドで MAS を実行した際，いずれかのタスクの応答が無くなればアプリケーション全体が停止してしまうため，個々のタスクを確実に処理することは極めて重要である．計算グリッドに関する耐故障性については，故障して中断したタスクをいかにして復旧させるかという観点で，リスタートやチェックポイントなど様々な議論[6] [7] が行われてきた. MAS ではこれらの議論の他に, タスクが中断されないことが要求される. これは結合度に関係なく必要であり，特に並列計算を構成するノードの信頼性を動的な分散処理により確保しなければ，実際のデスクトップグリッドへの MAS の展開は極めて困難である．

3. MAS の分散手法

本節ではタスクを計算機の障害から保護するシステムを提案する．動的にタスクを複製するアプローチは，チェックポイントリスタートやノードの交換等の複雑な構造を不要にし，システム全体をシンプルに保つことができる．また，上位レイヤーで並列化を行う場合は耐故障性について特に意識する必要は無い．提案システムの概要を図 1 に示す．提案システムは，並列化された各タスクの信頼性を確実なものにするために，タスク処理のための計算グループを複数のノードで構成する．タスクは計算グループ内で複製され，複製間の同期はリレー方式により実現する．なお，計算グループ間の並列化手法として文献[5] が利用できる．

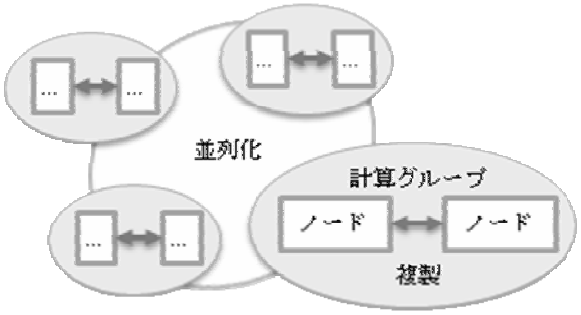

図 1 提案システムの概要

3.1 リレー方式による信頼性と性能の確保

長時間に渡る MAS を単純に並列処理した場合，スループットが変動するヘテロジニアスな環境で性能を得ることは難しい．また，ノードの参加・離脱を考慮すると，新規参入したノードに同じ計算を初めから行わせるのは非効率的であり，離脱するノードの計算結果が取得不可能な場合シミュレーションは永久に終了しない可能性がある．

我々は，1 つのシミュレーションを複数台で同時に実行しながら，MAS を一定の間隔で区切り，最も早くそのポイントのたどり着いた計算結果を採用し，同一のシミュレーションに参加する全ノード（同一計算グループに属するノード）を更新する手法を実装することで，信頼性と性能の確保を実現する．この間隔は時間によるものではなく，処理量によるものでなければならない．具体的には，MAS の全エージェントが意思決定を行う単位をステップとした時, 数 100～数 1000 ステップを同期する間隔とする. この手法はまた，ノードの動的な参加・離脱に対して頑健である．新規に計算に参加したノードは計算グループ内で共有されている最新の経過情報を取得し，その時点から計算を開始することができる．これにより，計算グループの全ノードが利用不可能にならない限り計算は継続されるため，ノードの離脱に対しては，特に意識する必要はない．また，経過情報を永続化すれば全ノードが利用不可能になることにも対処でき，提案システムは高い信頼性を MAS アプリケーションに与えることが可能である．

3.2 分散クローンの実現

計算グループの構築手法として，P2P の概念を採用する．サーバ・クライアント方式は管理や構築が容易であるが，サーバが利用不可能になったとき，計算グループが機能しなくなる．サーバの多重化を行ったとしても，サーバを切り替える際には全クライアントにその旨を通知し，新しいサーバの情報を提供する必要があるため，サーバ終了までに猶予が必要である．一方，P2P の概念を用いれば，全てのノードが対等なため，どのノードが利用不可能になったとしても，全体が停止することはない．また，サーバのアドレスを知る必要もなく，計算グループ内のいずれかのノードに接続することで参加が可能（図 2）

である．

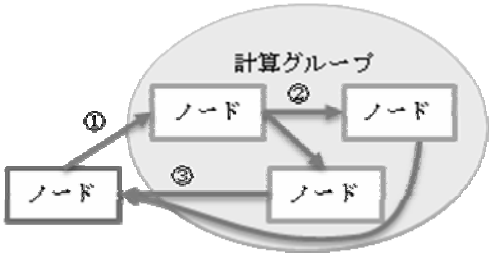

図 2 コネクションフロー

以下にその手順を示す．各項目は，図 2 中の番号と対応している．

(1) 計算グループに参加するノードは，計算グループ内のいずれかのノードを選択し，接続する．

(2) 接続を受けたノードは，計算グループ内に新しいノードの IP アドレスと待ち受けポートをブロードキャストする．

(3) 情報を受け取ったノードは新しいノードに接続を行う．

3.3 シミュレータの更新

計算の途中経過の配信はシミュレーションに必要な全データをシリアライズすることで行う．提案システムは P2P の概念を用いて構築されているため，どの途中経過を採用し，どのノードからデータを受信するかは各ノードが判断しなければならない．また，複数のノードが同時に同期ポイントに到達することで複数の経過情報を受信する可能性や，ネットワークの遅延によりデータを要求してから受信するまでの間により進んだ経過情報を受信する可能性も考慮しなければならない．そのため，経過情報の送受信は以下の手順で行われる．この中の受信側は経過情報を受信する度に行われ，複数の経過情報を受信した場合，並列に行われることに注意されたい．

(1) 同期ポイントに到達したノードは進捗状況の取得を行い, ブロードキャストする.

(2) 情報を受け取ったノードは，受信した経過情報の進捗と自ノードの進捗，受信した経過情報の進捗と記録済みの進捗をそれぞれ比較し，どちらも受信した情報が新しければ採用し，データ送信を要求する．この時，採用した進捗を記録しておく．不採用となった場合はここで終了する.

(3) 要求を受けたノードは進捗状況のデータを送信する.

(4) データを受信する. 受信が完了すると, 受信したデータの進捗と自ノードの進捗, 受信したデータの進捗と記録済みの進捗を再び比較する．これにより，ネットワークの遅延が原因となりシミュレータ更新の一連の手順が完了するまでに他のノ

ードからより新しい経過情報を受信した場合に，古い情報を採用する可能性を排除する．比較の結果，双方ともに受信したデータが新しければ採用し，シミュレータを更新，計算をリスタートする．

4. 実装

実装は Java で行った．Java は言語仕様としてシリアライズをサポートしており，クラスの継承やインターフェースの実装を用いることでアプリケーションを提案システムに容易に適合させることが可能である．また，ヘテロジニアスなグリッドを想定した場合，Java のポータビリティは不可欠である．

4.1 シリアライズ

提案システムのコアは，シリアライズによるシミュレーションの途中経過の配信である．MAS を提案システム上で実行するオーバーヘッドを最小化するためには，シリアライズのコストを抑えることが極めて重要である．Java では，シリアライザブルなオブジェクトをあらゆる出力ストリームに書き出すことが可能であり，ネットワークを介してオブジェクトを送信する際は，ソケットに対して直接シリアライズすることができる．しかし，この方法にはいくつかの重大な欠点がある．第 1 に，ソケット毎にシリアライズしなければならないため，接続するノード数だけシリアライズが必要となり，スケーラビリティが確保できない．第 2 に，ソケットに対してシリアライズを行う場合，全データの送信が完了するまで制御が戻らないため，ネットワークの速度に応じた時間が必要となる．第 3 に，データを要求する全てのノードに送信が完了するまで，シミュレーションを中断しなければならない．最後に，途中参加したノードに対して，次の同期ポイントまで途中経過を配信することができない．

以上の理由から，ソケットに対して直接シリアライズする方法では，オーバーヘッドが上昇するだけでなく，大規模システム構築時のボトルネックとなることが予想される．

4.2 オンメモリシリアライズ

これらの問題は，一旦メモリ上にシリアライズし，得られたバイナリデータを必要に応じて送信することで解決できる．ソケットに対してシリアライズすると，300KB のデータで 600ms ～ 3,000ms 程度を要するのに対し，メモリ上で行うと 30ms 程度に抑えられる．また，シリアライズを行うのは一度であるため，スケーラビリティが確保でき，シミュレーションに即座に復帰できる．さらに，新規に参加したノードに対し最新のデータを即座に配信することができ，既に計算済みの箇所を再計算する無駄を省くことが可能になると同時に，システム全体の信頼性を高めることができる．

4.3 乱数生成器の配信

シミュレーションの分散化を行う時には，計算結果の再現性を考慮しなければならない．エージェントの意思決定を含めたあらゆる計算が決定的であるモデルなら問題はないが，何らかの確率的な要素を含むモデルでは，乱数系列の同一性が問題となる．提案システムでは，同一計算グループ内の全てのノードは同じ計算を行っているため，同一期間内に各ノードで取り出される乱数系列が同一ならばシミュレーションの同一性は保証され，計算結果の再現性が確保できる．今回，提案システムは Java で実装しており，Java の Random クラスはシリアライザブルであるため，シミュレーションの途中経過に含める形で乱数生成器を配信することができる．これにより，シミュレータの作成者は特に意識することなく計算結果の再現性を確保できる．

4.4 MAS への適用

提案システム上で MAS を実行するには，以下に示すメソッドを持つインターフェースをシミュレーションの実行クラスに実装し，関連する全クラスをシリアライザブルにするだけである．なお，ここでは MAS はステップ単位で処理を行うものとする．

getStep()	現在のステップを返す．
isSynchronizePoint()	同期するかどうかを返す．このメソッドが true を返したとき，提案システムはシリアライズを行い，getStep() で取得した値を送信する．
setStaticMembers()	提案システムはこのメソッドをデシリアライズ後にコールする．実装するクラスは，ここでクラスフィールドの初期化を行う．
next()	シミュレーションを一定期間実行するために繰り返しコールされる．一度にどれだけ進めるかは実装クラスに委任されるが，ここで進めるステップ数は 1〜isSynchronizePoint() が true を返す間隔の間でなければならない．シミュレーションが終了すると false を返す．
complete()	シミュレーション終了後，一度だけコールされる．実装クラスは，ここで終了処理を行う．

5. 実験

提案システムの有効性を検証するために医療機関選択 MAS を作成し，提案システム上で実行した．本節では，医療機関選択 MAS の詳細と提案システム上で得られた実行結果について述べる．また，リレー方式の有効性を確認するために，計算グループを構成するノードに意図的に外乱を与えた実験を行った．

5.1 医療機関選択 MAS

今回実装した医療機関選択MAS には医療機関と人という2 種類のエージェントが存在し，通院人数に応じて医療機関が分裂と消滅を繰り返すことにより，人口分布に対して最適な医療機関の配置と規模を求めるものである．エージェントは二次元の空間に存在しており，自らの位置を表す座標情報を持っている．この座標は平均0，標準偏差1 000 の正規乱数で初期化される．なお，本シミュレーションでは人は移動しない．

人は一定の確率で病気になり（患者），最も近い医療機関に通院する．各人の疾病率（人エージェントが病気になる確率）は，シミュレータにパラメータとして与える疾病率を中心とした正規乱数で初期化される．医療機関は，通院してきた患者を単位ステップあたりの治療可能人数まで治療する．治療可能人数の初期値は 50 である．治療されなかった患者は病気のままで，治療は次のステップに持ち越される．

各医療機関は10 ステップに1 度規模の見直しを行い，治療可能人数に余裕があれば縮小し，待ち状態の患者が多ければ拡大する．この時の増減の単位は 1〜5 の間の一様乱数で決定する．治療可能人数の上限は100 である．治療可能人数が30 以下の場合一定の確率で廃業し，治療可能人数が70 以上の場合一定の確率で新たな医療機関を作成する．

この時，新しい医療機関の位置は次の式で求められる．

$$H_{new}(\hat{x},\hat{y}) = H_{base}(x + r_1, y + r_2)$$

r1, r2 は-100〜100 の一様乱数で，新しい医療機関は基となる医療機関の周囲 200 × 200 のエリアに作成される．なお，新しい医療機関を作成する基となった医療機関の治療可能人数は50 にリセットされる．

5.2 シミュレーション結果の妥当

提案システム上でのシミュレーション結果の妥当性を検証するために，単体で実行した結果との比較を行った．シミュレーションのパラメータは，人数10 000，初期医療機関数10，シミュレーション期間10 000，疾病率は2 500 ステップ毎に0.10，0.15，0.20，0.15 で遷移，提案システム上では，同期間隔2 000 ステップである．実験に使用したノードを表1 に示す．提案システムでは1 台ずつ 4 種類のノードを使用した．実験結果は5 回の平均である．図3 はシミュレーションの各ステップにおける医療機関数を示したものである．2 500 ステップ単位で疾病率が変化するため，病院数に変動があることがわかる．提案システム上の結果と単体の結果の差分を取り，結果に相違が無いことを確認した．これにより，提案システム上で実行するシミュレーションの妥当性が証明され，再現性が保証された．

表 1 妥当性検証時のノード

Node	A	B	C	D
CPU	Core Solo U1500	Pentium 4 2.8GHz	Pentium 4 2.4GHz	Xeon 2.4GHz
Memory	2GB	512MB		2GB
Network	100base-T			
OS	Windows Vista Business	Windows XP Pro SP2	Windows XP Home SP2	Windows 2000 SP4

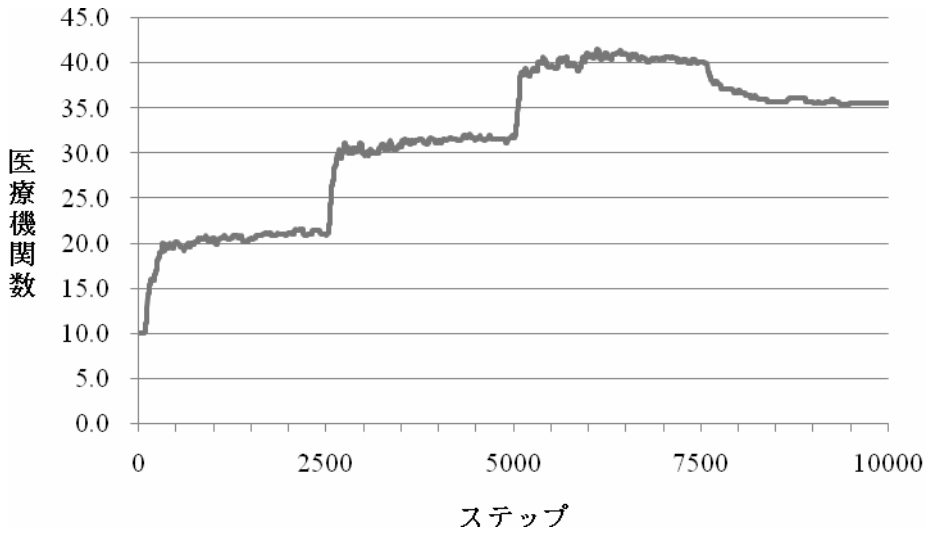

図 3 シミュレーション結果

5.3 リレー方式の検証

リレー方式によりシミュレーションが中断しないことを確認するために，意図的に外乱を与えて実験した．シミュレーションのパラメータと実験環境は基本的に前節と同様であるが，シミュレーションに時間をかけるために疾病率は 0.5 固定であり，同期間隔は 1 000 ステップに設定している．各ノードに与えた外乱のスケジュールを図 4 に示す．

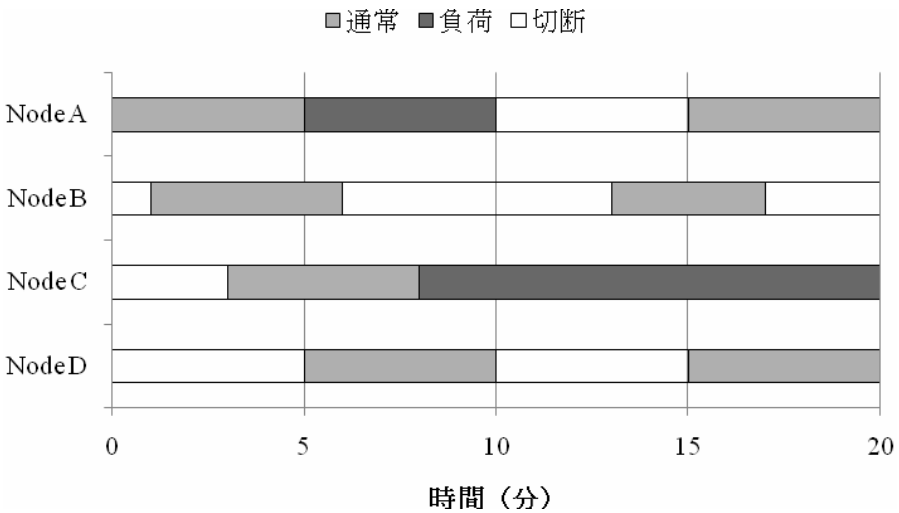

図 4 各ノードのスケジュール

通常はシステムに参加している状態, 負荷は参加しているが別プロセスが CPU を 100%使用している状態，切断はシステムから離脱している状態である．各ノードが処理したステップの状況を図 5 に示す.

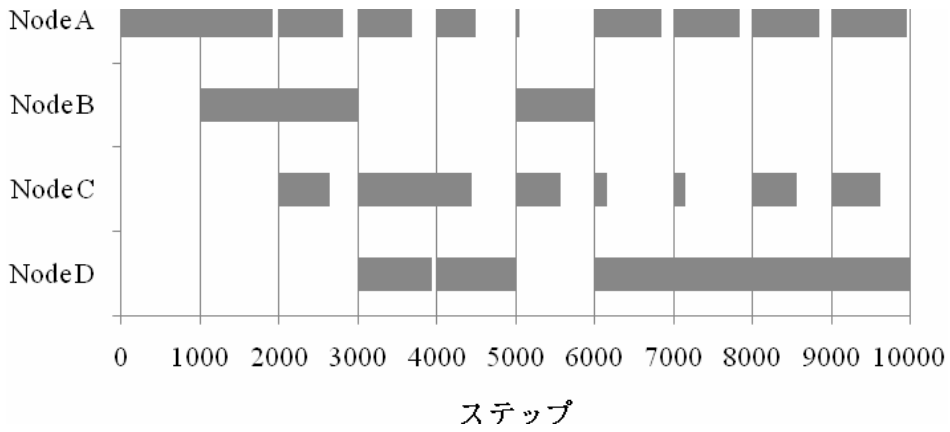

図 5 経過情報採用の遷移

この結果から，同期間隔毎にシミュレータが更新され，任意のノードが使用不可能になってもシミュレーション全体が安定して継続することが示された．また，高い処理能力を持つノードが参加した場合，そのノード上で多くの計算が行われ，シミュレーションが短時間で大きく進んだことが分かる.

5.4 ログ送受信と提案システムのコスト

今回の MAS では, 各ステップで医療機関毎に通院してきた患者数のログを取っている. 提案システムは，シリアライザブルなあらゆるデータを特別な処理なしに同期することが可能である. そのため, ログデータをシミュレーションデータと合わせて受け渡すことで, 各ノードのローカルストレージ上にログが分散することによる回収作業の煩雑さを回避することが可能である．しかし，ログデータはシミュレーションデータと異なり，後からシミュレーションに必要とされることはない．また，ログはシミュレーションの進捗に合わせて蓄積されていき，シミュレーション期間が長くなるにつれ肥大化していく．実際，前節のシミュレーションでは，最初の同期時には 400KB 程だったシリアライズされたデータが，回を増すごとに 200KB ずつ増加し，最終的には 2.2MB 程にまで肥大化した．本来シミュレーションに必要なデータが 260KB 程なので，総計で 10MB もの不要なデータを送受信していることになる．実験では 4 台で実行したため，最低でも 30MB 分の不要な通信が生じたことになる.

そこで，提案システム上のオーバーヘッドを検証するとともに，シミュレータサイズの増大に対するスケーラビリティを確認するために，A）同期間隔毎にログを書き出す場合と B）最後にまとめて出力する場合の提案システムと単体版との実行時間を比較した.

シミュレーションのパラメータは前節と同様，実験に使用したノードは表 1 のノード B × 1，ノード C ×2，ノード D × 1 である．単体版は予備実験の結果から最も処理能力の高いノード B で行った．実験は 5 回行った．実験結果を図 6 に示す．この結果から，ログを定期的に書き出す A では，提案システムと単体版の差は 8 秒であり，実行時間に対してたかだか 0.7%のオーバーヘッドしかないことが示された. 最後にまとめて書き出す B では，その差は 2 秒まで縮まり，オーバーヘッドは 0.2%でしかない．また，ログを含めることによるオーバーヘッドは 15 秒程度であり，シミュレータの大きさが 5 倍になっても実行時間は 1.4%しか増加しないことから，スケーラビリティが確認できた.

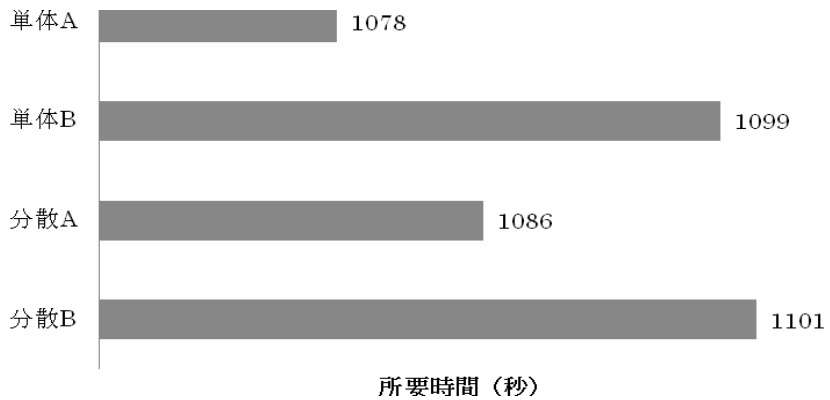

図 6 実行速度の比較

6. 関連研究

P3[8] は JXTA を用いたグリッド向けの P2P ミドルウェアである．P3 ではマスタワーカモデルとメッセージパッシングモデルを利用することができるが，計算開始後のノードの自由な参加離脱はマスタワーカモデルでしか認められていない．マスタワーカモデルでは部分問題への分割が必須であるため，提案システムのように単純にシミュレータを実装することは難しい．また，P3 の対象は数万台規模の計算機を用いたグリッドの構築であるが，我々はタスクを安全に処理できる程度の計算機を対象としている．Score[9] はメッセージパッシングを用いた並列処理環境であり，チェックポイントリスタートを用いた耐故障性を実現している．SCore ではチェックポイントデータを各ノードのローカルストレージに保管する．計算機の障害発生時には予備の計算機と交換することでグループを再構築し，並列タスクのリスタートを可能にするが，チェックポイントデータ保護のための冗長化によるオーバーヘッドが大きい[10]．また，計算結果の再現性については考慮されていない．

7. おわりに

本稿では，MAS における結合度を考慮した並列化・分散化について議論し，いずれの場合でも必要不可欠な計算の信頼性を高めるために動的なタスク複製を用いた分散システムを提案した．提案システムでは，複製間でリレー方式による同期を取ることで計算機の柔軟な参加・離脱を実現し，意図的に外乱を与えた実験により，デスクトップグリッドのような複雑な環境でも一定のパフォーマンスを得られることを実証した．提案システム上に MAS を適用することは容易でありながら，従来のチェックポイント機能により信頼性を高める研究と比較し，わずかなオーバーヘッドで耐故障性を実現できること，短期間の参加であっても高い処理能力を有する計算機を有効利用できることを示した．

提案システムは計算グループを用いることでタスクの信頼性を確保するが，複数の計算グループが存在する時，計算機の自由な参加・離脱により，計算グループ間で各々の信頼性に相違が生じる可能性がある．計算グループ間の信頼性を均衡化するために，新たに加わる計算機をどの計算グループに参加させるかについては今後の課題である．

謝辞

本研究の一部は，文部科学省社会連携研究推進事業（平成 17 年度～平成 21 年度）による私学助成を得て行われた．

参考文献

[1] Litzkow, M., Livny, M. and Mutka, M.: Condor – A Hunter of Idle Workstations, In Proceedings of the 8th International Conference of Distributed Computing Systems, pp. 104–111 (1998).

[2] Foster, I. and Kesselmany, C.: Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, Vol. 11, No. 2, pp.115–128 (1997).

[3] Anderson, D. P.: BOINC: A System for Public-Resource Computing and Storage, In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID '04), pp. 4–10 (2004).

[4] 上村佳史, 中島佳宏, 佐藤三久 : ヘテロな OS の計算資源を活用するグリッド RPC の設計, 情報処理学会研究報告, Vol. 2006, No. 87(HPC-107),pp. 269–274 (2006).

[5] 田中義久, 蟻川 浩, 村田忠彦 : 相互影響のあるタスク向けグリッド環境の提案, 情報処理学会研究報告, Vol. 2007, No. 17(ARC-172 HPC-109),pp. 209–214 (2007).

[6] 服部晃和, 薬師寺健太, 横田隆史, 大津金光, 古川文人, 馬場敬信 : 計算グリッド向けフォールトトレラントシステム Eagle の提案と初期評価, 情報処理学会論文誌 : コンピューティングシステム, Vol. 45, No. SIG 11(ACS 7), pp. 182–195 (2004).

[7] 谷村勇輔, 池上 努, 中田秀基, 田中良夫, 関口智嗣 : 耐障害性を考慮した Ninf-G アプリケーションの実装と評価, 情報処理学会論文誌 : コンピューティングシステム, Vol. 46, No. SIG 7(ACS10), pp. 18–27 (2005).

[8] 首藤一幸, 大西丈治, 田中良夫, 関口智嗣 : 計算資源の流通および集約のための P2P ミドルウェア, 情報処理学会論文誌 : コンピューティングシステム, Vol. 45, No. SIG 6(ACS 6), pp. 208‐221(2004).

[9] Hori, A.: SCore: An Integrated Cluster System Software Package for High Performance Cluster Computing, Proceedings of the 3rd IEEE International Conference on Cluster Computing, pp. 449(2001).

[10]林田卓郎, 鈴木優介, 近藤正章, 今井 雅, 中村 宏, 南谷 崇, 堀 敦史 : SCore クラスタシステムにおけるチェックポインティング機構の性能評価, 情報処理学会研究報告, Vol.2003, No.29(HPC-93),pp. 25–30 (2003).