PG Lab ディスカッションペーパーシリーズ
第24号　2008年2月

ISSN 1880-9146

Discussion Paper Series
No.24 February, 2008

断続的な計算機利用を可能にする分散計算システムの開発

田中義久，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).

断続的な計算機利用を可能にする分散計算システムの開発

田中義久 [1]，前田太陽 [2]，村田忠彦 [1,2]

DEVELOPMENT OF DISTRIBUTED COMPUTING SYSTEM FOR INTERMITTENTLY AVAILABLE COMPUTING RESOURCES

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

概要

我々は，断続的に利用可能な計算資源や遊休状態にある計算資源の効率的な利用を目的とした分散計算システムを開発した．本システムでは，動的に分散した個々のノードが自律的にジョブの複製を行う．これにより，本システムは耐故障性を実現し，特定のノードを占有することなくジョブを実行することが可能である．本稿では，適用事例として協調行動シミュレーションと 3 次元 MD シミュレーションを行った．その結果，本システムは複数ノード間でのジョブの遷移を実現し，任意のノードに障害が発生してもジョブを中断することなく実行可能であることが示された．本システムを適用することによる計算時間に対するオーバーヘッドはわずか 1%程であった．

Abstract

In this paper, we propose a distributed computing system in order to effectively use intermittently available or idle computing resources. Our proposed system clones a running application at every checkpoint in order to realize the fault tolerance. Through experiments in our testbed environment, we find that the overhead time for using the proposed system is only 1%.

キーワード : 分散, フォールトトレランス, Java

Keywords: Distributed, Fault tolerance, Java

1 関西大学総合情報学部　Faculty of Informatics, Kansai University

2 関西大学政策グリッドコンピュティング実験センター　Policy Grid Computing Laboratory, Kansai University

1. はじめに

長時間を要するシミュレーションを安定して実行するためには高い稼働率を有する計算機が必要となるが，一般的に計算機はいつかは必ず故障するものであり，突発的に生じるハードウェアトラブルや自然災害からシミュレーションを保護することはできない. また，専用の計算機を用意できない環境では，他プロセスやユーザの介入により，予期しない中断が生じる可能性も考慮する必要がある．そのため，計算機の障害からシミュレーションを保護することを目的としたチェックポイントに関する研究が従来より行われてきた.

チェックポイント機構を有する代表的なシステムとして Condor [1]がある．Condor では定期的にチェックポイントを設けることで耐障害性を実現し，任意のタイミングでのチェックポインティングを可能にすることで，ワーカ間のジョブマイグレーションを実現している．しかし，チェックポイント機構を利用するためには，C, C++, FORTRAN でシミュレータを開発し，専用コマンドを用いてコンパイルする必要がある．また，ワーカは Unix 系 OS を搭載した計算機に限られる.

Ninflet [2]は Java を用いることでプラットフォーム非依存を実現したジョブ管理システムであり，ジョブを割り当てる Dispatcher，クラス定義を提供する Web Server，計算リソースを提供する Server により構成される．Ninflet を用いてプログラムを実行するためには，手元で実行されるクライアントプログラムと Ninflet 上で実行される Ninflet プログラムを個別に作成し，Ninflet プログラムを事前に Web Server に配置しておく必要がある．そのため，Ninflet を利用するためには専用の知識が必要となる.

チェックポイント機構を実際に利用するためには，Condor や Ninflet のようなシステムを導入する必要があるが，ジョブの実行に中央サーバを必要とするアプローチはシステム導入に大きな労力を強いることになる．また，システムの稼働率を考慮すると，サーバが単一故障点になるというデメリットは無視できない．計算の耐障害性については他にも様々な議論[3-5]が行われてきたが，いずれも大規模システムの導入を前提としている．特別なシステムの導入や環境の構築を行わず，計算に十分な時間だけ計算機を占有できない状況であっても，複数台の計算機による動的な構成が可能なシステムを実現できれば，常にいずれかの計算機上でジョブを処理し，特定の計算機を占有することなくジョブを完遂することが可能となる．そこで本研究では，動的なジョブの複製を個々の計算機で行うことで計算のミラーリングを実現し，複数の計算機を断続的に利用して 1 つの計算を行う分散計算システムを開発した．本稿では，システムの詳細について述べた後，適用事例として協調行動シミュレーションと 3 次元 MD シミュレーションを行い，システムの評価と有効性を議論する.

2. 提案システム

本システムは文献[6]で MAS（Multi-Agent Simulation）向けに提案したシステムを汎用化し，改良を施したものである．実装にはマルチプラットフォームで動作可能であり，オブジェクト指向により堅牢なシステムを構築可能な Java を用いた．

2.1 処理の流れ

本システムでのシミュレーション実行の流れを図 1 に示す.

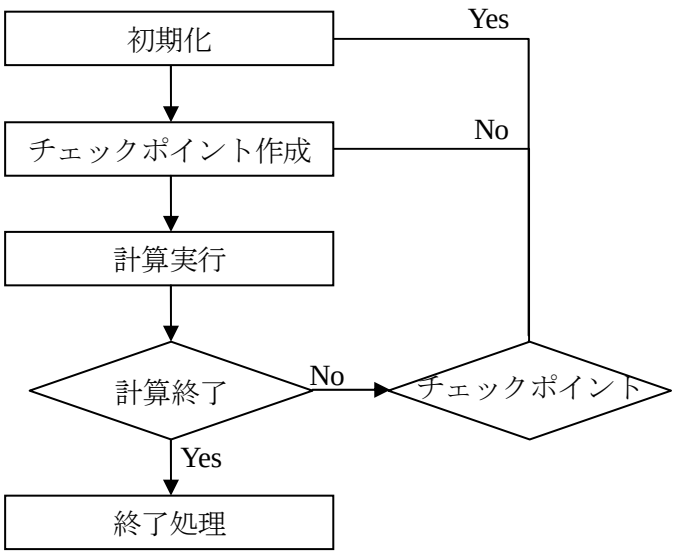

図 1 処理の流れ

シミュレータのインスタンスはシステムによって作成され，初期化される．続いてシステムによって自動的にチェックポイントが作成される．初期化済みの状態のチェックポイントを作成することで，初期化方法を知らない他のノードを即座に利用することが可能になるとともに，毎回異なる初期状態を生成する初期化コードを持つシミュレータにも対応可能となる.

次に，システムはシミュレータの計算ルーチンを実行する．このルーチンは計算の最小単位であることが望ましい．例えば，ループ処理を行っている場合はループの中身だけを記述する. システムはこのルーチンを指定回数実行した後, チェックポイントを作成する. 全ての計算が終了すると，シミュレータの持つ終了処理が実行され，シミュレーションは終了する.

2.2 システムの適用方法

シミュレータへのシステムの適用は，システムの提供するインターフェースを実装したクラスを作成し，関連する全クラスをシリアライズ可能にすることで実現できる．このインターフェースを図 2 に示す．なお，UpdateListener インターフェースを実装し，同クラスをシステムに登録することで，シミュレータ更新時に独自の処理を追加することが可能である．

```
public interface Sequential extends Serializable {
    public boolean proceed();
    public void onComplete();
}
```

図２ シミュレーションのインターフェース

proceed()には繰り返し行う計算処理を記述する. 全ての計算が終了した時に false を返し，そうでなければ true を返す．このメソッドが true を返す間，システムはこのメソッドを繰り返し実行し，指定回数実行後にチェックポイントを作成する．

onComplete()には必要であれば終了処理を記述する．このメソッドは，シミュレーション終了時に一度だけ実行される．

シミュレータの初期化は図 2 のインターフェースを実装したクラスのパブリックコンストラクタにて行う．String[]型を引数に取るコンストラクタを作成することで，引数を与えて初期化することも可能である．

2.3 シミュレーションの実行

シミュレーションを提案システム上で実行するには，本システムを適用させたシミュレータを JAR ファイルにまとめた上で，ランタイムプログラムに JAR ファイルへのパスと図 2 のインターフェースを実装するクラス名，シミュレーション固有の引数をパラメータとして渡す．ランタイムプログラムは他にも，チェックポイント間隔，待機ポートをパラメータに取る．システムは JAR ファイルから必要なクラスをロードし，リフレクションを用いて与えられたクラス名からインスタンスを生成する．

2.4 シミュレータの更新

提案システムはノードが 1 台の状態からシミュレーションを実行することが可能である

が，複数台のノードを用いることで計算のミラーリング機能を有効化できる．本システムは，これにより耐障害性を実現している．システムは，各チェックポイントでシミュレータをメモリ上にシリアライズすることで，その時点のシミュレータのスナップショットを保存する．このスナップショットをシステムを構成する全てのノードで共有することで，計算のミラーリングを実現する．

新たにシステムに参加したノードに対して，参加を受け付けたノードはまず，シミュレータの JAR ファイルを転送し，チェックポイント間隔を通知する．続いて，最新のスナップショットを転送する．以降は各ノードが独立してシミュレーションを実行し，チェックポイント毎にチェックポイント番号を通知し，必要に応じてスナップショットを転送，シミュレータを更新，リスタートする．これにより，システムを構成する全てのノードが常にシミュレータの最新の状態を共有する．

2.5 ネットワークの構成

提案システムは P2P 型のネットワークを用いて各ノードを接続している．P2P 型のメリットとして，単一故障点の排除がある．また，サーバを用意する必要がないため，特別な環境の構築が不要なシステムの開発という本研究の目的にも合致する．P2P 型でネットワークを構築するために，新たなノードの参加を受け付けたノードは，その情報を他のノードに通知し, 各々のノードが新たなノードとの接続を確立し, ネットワークを再構成する．システムは実行中のノードの変動を許容し，ランタイムプログラムを実行するだけでシステムに参加できるため，必要に応じて容易にノードを追加することが可能である．

3. 適用事例

提案システムを評価するために，協調行動シミュレーションと 3 次元 MD シミュレーションの両プログラムを改変し，提案システムを用いて実行した．5 秒毎に各ノード上のシミュレーションの進捗状況をプロットすることにより，システムの評価を行う．評価には表 1 に示すノードから最大で 4 台を使用した．

表 1　ノードの諸元

CPU	Pentium 4 3.4GHz
Memory	2GB
Network	1000BASE-T
OS	Windows XP Pro SP2
JRE	JRE 6.0 update 5

3.1 協調行動シミュレーション

協調行動シミュレーションのひとつである QDSEGA を用いた MAS [7]を実行した．QDSEGA は Q 学習の Q テーブル更新に GA を用いることで，Q テーブルの縮小を目指したアルゴリズムである．オリジナルのソースコードは 13 ファイル，1347 行である．このシミュレータを本システムに適用するためのソースコードの追加・変更箇所は 242 行であった．変更箇所の大部分はシミュレータのループの解体に費やしており，オリジナルのループは近傍パラメータ，試行回数，世代の 3 重にネストしている．これをひとつのカウンタ変数を用い, 条件判定によりループの開始, 終了時の処理を行うように変更した. また, ループ内で用いられているローカル変数全てをクラスのプライベートフィールドに設定した．図 3 に各ノード上での処理経過を示す.

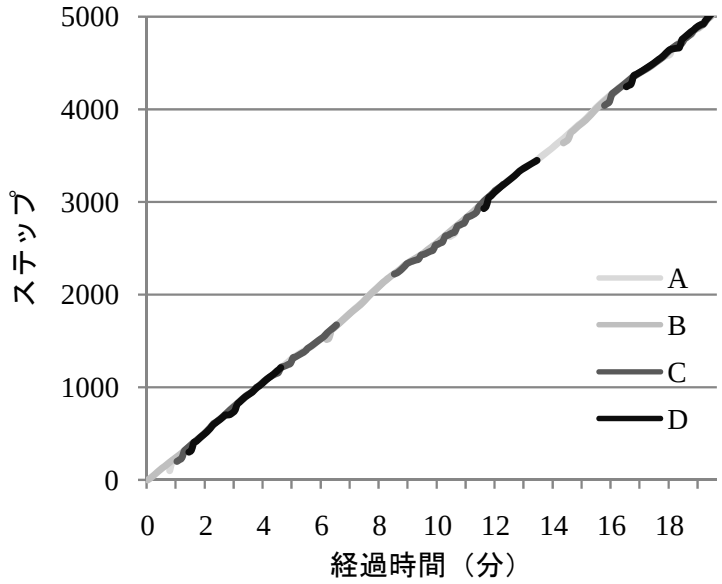

図 3　協調行動シミュレーションの処理経過

横軸が経過時間，縦軸がステップであり，各ノードを表す直線の傾きが急な程，単位時間あたりの処理が高速に進んだことを示している．今回は 1 世代を 1 ステップとし，全体を通して継続的にカウントしている．1 試行が 101 世代であり，各パラメータについて 10 試行を行い，パラメータを 5 通りに変化させているため，5050 ステップで終了となる．チェックポイント間隔は 101 ステップ，すなわち 1 試行毎とした.

図 3 からはシミュレーションがノード B 上で開始され, その後に参加したノード A, C, D が最新のチェックポイントから計算を開始したことが読み取れる．また，ノード C を表す線は 10 分前後でギザギザになっている．これは，一時的にノード C の処理速度が低下し，傾きが緩やかになったが，ノード B がチェックポイントに到達する度にシミュレータ

の更新処理が行われ，ノードC の処理状況が最新のチェックポイントまで飛躍し，その時点を表す傾きが急になったためである．さらに，14 分前後ではノードA のみで処理を行っており，計算を開始したノードB が使用不可能になっても，シミュレーションが安定して継続することが示された．なお，計算結果については，本システムを用いずに単独で実行した結果との差分を取ることにより，同一であることを確認した．

3.2　3 次元 MD シミュレーション

MAS 以外のシミュレーションとして，3 次元 MD シミュレーションを実行した．オリジナルのソースコードは 3 ファイル，364 行であり，追加・変更箇所は 28 行であった．本シミュレータはタイムステップをカウントする単一のループであったため，タイムステップをプライベートフィールドに設定し，条件分岐により終了判定を行うように変更した．そのため，協調行動シミュレーションと比較して，システムの適用は容易であった．シミュレーションのパラメータは，分子数 3072，タイムステップ 2000 である．チェックポイント間隔はタイムステップ 100 毎に設定した．なお，差分を取ることにより結果が同一であることを確認した．

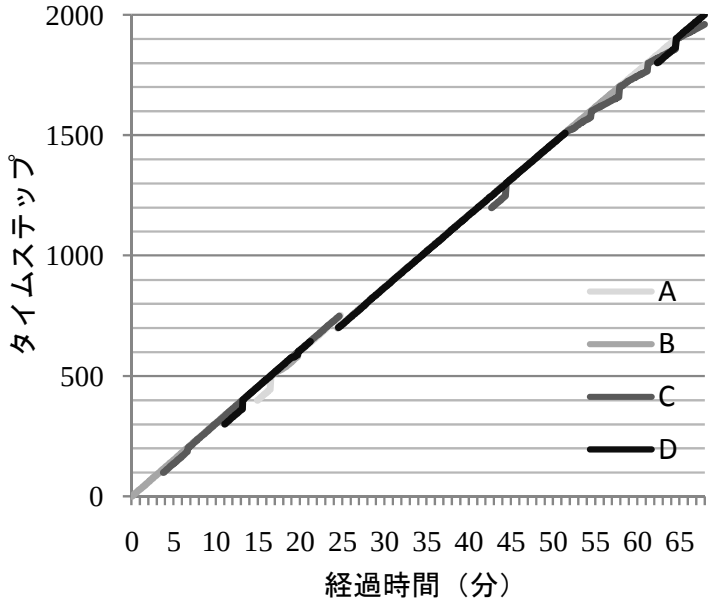

図 4　3 次元 MD シミュレーションの処理経過

図 4 は各ノード上での処理経過であり，タイムステップ 100 毎，すなわちチェックポイント毎に目盛り線を追加している．この結果から読み取れるシステムの振る舞いは基本的には協調行動シミュレーションと同一であるが，25 分前後に本システムの特徴的な挙動を

観測できる．ノード A，C がチェックポイント半ばでシステムから離脱したため，各ノード上でチェックポイント以降に行われた計算は破棄されたが，直前にノード B，D が参加し，シミュレータの最新のスナップショットを取得していたため，それ以前に処理された 23 分間分の計算を引き継ぐことができた．このことから，別のノードを用意してミラーリングを行うことで，任意のノードをシステムから離脱させ，他の用途に用いることが可能であることが示された．

3.3　システムのオーバーヘッド

提案システムを用いることによる実行時間に対するオーバーヘッドを検証するために，協調行動シミュレーションと 3 次元 MD シミュレーションを提案システムを用いずに単独で実行し，実行時間を比較した．単独実行に使用したノードは提案システムを用いた場合と同一である．それぞれの実行時間を図 5 に示す．

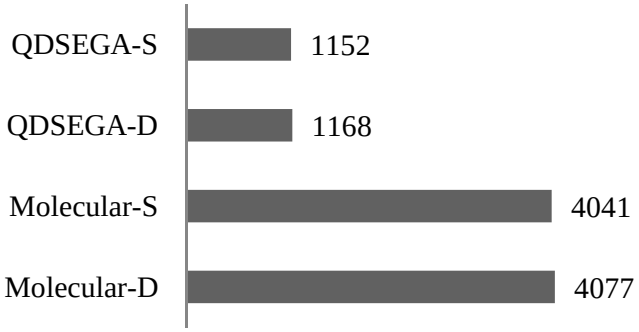

図 5　システムのオーバーヘッド

-S は単独実行，-D は提案システムを用いた実行を表している．図中の数字は実行時間（秒）である．単独実行の結果は 3 台のノードでそれぞれ実行した結果の平均値である．この結果から，実行時間に対するオーバーヘッドは，協調行動シミュレーション（QDSEGA）で 1.4%，3 次元 MD シミュレーション（Molecular）で 0.9%であり，提案システムを用いることによるオーバーヘッドはわずかであることが示された．

3 次元 MD シミュレーションを単独で実行した際の実行時間はそれぞれ，4176 秒，3975 秒，3971 秒であり，3 台のノードのうち 1 台の実行時間が提案システムを用いた場合よりも遅いという興味深い結果が得られた．当該ノードの処理性能が他のノードに劣っていたことは，同一のハードウェアであっても，ノードの使用状況により処理性能が低下していたためと考えられるが，ここで注目すべきは本システムがシミュレーションの高速化の可能性を持つことである．今回の結果は，各チェックポイントでシミュレータの最新のスナップショットを共有する本システムの更新アルゴリズムが上手く作用し，処理の遅いノー

ドを処理の早いノードが牽引したことによるものである．

4. おわりに

本稿では，動的に計算の複製を行うことで計算の信頼性を高める分散計算システムとその適用事例について報告した．提案システムは少ないコードの変更でシミュレータへ適用することが可能でありながら，従来の研究と比較してわずかなオーバーヘッドで耐障害性を実現すること，特定の計算機を占有することなく，細切れの時間を用いて処理を完遂できることを示した．また，計算途中でより高速な計算機を用いて計算をミラーリングすることで，スループットの向上が期待できる．本システムは特別な導入作業が不要で，実行中の計算機の動的な参加・離脱を許容する柔軟さを持つため，必要に応じて耐障害性を確保することができ，シミュレーションを任意の計算機に移動することも可能である．

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 ComputingSystems, pp. 104-111, 1998.

[2] 高木浩光, 松岡 聡, 中田秀基, 関口智嗣, 佐藤三久, 長嶋雲兵 : Ninflet: Java による World-Wide High Performance Computing 環境, インターネットコンファレンス論文集, Vol. 1997, pp. 133-147, 1997.

[3] 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. SIG11 (ACS 7), pp. 182-195, 2004.

[4] 中田秀基, 田中良夫, 松岡 聡, 関口智嗣 : 耐故障性を重視した RPC システム Ninf-C の設計と実装, 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.SIG11 (ACS 7), pp. 160-170, 2004.

[5] 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 (ACS 10), pp. 18-27, 2005.

[6] 田中義久, 蟻川 浩, 前田太陽, 村田忠彦：エージェント間結合を考慮した MAS 分散処理の実装, 情報処理学会研究報告, Vol. 2007, No. 80 (HPC-111), pp. 243-248,2007.

[7] Tadahiko Murata, Yusuke Aoki: Developing Control Table for Multiple Agents Using GA-Based Q-Learning with Neighboring Crossover, Proc. of IEEE Congress on Evolutionary Computation: CEC 2007, pp. 1462-1467, 2007.