グリッド環境における チェックポインティング手法の検討と初期評価

薬師寺 健太† 服部 晃和† 横田 隆史† 古川 文人‡ 大津 金光† 馬場 敬信†
†宇都宮大学工学部情報工学科
‡宇都宮大学サテライト・ベンチャー・ビジネス・ラボラトリー

1 はじめに

近年、大規模化・複雑化の一途を辿る計算問題に対応するためのインフラストラクチャとして、グリッド[1]技術が注目されている。一般に、大規模な計算資源を利用し長時間に渡って計算を行う場合、システム内に発生した障害(フォールト)のため計算を完了できなくなる可能性が高くなる問題がある。我々はこの問題の現実的な解決のために、高信頼化グリッドシステムEagle[2]を提案している。本システムはpessimistic loggingをベースにしたものであり、各計算ノード上の計算プロセスの実行イメージをチェックポイントデータとして定期的に取得し安全に保持(チェックポインティング)する。障害発生時には保存チェックポイントデータを元にプロセスを回復(ロールバック)する。ロールバックによるペナルティを抑えるためにはチェックポイント間隔は短いほうが望ましい。一方、チェックポイントデータはプロセスの実行イメージであり、一般には巨大なデータ量となる。このため、頻繁なチェックポインティングは、ネットワークトラフィックの増加を始め、本来の計算に影響を与える。

そこで本稿では、様々なチェックポインティング手法における並列分散プログラム実行時のオーバヘッドを計測、評価することで、Eagleに有効な手法を検討する。

2 チェックポインタ

2.1 チェックポインタの分類

現在、種々のチェックポインタが実装されているが、チェックポインタは大きく分けて以下の2つのレベルで実装される。

2.1.1 カーネルレベルのチェックポインタ

カーネルレベルのチェックポインタは、OSのカーネルモジュール、あるいはカーネル自身に実装される。チェックポインティング時の動作が高速であり、ファイル管理情報などのカーネル領域のデータがチェックポインティング可能である。しかし、実装は容易ではなく、特定のOSでしかチェックポインティング/リスタートができないといった移植性にかける欠点がある。

2.1.2 ユーザレベルのチェックポインタ

ユーザレベルのチェックポインタは、ユーザープログラムのランタイムライブラリとして実装され、コンパイル時または実行時にユーザコードとリンクすることにより、チェックポインティングの機能を実現する。このチェックポインタはカーネルレベルで実装されるチェックポインタに比べ動作が遅く、カーネル領域のデータは保存することができない欠点がある。しかし、チェックポイントデータを抽象化することによって、グリッド環境における異なるOSといったヘテロ性を許容することができる。

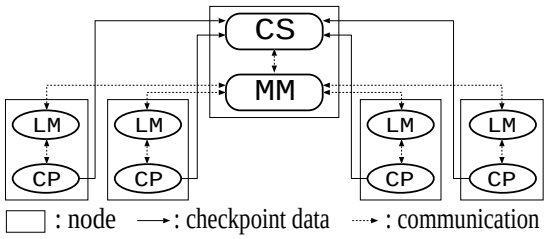

図 1: Eagleのチェックポインティング実装部

2.2 ckpt

我々は移植性という観点から、ユーザレベルで実装されるチェックポインタckpt[3]を用いる。ckptはプロセスのチェックポインティング/リスタートを行うためのライブラリであり、実行時にダイナミックリンクされる。チェックポインティングする項目はプロセスのメモリイメージ、シグナル状態、CPUのコンテキストの3つである。

次に実装方法を説明する。まずckptはメモリにロードされた段階で_initを実行し、チェックポインティング処理を行う関数をシグナルハンドラに登録する。ユーザコードはシグナルを受け取ると、実行中の計算を一時中断しチェックポインティングを開始する。setjmpを使いCPUのコンテキストを保存する。そして、チェックポインティング終了後に通常計算に戻る。リカバリ時にはlongjmpを使いCPUのコンテキストを復元し、リスタートする。

3 チェックポインティング手法

Eagleのチェックポインティング実装部の構成を図1に示す。このシステムはckptがリンクされるCP(Computation Process)、チェックポイントデータを受け取るCS(Checkpoint Server)、CPとの通信、およびCPの状態を監視するLM(Local Monitor)、LMを介して全てのCPと通信するMM(Master Monitor)から構成される。CSとMMを有するノードがチェックポイントサーバであり、CPとLMを有するノードが計算ノードである。次に、計算時間のオーバヘッドの要因となる2つの問題と、それを解決するチェックポインティング手法を説明する。

3.1 チェックポインティング時のプロセス複製

CPはチェックポインティングしている間、計算が一時中断してしまうため、チェックポインティングにかかる時間そのものが計算時間のオーバヘッドとなる。このオーバヘッドを削減する手段としてチェックポインティング時のプロセス複製が挙げられる。実装方法としてチェックポインティング起動時にUnixのforkシステムコールを使ってプロセスを複製する。複製する側のプロセスはすぐにシグナルハンドラから通常計算に復帰して、複製されたプロセスはチェックポインティング完了後に終了する。この手法はチェックポインティングが通常計算の進行をブロックしないことから、Asynchronous Checkpointing(AC)と呼ばれている。これに対して、1プロセスでチェックポインティングする手法はSynchronous Checkpointing(SC)と呼ばれている。

---

* An Analysis and Preliminary Evaluation of Checkpointing Methods for Grid Environment
Kenta Yakushiji†, Akikazu Hattori†, Takashi Yokota†, Fumihito Furukawa‡, Kanemitsu Ootsu†, and Takanobu Baba†
† Department of Information Science, Faculty of Engineering, Utsunomiya University
‡ Satellite Venture Business Laboratory, Utsunomiya University

3.2 チェックポインティング時期のスケジューリング

並列計算時において、チェックポインティング時の計算ノードの処理速度低下が他計算ノードの処理速度低下につながる。このとき、いくつの計算ノードが同じタイミングでチェックポインティングを行うかによって処理速度の低下は変化する。またチェックポイントサーバが各計算ノードから同時に複数のチェックポイントデータを受け取る場合、負荷がかかり、チェックポイントサーバのI/O待ちにつられて計算ノードの処理速度が低下する。この2つの計算処理速度低下は計算時間のオーバヘッドを招く。この計算処理速度低下を防ぐために各CPのチェックポインティング時期のスケジューリングが必要になる。チェックポインティング時期のスケジューリング方法として我々は以下の3つの手法を提案する。

3.2.1 Sequential Request

Sequential Request(SR)は予め設定された一定時間毎のインターバルに従い、MMがLMを介し各CPに対して逐次チェックポインティング要求を出す手法である。この手法ではチェックポイントサーバがチェックポイントデータを受信・保存する負荷が平均化される。各計算ノードには1ノード毎にチェックポインティングの負荷がかかっていく。

3.2.2 Batch Request

Batch Request(BR)は予め設定された一定時間毎のインターバルに従い、MMがLMを介し各CPに対して一斉にチェックポインティング要求を出す手法である。この手法では各計算ノードからチェックポイントデータが同時に送られるため、これらを受信するチェックポイントサーバには一時的に大きな負荷がかかる。各計算ノードには同時にチェックポインティングの負荷がかかる。

3.2.3 Self Timer

Self Timer(ST)は予め各CPに一定時間のインターバルを設定し、そのインターバル毎に自律的にチェックポインティングを開始する手法である。チェックポインティング時期のスケジューリングに伴う通信が不要になるために、ネットワークのトラフィックを軽減できる。

4 評価

評価には3.1と3.2で説明したチェックポインティング手法の組み合わせを用いる。チェックポインティング手法はSC、AC、SR、BR、STの5つあるが、SCとAC、SRとBRとSTは互いに排他的なチェックポインティング手法である。よって、評価対象のチェックポインティング手法の組合せはSC&ST、SC&SR、SC&BR、AC&ST、AC&SR、AC&BRの6種類となる。実験にはチェックポイントサーバ1台、計算ノード4台を用いた。実験に用いた各ノードの性能を表1に示す。並列計算ライブラリとしてMPICH-G2、対象プログラムはNAS Parallel Benchmarks(NPB-2.3)のBT、問題クラスをAとし、各計算ノードに1プロセスを割り当てた。また、オブジェクトコードのサイズ、1プロセスあたりのチェックポイントデータのサイズ、チェックポインティングなしでの実行時間を表2に示す。6つのチェックポインティング手法の組合せに対して、チェックポインティング回数1、2、3回の場合のチェックポインティングにかかる時間的オーバヘッドを図2に示す。

(1)ACはSCに比べて、どのチェックポインティング手法の組み合せ、チェックポインティング回数においても、オーバヘッドが小さい。チェックポインティング回数3回のときのSC&BRとAC&BRを比べると、70.7%のオーバヘッドが2.3%になっている。このことよりACの有効性が示された。

表 1: ノード性能

CPU	Intel Celron 2.0[GHz] × 1
Memory	512[MB]
Network	100Base-T
OS	Linux 2.4.18

表 2: NPB-2.3

Executable Size	1167.7[KB]
Checkpoint Size	89.51[MB]
Normal Computation Time	300.00[s]

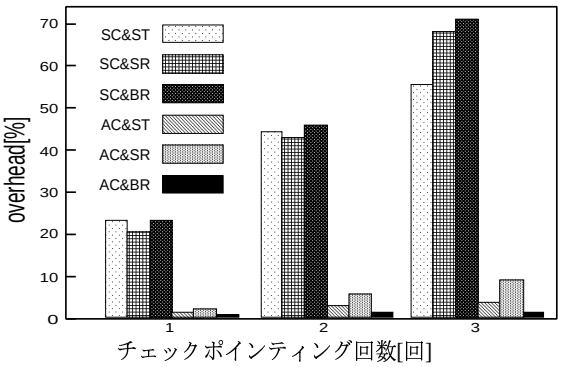

図 2: オーバヘッドの評価

(2)AC&BRはAC&SRに比べて、どのチェックポインティング回数においてもオーバヘッドが小さい。このことから、並列計算する複数の計算ノードに対して、1ノード毎にチェックポインティングの負荷をかけていくよりは、同時に複数の計算ノードに対してチェックポインティングの負荷をかけたほうが計算実行時間が短くなると考えられる。

(3)AC&BRとAC&STを比べると、チェックポインティング回数が1、2、3回と増える毎に、オーバヘッド値の差分が増えている。チェックポインティングの開始時間を解析した結果、AC&STは各CPの最初のチェックポインティング開始時間は同一であるが、チェックポインティング終了時間は別になり、2回目以降のチェックポインティングの開始時間がずれていた。このことから、各CPのチェックポインティング開始時間のずれがオーバヘッドを招いたと考えられる。

5 おわりに

本稿では、グリッド環境を対象とした並列計算においてのチェックポインティング手法を検討し評価した。今後はEagleにとってより有効なチェックポインティング手法を実現し、評価する。そしてそのチェックポインティング手法を導入したEagleを構築する。

謝辞 本研究は，一部日本学術振興会科学研究費補助金(基盤研究(B)14380135, 同(C)14580362, 若手研究14780186)の援助による.

参考文献

[1] I. Foster, C. Kesselman and S. Tuecke: “The Anatomy of the Grid: Enabling Scalable Virtual Organizations,” International Journal Supercomputer Applications 15(3):200-222(2001.1).

[2] 服部 晃和, 横田 隆史, 大津 金光, 古川 文人, 馬場 敬信: “大規模グリッド向けフォールトトレラントシステムEagleの提案” 先進的計算基盤システムシンポジウム SACSIS2003:185-186 平成15年5月

[3] Victor C. Zandy : “ckpt: A process checkpoint library” http://www.cs.wisc.edu/~zandy/ckpt