MPI/FT: A Model-Based Approach to Low-Overhead Fault Tolerant Message-Passing Middleware

著者： R. Batch, Y. S. Dandass, A. Skjellum and M. Beddhu
出典： *Cluster Computing, Springer Netherlands, Vol. 7, No. 4, pp. 303–315, 2004*
発表者： 高性能コンピューティング学講座 本多・近藤研究室 0853020 松本優人

1 序論

MPI に対して小さいオーバーヘッドでプロセス故障の検知・修復機能を付加したミドルウェア、MPI/FT の設計と実装を目指し、MPI-1.2 標準に準拠したソフトウェア MPI/Pro 1.51[3] をベースとして MPI/FT を実装する。

MPI とは、並列分散システムのプログラミング規格である。MPI 標準は通信機器や計算機器の故障を定義していないために、耐故障システムを構築したい場合は MPI 以外の技術を付加しなければならない。

上記のような理由から、MPI に耐故障性を付加したミドルウェアが数多く研究されている。Egida[1] と呼ばれるツールキットは、通信ログをとることで、障害時にログを元にロールバックして回復する。MPI ミドルウェアである FT-MPI[2] は、アプリケーションにプロセス故障を通知する方式により、ユーザが柔軟に故障検知・回復を設定できる。

2 目的・設計

従来の耐故障 MPI ミドルウェアは固定された耐故障サービスを提供しているために、アプリケーションの要件によっては不必要にオーバーヘッドが大きくなることがある。本稿では、商用パッケージソフト MPI/Pro 1.51 をベースとし、耐故障 MPI ミドルウェア MPI/FT を実装する。MPI/FT は数種類の実行モデルを提供し、ユーザはアプリケーションの要件に従って実行モデルを選択する設計としている。これによって、オーバーヘッドが小さくなるようにユーザが調整できる。また、開発にかかる労力を低減することができる。

MPI/FT は、表 1 に挙げる実行モデルを提供する。

表 1: 実行モデル

名称	通信構造 (スタイル)	監視プロセスの冗長性	実装
Model-Ia	Star	なし	○
Model-Ib	(Master-slave)	passive	×
Model-Ic		active	×
Model-IIa	All-to-all	なし	○
Model-IIb	(SPMD)	passive	×
Model-IIc		active	×

MPI/FT が提供する実行モデルは、Model-I および Model-II の 2 つに大別することができる。Model-I は、通信構造が Star 型、すなわちサーバ／クライアントのような通信構造を持つ場合に選択するモデルである。Model-II は、通信構造が All-to-all 型、すなわち各計算プロセスが互いに独立して通信できる構造を持つ場合に選択するモデルである。通信構造が異なる場合、一般的にコーディングスタイルも異なる。Star 型の通信構造では、Master プロセスと Slave プロセスは異なるプログラム (ソースコード) を用意する。一方、All-to-all 型では、すべてのプロセスは同じプログラム (ソースコード) であり、条件分岐を用いてそれぞれ異なる処理を行う。ユーザはこれらの通信構造の違いを意識して実行モデルを選択することで、故障検知・回復方法を適切に行うことができる。

各プロセスは、ユーザが定義するアプリケーションスレッド (AT) の他に、セルフチェッキングスレッド (SCT) を持つ。それぞれの SCT が監視スレッド (Coodinator Thread) に対してある時間間隔で heartbeat メッセージを送信することで、プロセスが正常に動作しているかを確認する。

Model-Ia では、Master プロセスの SCT が Coodinator Thread となり、Slave プロセスの故障を検知する。故障が検知されたとき、以下のような手順で回復する。

1. Coodinator Thread が Slave プロセスの故障を検知すると、Master プロセスの AT に故障通知を行う。また、故障プロセスに対するメッセージキューをクリアし、故障プロセスとの接続を無効にする。通知を受けた AT は、`MPIFT_GET_DEAD_RANKS()` メソッドを呼び出して障害の状態を取得する。
2. 障害の状態を得た AT は、`MPIFT_RECOVER_RANK()` メソッドを呼び出して回復における初期化処理を開始する。まず、Coodinator Thread が予備のプロセスに対して初期化処理を行うように指示し、予備プロセスは初期化処理を行う。さらに、予備プロセスは AT 間の接続、および SCT 間の接続をセットアップする。
3. Master プロセスの AT および Coodinator Thread は、予備プロセスからの接続要求を受ける。
4. Master プロセスの AT は、予備プロセスへ新しいジョブを送信する。

Model-Ia は、Master プロセスを監視するプロセスがいない。すなわち Single point of failure(SPOF) な耐故障性しか有していない。この問題点を解消したものが Model-Ib および Model-Ic である。これらはチェックポインティング方式を用いて、Master プロセスの故障検知・回復を実現する。Model-Ib は、Master プロセスが故障したときに予備

のプロセスを初期化し、故障プロセスと置換する。Model-Ic は、Master プロセスを複製しておくことで、Model-Ic で必須だった初期化処理を省略する。しかし、Model-Ib および Model-Ic はまだ実装が完了していない。

Model-IIa の故障検知は、Model-Ia のそれとほぼ同じ方法で行う。具体的には、rank0 のプロセスの SCT が Coodinator Thread となり、他のプロセスを監視する。故障の回復は以下の手順で行う。

1. rank0 の SCT、Coodinator Thread が故障を検知すると、すべての生きているプロセスの AT へ故障を通知する。通知を受けたプロセスは、`MPIFT_GET_DEAD_RANKS()` メソッドを呼び出して障害の状態を取得する。
2. AT は障害情報を得た後、`MPIFT_RECOVER_RANK()` メソッドを呼び出して回復における初期化処理を行う。さらに、故障プロセスに対するメッセージキューをクリアし、故障プロセスとの接続を無効にする。
3. Coodinator Tread は予備プロセスに対して初期化処理を行うように指示する。予備プロセスは初期化処理と、各プロセスとの接続をセットアップする。
4. 予備プロセス以外の各プロセスは、予備プロセスからの接続要求を受ける。
5. すべてのプロセスの接続が完了したのち、各プロセスは `MPIFT_CHECKPOINT_RECOVER()` メソッドを呼び出してチェックポインティングによる回復を行う。

Model-IIa も Model-Ia と同様に SPOF である。Model-IIb および Model-IIc は、Model-IIa の rank0 プロセスを故障検知・回復できるようにした実行モデルである。これら 2 つの実行モデルも未実装である。

3 実装・検証

Model-Ia の実装オーバーヘッドと、故障検知・回復における実行速度の検証を行った。マンデルブロー集合を表示するプログラムを、プロセス数 4(Slave 数 3)、Star 型の環境で実行した。以下の 4 つの条件でそれぞれ行った結果を図 1 に示す。

- MPI/FT を実装する前の MPI/Pro(故障は起きない)
- MPI/FT(故障は起きない)
- MPI/FT(プログラムが 50%進行したところで 1 つの Slave プロセスが故障する。プロセスの回復を行わない。)
- MPI/FT(プログラムが 50%進行したところで 1 つの Slave プロセスが故障する。プロセスの回復を行う。)

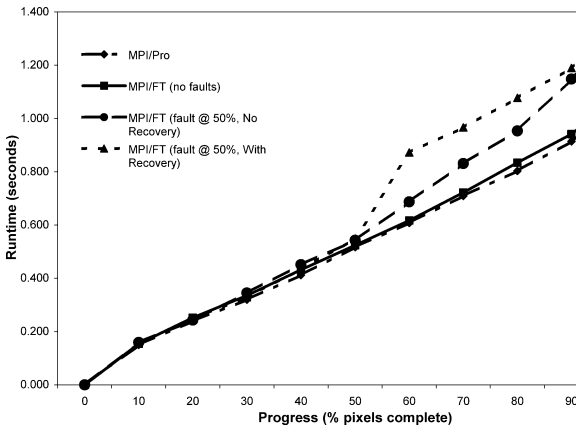

図 1: Model-Ia の実行時間

MPI/Pro(no fault) と MPI/FT(no fault) の結果から、実装におけるオーバーヘッドが小さいことがわかる。また、プログラムの進行度が 50%で故障が発生したとき、プロセス回復を行う場合と行わない場合でほぼ同じ実行時間となっている。追加実験によれば、故障が発生するタイミングを進行度の 30%、10%と早くした場合、プロセス回復を行った方が短い実行時間となり、70%と遅くした場合はプロセス回復を行わない方が短い実行時間となった。このことから、故障による資源損失と故障回復にかかるコストが障害回復する前にわかっている場合、プロセス回復をするかどうかを評価することが好ましいといえる。

次に、Model-IIa の実装におけるオーバーヘッドを検証した (表 2 参照)。実行したプログラムはライフゲームプログラムで、10,000 イタレーション実行した。プロセス数は 4 であり、通信構造は All-to-all 型である。問題サイズを変えながら、実行時間を計測した。MPI/FT の実装を行う前の MPI ミドルウェアである MPI/Pro と、チェックポイントを行わない MPI/FT でそれぞれ計測を行い、実装オーバーヘッドを評価した。

表 2: Model-IIa の実装オーバーヘッド

Grid Size	MPI/Pro	MPI/FT	Overhead
4×4	8.84s	9.26s	4.7%
16×16	8.97s	9.43s	5.2%
100×100	10.63s	11.17s	5.0%
250×250	20.75s	21.25s	2.4%

表 2 より、実装オーバーヘッドは 2〜6%程度の範囲に収まっており、比較的小さい値となっていることがわかる。

4 結論

ユーザが実行モデルを選択して開発を行う、耐故障性 MPI ミドルウェア MPI/FT を提案した。MPI/Pro をベースに、実行モデルの基本となる Model-Ia、Model-Ib を実装した。実装における実行時間オーバーヘッドと、故障回復における実行時間オーバーヘッドを実験により検証し、本稿の手法の有用性を示した。SPOF の問題に対処した実行モデルの実装や、スケーラビリティの問題を解決することが課題である。

参考文献

[1] S.Rao, L.Alvisi, H.M.Vin, “Egida: an extensible toolkit for low-overhead fault-tolerance,” Digest of Papers. 29’th Annual International Symposium of Fault-Tolerant Computing, pp.48–55, 1999

[2] G.Fagg, J.Dongarra, “FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world,” Lecture Notes in Computer Science, Vol.1908/2000, pp.346–353, 2000

[3] IVerari Systems, http://www.verari.com/software_products.asp