

フォールト/リカバリモデルを考慮した耐故障性をもつ MPI フレームワーク ABARIS の提案と評価

實本英之[†] 遠藤敏夫[†] 松岡 聡^{†,††}

大規模クラスタやグリッド等では、故障が発生しやすく長時間にわたる MPI アプリケーション実行のためには耐故障性が不可欠である。故障発生時の復旧方法は実行環境やアプリケーションによって変わるが、既存の耐故障性 MPI は復旧方法の変更にはほとんど非対応であり、対応可能でも、故障に応じた復旧方法をアプリケーションコード中に記述する必要がありユーザ負荷が高い。ABARIS は実行環境やアプリケーションに合わせた復旧方法を用いることが可能な MPI フレームワークである。ユーザは故障検知と復旧方法のコンポーネントを実行環境やアプリケーションに応じて選択することで復旧方法を変更可能となる。プロトタイプとして MPICH-P4MPD に ABARIS を適応し、NPB を用いてオリジナルの MPICH-P4MPD との性能比較を行った。結果、ABARIS を適用することによるオーバーヘッドはほとんど無い(1%以下)であることがわかった。また、故障検知と復旧手法を発生した故障に応じて適切に変えることが、性能に大きな影響を与えることを示した。

ABARIS: An Adaptable Fault Detection/Recovery Component Framework for MPIs

HIDEYUKI JITSUMOTO[†], TOSHIO ENDO[†] and SATOSHI MATSUOKA^{†,††}

Long-running MPI applications on clusters and grids that are prone to node and network failures, motivates the use of fault tolerant MPI implementations. However, previous fault tolerant MPIs lack the ability to allow the user to easily choose appropriate fault recovery strategies according to the execution environment, independent of the application codes. ABARIS is our new Fault/Recovery model aware component framework for MPI, where users can customize MPI fault detection and recovery algorithms according to their application and execution environmental requirements by merely selecting appropriate fault/recovery components, independent of the application code. Currently, the ABARIS framework prototype is implemented on top of MPICH-P4MPD. Preliminary evaluation of the prototype using NPB on our MPI fault simulator demonstrates that overhead compared to the original MPICH-P4MPD is almost negligible (less than 1%) under normal execution, and when faults occur, appropriate selections and pairings of fault model and recovery method components for corresponding to the execution environment is significant to the overall execution time.

1. はじめに

MPI はさまざまな分野において長時間の科学技術計算に頻繁に利用されている。大規模クラスタやグリッドでは故障が発生しやすく長時間にわたる MPI アプリケーション実行のためには耐故障性が不可欠である。加えて理想的な MPI には、様々なユーザが利用できる簡便性、ユーザの実行環境や実行アプリケーションに制限されない可搬性などが必要となる。既存の耐故障性 MPI である LAM/MPI¹⁾ や MPICH-V²⁾ はアプリケーションコードを変更することなく

耐故障性を得ることができるが、故障発生時の復旧方法が画一的に決まっており、余分なコストがかかる。例をあげると、発生した故障が一時的なプロセス故障であるときと、ハードウェア故障に起因し、繰り返し起こる故障であるときには最適な復旧方法は違う。ハードウェア故障に対応するにはプロセスを健全なノードへマイグレーションする必要がある。しかし、常にマイグレーションを行った場合、健全なノードを無駄に消費する可能性がある。

FT-MPI³⁾ のようにプロセスの生成/破棄といった基本的な API の提供によりアプリケーションコードを変更し、柔軟な耐故障性を実装できる MPI も存在するが、故障復旧はアプリケーションユーザが実装する必要があるため利用困難である。

以上を解決するため、環境に合わせた復旧方法を容易に用いることが可能な MPI フレームワーク ABARIS を提案す

[†] 東京工業大学
Tokyo Institute of Technology
^{††} 国立情報学研究所
National Institute of Informatics

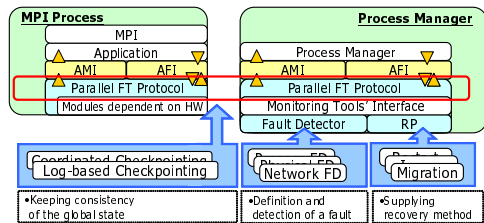


図 1 ABARIS コンポーネント

る．ABARIS は故障モデルの定義と検知を行うコンポーネント (Fault Model component) が適切な復旧処理を行うコンポーネント (Recovery Protocol component) を選択することにより柔軟な故障復旧を提供する．ユーザ，システム管理者，ソフトウェア開発者は，適切なコンポーネントを組み合わせることで，実行環境，アプリケーションに最適な耐故障性 MPI が利用可能になる．

本研究ではプロトタイプとして MPICH-P4MPD に ABARIS を適用し，オリジナルの MPICH-P4MPD との性能比較を行った．結果，ABARIS 適応のオーバーヘッドはほとんど無い (1%以下) ことを確認した．また，故障シミュレータ上で故障の種類と頻度を変更しながら，複数の故障検知と復旧手法の組み合わせを用いた性能比較を行った．結果，故障検知と復旧手法を発生した故障に応じて適切に変えることが性能に大きな影響を与えることを示した．

2. ABARIS フレームワーク

2.1 アーキテクチャ

ABARIS は MPI プロセスの一貫性保障 (PFTP: Parallel Fault Tolerant Protocol component)，故障モデルの定義と検知 (FD: Fault Detector component)，復旧処理 (RP: Recovery Protocol component) からなる主要コンポーネントと，モニタリングツールやチェックポイント等の補助コンポーネントをもつ (図 1)．

また ABARIS はコンポーネント利用のインターフェースとして ABARIS Fault tolerant Interface (AFI) と ABARIS MPID Interface (AMI) を持つ．AFI はベースとなる MPI 実装や ABARIS コンポーネントから，他の ABARIS コンポーネントを利用するために用いる．ABARIS は各ノードにプロセスマネージャ (PM) を持つ MPI システムを想定しており，すべての ABARIS コンポーネントは PM-PM 間，同ランクの PM-MPI プロセス間，MPI プロセス-MPI プロセス間のメッセージをハンドラを AFI として持っている．また，PFTP は一定間隔でイベントを生成するためのハンドラも持つ．一方 AMI は各 ABARIS コンポーネントからベースとなる MPI 実装や PM の機能を利用する際に用いられる．例として，MPI メッセージの送信や受信や MPI ランクの取得，PM 間メッセージの送受信等である．

故障が発生したとき，ABARIS は以下のように復旧を行う

- (1) PFTP は一定間隔で FD にモニタリングツールからの情報と FD コンポーネントの持つ故障モデルの比

表 1 復旧方法の特徴

| Protocol | IGN. | RES. | MIG. | PRO. |
|----------|------|--------|------------|-------|
| 復旧コスト | 0 | medium | large | small |
| 必要ノード数 | 0 | 0 | each fault | large |
| 耐プロセス故障 | No | Yes | Yes | Yes |
| 耐ハード故障 | No | No | Yes | Yes |

較を行わせる．

- (2) 情報が，定義された故障モデルと一致した場合，FD は適切な RP を選択する
- (3) PFTP は選択された RP を用いてプロセスの一貫性を保ちながら復旧処理を開始する

ABARIS コンポーネントは並列コンピューティングの知識が十分にあるプログラマによって実装される．エンドユーザやシステム管理者はすでに実装されたコンポーネントを実行環境やアプリケーションに合わせて選択するだけで故障復旧をカスタマイズすることができる．

2.2 コンポーネントとリカバリモデル

2.2.1 Parallel Fault Tolerant Protocol (PFTP) Component

PFTP は MPI プログラム全体の状態の一貫性をとる．このために，各 MPI プロセスと通信路の状態の一貫性が取れている必要がある．これは，ある状態においてそれぞれプロセスが受信している全てのメッセージに対して，メッセージの送信者が，メッセージを送信したことを確認できる状態にある必要がある．簡単な PFTP の実装として同期チェックポイントングを例に挙げる．同期チェックポイントングは 2 つの動作からなり，まず PFTP は定期的にチェックポイントを作成する．その際，MPI プロセス間の通信路に Drainage バケットを流すことにより in-light なメッセージが無いことを保障する．これにより，すべてのプロセスにおいて受信されたメッセージは送信済みのものであることが保障される．次に PFTP は故障復旧時にすべての MPI プロセスの状態を同時に復旧する．これにより MPI プロセスの状態の一貫性が保障される．

2.2.2 Recovery Protocol (RP) component

RP は MPI プロセス毎をどのように復旧するかを決定する．ABARIS において，RP は以下の 4 種類に分類されている．なお，RESTART と MIGRATION を利用するためにはチェックポイントングが必要であり，PROMOTE を利用するためにはプロセスレプリケーションが必要になる．

IGNORE プロセスは故障を無視する

RESTART プロセスは故障前に実行されていたノードで復旧される

MIGRATE プロセスは故障前に実行されていたノード以外のノードで復旧される

PROMOTE 予備のレプリケーションプロセスを故障プロセスに変わりプライマリにする

表 1 はこれらのプロトコルの特徴である．復旧時コストはプロセスイメージの移動を伴う MIGRATION が大きく，プロセス再生処理の必要ない PROMOTE は小さくなる．

また必要な冗長ノード数はプロセスレプリケーションを行う PROMOTE が大きく、MIGRATION においては障害が起きるたびに冗長ノードが消費される。ユーザは故障の程度にあわせて適切なりカバリプロトコルを選択することにより障害復旧のためのコストを押さえることが可能である。例としては、利用可能なノード数の大きい環境では PROMOTE を使用することや、耐故障性を備えたアプリケーションを利用するときは IGNORE を選択するといった手法がとれる。

2.2.3 Fault Detector (FD) component

FD は故障モデルの定義と検出を行う。FD は定期的にモニタリングツールからの情報と故障モデルの定義を比較し、コンポーネント毎に定められた何らかの閾値を越えたときに前述した 4 種類の RP コンポーネントから 1 種を選択する。FD がモニタリングツールから受け取る情報は、ノードが正常に動作しているかどうか、あるノードで今までに何回の故障が起こったか等がある。以下に、簡単な故障モデルの定義の例をあげる。

Process Fault 故障が MPI プロセスのみで起き、MPI プロセスを実行していたノード自身は利用可能な状態。これはハートビートや Ganglia Cluster Toolkit⁴⁾ などのモニタリングツールを利用することによっても検出できる。

Physical Fault 故障がハードウェアリソースにおいて起こった状態。Process Fault と同様、モニタリングツールを利用することにより簡単に検出可能である。

Network Fault 故障がネットワークリソースにおいて起こった状態。各ノード上でこのモデルを検知するのは難しく、スイッチからの情報とネットワークポロジに関する知識が必要になる。多くのクラスタではスイッチとそれらのトポロジは管理者などに問い合わせることができる。また、グリッドや他の大規模システムにおいてはスイッチトポロジを検出するさまざまな手法が提案されている⁵⁾。

また、これらを段階的に用いることで故障の繰り返しなどの複雑な故障モデルを定義することも可能である。例えば、訂正不能な ECC エラーなどによりプロセスのみが繰り返し故障するとき、これを Process Fault モデルとして検出し、RESTART を適用すると故障が再び繰り返され復旧できない。ABARIS では繰り返しの閾値と故障回数を保持する FD を用い、故障回数が閾値を越えたときに他の FD へ RP の選択を委譲することによって対処可能である。またもし ECC エラーを検出する FD が利用できれば Physical Fault モデルとして直接検出することもできる。

3. プロトタイプ実装

プロトタイプとして、MPICH-P4MPD に ABARIS を導入した。P4MPD はデーモンベースのアーキテクチャであり、ABARIS を導入するのに適している。すべての ABARIS コンポーネントは動的ライブラリによって実装されており、コンポーネントの読み込みは設定ファイルに読み込む動的ラ

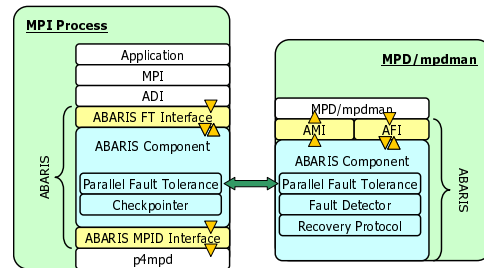


図 2 MPICH-P4MPD 上の ABARIS プロトタイプ実装

イブラリを記述することによって行う。

なお、現在のプロトタイプ実装は、実際に発生した故障に対応することができない。本研究ではシミュレータを用い、仮想的に故障発生と復旧を行う。

3.1 MPICH-P4MPD への ABARIS の適用

MPICH-P4MPD では MPD と呼ばれるデーモンが各ノード毎に実行されている。すべての MPD はリングポロジで接続されており、2.1 章で説明した PM として動作する。MPICH-P4MPD ではジョブの起動は以下のように行われる。

- (1) ユーザが起動した mpirun はローカルに存在する MPD にプログラム起動メッセージを送る
- (2) プログラム起動メッセージはリングを経由して他の MPD に伝播して行き、実行したいプロセスにつき 1 つの mpdman を起動する
- (3) mpdman は隣の rank の mpdman と接続を張り、リング型の通信路をつくる
- (4) mpdman によりユーザプログラムが実行される

図 2 に ABARIS 適用による MPICH-P4MPD の変更を示す。白で示したコンポーネントは MPICH-P4MPD に元から備わっているものである。まず ABARIS コンポーネントの呼び出しを行う AFI を MPI プロセスおよび PM 内に追加した。これにより MPI プロセス、PM 間のすべてのメッセージのハンドルが可能になる。加えて、mpdman に定期的なイベント発生させるように変更を行った。また、ABARIS コンポーネント内では AMI を利用してプロセスのランクの取得や MPICH-P4MPD のデバイス (P4) を用いた MPI 通信などを利用する。

3.2 ABARIS コンポーネントの実装

3.2.1 PFTP component

我々は PFTP として同期チェックポイントを実装した⁶⁾。チェックポイントは設定ファイルに記述された一定間隔毎に、以下の手順で行われる。

- (1) mpdman の 1 つがチェックポイントをスタートするメッセージ (start-cp) をすべての mpdman に送信する。
- (2) 'start-cp' を受け取った mpdman は自分の管理する MPI プロセスに 'start-cp' をリダイレクトする。
- (3) 'start-cp' を受け取った MPI プロセスは実行を中断し接続されている他の MPI プロセスすべてに 'dri-

- nage' メッセージを送信する
- (4) 接続されている MPI プロセスすべてから 'drainage' メッセージを受け取った MPI プロセスはローカルにチェックポイントを作成する。これには ckpt ライブラリ⁷⁾ を利用する
- (5) チェックポイント作成後、MPI プロセスは、全ての MPI プロセスのチェックポインティングを確認するまでメッセージ送信を停止する

3.2.2 RP component

RP として RESTART および MIGRATE プロトコルのプロトタイプを実装した。本研究では故障シミュレータを利用しており、これらのコンポーネントは実際の故障復旧を行わないが、MPI プロセスのサスペンドによって仮想的に故障と故障回復を実現する。サスペンド時間 $T_{recovery}$ は次のように求められる。 $T_{elapsed}$ を最後のチェックポイントから故障発生までの経過時間、 T_{load} はローカルディスクからのチェックポイント読み込み時間、 $T_{transfer}$ をネットワークを利用したチェックポイント転送時間とする。また T_{load} , $T_{transfer}$ は予備実験により計測しておく。このとき、 $T_{recovery}$ は $T_{recovery} = T_{elapsed} + T_{transfer} + T_{load}$ によって求められる。RESTART ではローカルのチェックポイントイメージを利用するために常に $T_{transfer}$ が 0 になる。

3.2.3 FD component

我々は簡単な FD として以下を実装した。これらは 2.2.3 章に示した故障モデルを元にしており、MPI プロセス内で致命的な (仮想的) 故障が発生した場合に実行される。

Process FD ノードの生死をチェックしノードが利用可能 (Process Fault) ならば RESTART プロトコルを選択

Physical FD ノードの生死をチェックしノードが利用不能 (Physical Fault) ならば MIGRATION プロトコルを選択

Network FD ノードが接続されているスイッチの状態を確認する。スイッチにつながっているリソースの利用不能率が閾値を越えると MIGRATE プロトコルを選択

Repeated FD 各ノードの Process Fault の発生回数をチェックし、あらかじめ定められた閾値を越えたときに故障の繰り返しを避けるために MIGRATE プロトコルを選択

スイッチ、ノードの状態は故障シミュレータにより仮想的に定義されて、ランク 0 の mpdman によって集中管理されている。このため、すべての MPI プロセスの故障検出をランク 0 の mpdman で行うことができる。しかし、実際のスイッチ、ノードの故障検出を行う際は、各ノードにリソースの情報が分散しており、ランク 0 による集中的な故障検出ではボトルネックが発生する。このため、将来的には、我々は分散モニタリングツールを利用を検討している。

3.2.4 故障発生器

FD はノード状態をモニタリングツールから取得する。このため我々はモニタリングツールとして故障発生器を実装した。故障発生器は、ノードに割り当てられたプロセスのラン

表 2 評価環境

| | |
|---------|--|
| CPU | AMD Opteron(tm) Processor 242 (1.6GHz)×2 |
| Memory | 2GB |
| Network | GbE×2(1 つのみ使用) |
| HDD | 250GB IDE ATA100 |
| OS/GCC | Linux 2.6.12, GCC 3.3.5 (-03) |

表 3 プロトタイプとオリジナルの MPICH-P4MPD との性能比較

| CG | | | |
|-------|----------|-----------------|------------|
| | Original | prototype impl. | d-ratio(%) |
| AVE | 1569 | 1562 | 0.4745 |
| STDEV | 138.5 | 124.3 | |
| MG | | | |
| | Original | prototype impl. | d-ratio(%) |
| AVE | 6621 | 6616 | 0.06789 |
| STDEV | 232.2 | 142.4 | |
| EP | | | |
| | Original | prototype impl. | d-ratio(%) |
| AVE | 175.4 | 175.0 | 0.2770 |
| STDEV | 0.1611 | 0.4793 | |

ク、ノードの物理故障率、プロセス故障率を仮想ノードの状態として保持している。MPI アプリケーション実行時に、これらの仮想ノードがプロセスに割り当てられ、それぞれの故障発生率に応じてランダムに故障を発生させる。MIGRATE プロトコルの実行は、プロセスに割り当てられた仮想ノードの状態を変更することによって行われる。

4. 性能評価

4.1 評価環境

評価は 256 ノードクラスタのうち 32 台を用いて行い、各ノードには MPI プロセスを 1 つずつ配置した。各ノードの構成を表 2 に示す。ノードは 20 ノードごとに同じスイッチ (Dell Power connect 5224) に接続されており、スイッチ間は 4 ポートのトランキングで接続されている。

すべての実験は 3.2.2, 3.2.4 章で示した故障シミュレータを用いて行った。シミュレータ上で故障発生イベントが生成されると、MPI プロセスはサスペンドを用いて一定期間仮想的な故障状態となるよう実装されている。また、十分高速なネットワークに接続され、ストレスなくチェックポイントを保存可能な理想的なチェックポイントサーバを仮定している。各 MPI プロセスのチェックポイントはまずローカルドライブに保存され、すべてのチェックポイントが作成された時点で、サーバによって回収されるとする。

4.2 ABARIS 適用オーバーヘッド

ABARIS 適用によるオーバーヘッドを測定するために、Nas Parallel Benchmark 2.4 CG/MG/EP Class-C によるプロトタイプとオリジナルの MPICH-P4MPD の性能比較を行った。この際、プロトタイプは一切の ABARIS コンポーネントを利用せず、AFI

頻度と各 FD セットの性能比較である。いずれにおいても明らかに PROC-PHY-SW セットの性能がよく、性能差は 18%から 35%であった。これより必要があれば、健全なノードの利用を諦めるといった手法も環境によっては効果的であることを確認した。つまり、あらかじめ故障の発生を予測し、故障発生ノードを避けることが性能に大きな影響を与えられらる。ABARIS は故障発生の予測と予測に応じた故障復旧を柔軟に定義することが可能であり、耐故障機能によるオーバーヘッドを効果的に抑えられる。

5. 関連研究

LAM/MPI¹⁾ は Chandy-Lamport アルゴリズム⁸⁾ による同期チェックポイントを実装した耐故障 MPI である。しかし、LAM/MPI は他の並列耐故障アルゴリズム (本研究での PFTP) をサポートしていない。加えて故障検知を行わないため、自律的な復旧は行わない。

Egida⁹⁾ は自律的な故障復旧を目的とした耐故障 MPI で耐故障アルゴリズムとして 3 種類のログベースチェックポイントを利用可能である。

MPICH-V²⁾ はより広範囲な並列耐故障アルゴリズムが利用可能で、並列耐故障アルゴリズムの性能比較を行うことを主眼にした研究である。この比較の結果、同期チェックポイントが故障頻度にかかわらずよい性能を持つことを示している。本研究との差異として、ABARIS は複数の耐故障アルゴリズムのみでなく、複数の故障モデル定義と検知、復旧方法の組み合わせによる柔軟な故障復旧を提供していることが挙げられる。

FT-MPI³⁾ は、MPI のコミュニケータレベルで故障に対処する。故障が発生すると、MPI 命令がエラーを返すように実装されている。これにより、FT-MPI は柔軟な故障復旧を実現できるが、アプリケーションコードに耐故障機能を実装する必要があり、ユーザ透過ではない。

6. おわりに

本研究では故障 / 復旧モデルを考慮した耐故障性を持つ MPI フレームワークである ABARIS を提案した。また、プロトタイプとして MPICH に ABARIS を適用し、ABARIS 適用によるオーバーヘッドを測定した。結果、オーバーヘッドは無視できるほど少ないことを確認した。次に、2 つの実行環境モデルにおいて故障シミュレータを用い、ABARIS フレームワークの正当性を検証した。結果、実行環境毎に適した故障 / 復旧モデルの利用が耐故障機能によるオーバーヘッドを抑えることを確認した。今後の課題としては ABARIS コンポーネントの選択やコンポーネントパラメータの自律構成を検討している。また、ABARIS

を MPICH-2 の仮想通信チャンネルレイヤに実装することも検討しており、これにより、myrinet や Infiniband といったデバイスに依存しない耐故障 MPI を実現できる。

謝辞 本研究の一部は平成 18 年度科研費特定領域研究「情報爆発に対応する高度にスケーラブルな高性能自律構成実行基盤」による

参考文献

- 1) Squyres, J. M. and Lumsdaine, A.: A Component Architecture for LAM/MPI, *Proceedings, 10th European PVM/MPI Users' Group Meeting*, Lecture Notes in Computer Science, No. 2840, Venice, Italy, Springer-Verlag, pp. 379-387 (2003).
- 2) Bouteiller, A., Herault, T., Krawezik, G., Lemarinier, P. and Cappello, F.: MPICH-V: a Multiprotocol Fault Tolerant MPI, *International Journal of High Performance Computing and Applications*. (2005).
- 3) Fagg, G. and a Dongarra: FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World, *Euro PVM/MPI User's Group Meeting 2000*, Springer-Verlag, Berlin, Germany, pp. 346-353 (2000).
- 4) Massie, M. L., Chun, B. N. and Culler, D. E.: The Ganglia Distributed Monitoring System: Design, Implementation and Experience, *Parallel Computing*, 30, (2004).
- 5) Lowekamp, B., O'Hallaron, D. and Gross, T.: Topology discovery for large ethernet networks., *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*, pp. 237-248 (2001).
- 6) Elnozahy, E., Johnson, D. and Wang, Y.: A survey of rollback-recovery protocols in message-passing systems, *ACM Computing Surveys*, Vol. 34, No. 3, pp. 375-408 (2002).
- 7) Zandy, V. C.: ckpt: A process checkpoint library (2002). <http://www.cs.wisc.edu/~zandy/ckpt>.
- 8) Chandy, K. M. and L.Lamport: Distributed snapshots : Determining global states of distributed systems., *Transactions on Computer Systems*, vol. 3(1). ACM, pp. 63-75 (1985).
- 9) Rao, S., Alvisi, L. and Vin, H. M.: Egida: An Extensible Toolkit For Low Overhead Fault Tolerance, *In Proceedings of the 29th Fault-tolerant Computing Symposium (FTCS-29)*, Madison, Wisconsin, pp. 48-55 (1999).