

ポータブルな耐故障性コンポーネントフレームワークを持つ MPI 実装に向けて

實 本 英 之[†] 松 岡 聡^{†,††}

クラスタの信頼性の低さから、並列、広域化に広く用いられる MPI においても、耐故障性を備えたものが作られた。しかしそれらは環境ごとに変わる Fault/Recovery Model を考慮した物が少なく、考慮した物も、状況に応じた復旧手段をコード中に記述する必要がありユーザーの負担が多い。本研究では環境に合わせた Fault/Recovery Model を容易に用いることが可能な MPI を提案する。Fault/Recovery Model をコンポーネントとして提供することにより、ユーザーはコンポーネントを選択するだけで、環境に最適な耐故障性 MPI を用いることが可能となる。ナイーブな MPI である MPICH に Fault/Recovery Model Aware な耐故障性を実装し、東工大松岡研 PrestoIII クラスタ上において NPB-CG によるベンチマークを行った。結果、オリジナルの MPICH に比べ最大 10% 程度の性能向上がみられた。

Towards a Portable Fault Tolerant Component Framework for MPI

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

Execution of MPI applications on clusters and Grid deployments suffering from node and network failures motivates the use of fault tolerant MPI implementations. Therefore, some fault tolerant MPI was implemented. But, these fault tolerant MPI implementations cannot choose easily appropriate restoration according to the environment. We present CuckooMPI, used Fault/Recovery model aware component framework. Users can get a MPI implementation according to their executing environment by the selection of the components. This paper presents the architecture of CuckooMPI, its theoretical foundation and the performance of the implementation. Preliminary evaluation using NPB-CG with 32 processes showed, CuckooMPI has 10% performance improvement compared with MPICH.

1. はじめに

コモディティクラスタリングシステムやグリッドシステムの発展に伴い、現在多くの科学技術計算が並列、広域化されている。これらのシステムは大きな問題として信頼性の低さを抱えている。しかしながら科学技術計算は長時間にわたる事が多く、システム全体の信頼性は大きく下がっている。

これをうけ、並列、広域化に広く使われる MPI¹⁾ においても、耐故障性を備えたものが実用化されはじめています。LAM/MPI²⁾ をはじめとする耐故障性 MPI はすべてのフォールトに対し、常に同じリカバリモデルを適用するが、本来、フォールトの種類や、それに対する適切なリカバリモデルは環境毎に変わる。このため余計なコストがかかってしまう。これに対し、リカバリモデルを考慮した FT-MPI³⁾ など存在するが、これは環境に合わせた復旧手段をアプリケーション

のユーザが記述する必要がある。これにはアプリケーションに関する詳細な知識が必要でありユーザの負担が大きい。

環境に合わせたフォールト/リカバリモデルを容易に用いることが可能な MPI である CuckooMPI を提案する。CuckooMPI は以下の要件を満たすよう設計した。

- ・環境に合わせ柔軟なフォールト/リカバリモデルを使用可能
- ・特定の OS やハードウェアに非依存
- ・ユーザによるアプリケーションプログラムの変更が不必要

CuckooMPI はフォールト/リカバリモデル中心のコンポーネントモデルにより、これを実現した。コンポーネントは、動的ライブラリで提供されており、ユーザ、システム管理者、ソフトウェア開発者が、適切なコンポーネントを組み合わせることで、環境およびソフトウェアアーキテクチャに最適な耐故障性 MPI を用いることが可能になる。

以下、2 節では現状の耐故障性 MPI の問題について述べる、3 節において本研究の提案する CuckooMPI のアーキテクチャについて述べ、4,5 節ではプロトタ

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

イプの実装と性能について述べる．最後に 6,7 節において関連研究を紹介しまとめを行う．

2. MPI の耐故障化

ここでは、理想的な MPI に求められる要件と、MPI の耐故障化によって起こる問題について述べる．

2.1 理想的な MPI

MPI は並列プログラムで使われる汎用通信インターフェースであり、様々な環境に適した実装を持っている．さらに、物理、化学、地学といった様々な分野のユーザが使用しており、長時間の科学技術計算にも利用されている．これより、理想的な MPI は以下のような機能を備えている必要がある．

異常終了を起こさない MPI が用いられる科学技術計算は長時間にわたる．しかし、実行環境となるクラスタやグリッドは様々な要因でノードの信頼性が低い．よって障害復旧の能力を持つべきである．簡単に利用できる MPI は計算機科学以外の分野でも研究のツールとして利用される．このため、手軽にユーザの必要とする並列アプリケーションプログラムを記述できる必要がある．

環境に依存しない MPI は様々なオペレーティングシステムやハードウェアの上で実行される．また、様々なソフトウェアアーキテクチャを実装可能である．このため、特定の環境やソフトウェアアーキテクチャに依存しないよう実装する必要がある．

2.2 耐故障性 MPI の問題

初期の、MPICH¹⁾、LAM⁴⁾などの MPI 実装は、MPI インターフェースを用いて様々な環境で簡単に利用可能だが、耐故障性に関しては実装されていない．近年のクラスタやグリッドなどの普及により、耐故障性が必須となった結果、LAM/MPI²⁾や FT-MPI³⁾といった耐故障性 MPI が実用化され始めている．しかしながら、これには以下のような問題がある．

エラーに対するリカバリモデルが使いづらい 多くの耐故障性 MPI は単一の復旧方法しか持たないため、ソフトウェアアーキテクチャや実行環境によって必要のないリカバリコストがかさんでしまう．また、ユーザがアプリケーションプログラムを変更して耐故障部分を記述できるものも存在するが、アプリケーションプログラムの構造を熟知している必要があり、計算機の知識を持たないユーザが使用することも考えると負荷が大きい．

特定の OS/HW への依存性が高い 耐故障性 MPI はその実現に、チェックポイントや高信頼ネットワークなどを使用している．これの実装は特定の OS やハードウェアに依存してしまうことが多く MPI に必要とされる多くの環境へ非依存であることを損なう．

以上を考えると理想的な MPI 実装を行うためには

以下の要件を満たす必要がある．

- リカバリモデルを容易に変更可能である
- 耐故障機能を使用するためにユーザに負担を与えない
- 耐故障機能が OS/HW 依存にならない

3. CuckooMPI

本研究で提案する CuckooMPI はポータブルな耐故障性コンポーネントフレームワークを持つ MPI である．CuckooMPI はフォールト/リカバリモデルおよび、環境依存性の高い耐故障機能の部分をコンポーネントとして提供する．MPI ユーザは容易されたコンポーネントを選択することにより、アプリケーションプログラムを変更する必要なく、柔軟に実行環境に適応した耐故障性 MPI を使用することが可能になる．

3.1 リカバリプロトコル

プロセスの障害にたいして耐故障性を実現するに当たり、頻繁に用いられる技術として以下の二つが存在する．

チェックポインティング/リスタート・マイグレーション

プロセスのイメージを定期的にディスクに保存（チェックポインティング）．障害発生時は、利用可能なノードにおいてイメージの復元を行う（マイグレーション、リスタート）．

プロセスレプリケーション 計算を行うプロセスと同様のイメージを持つ冗長なプロセスを用意しておく．プライマリプロセスに送られたメッセージ等はレプリケーションプロセスにも送信される．障害発生時はレプリケーションプロセスの 1 つをプライマリプロセスにする

MPI 構成プロセスを復旧させる手段として CuckooMPI はこの 2 つのアルゴリズムを仮定している．具体的に CuckooMPI は以下のプロトコルを障害プロセス毎に決定する．

IGNORE 障害の発生したプロセスの復旧は行わない．コストは一切かからないがプロセスは失われる．

RESTART 障害の発生したプロセスを元々動いていたノードで復旧する．正常実行時にはチェックポイントを取っておくコストが必要でその大きさはチェックポインティングのインターバルによる．復旧時は、チェックポイントから復元する動作が必要となる．ノードに障害が発生した場合には復旧ができない．

MIGRATION 障害の発生したプロセスを元々動いていたノード以外のノードで復旧する．複数のノードからアクセス可能な場所にチェックポイントを取っておく必要があり、正常実行時のコストは RESTART に比べ大きなものになる．復旧時のコストも RESTART に加え、チェックポイン

表 1 リカバリプロトコルの性質

プロトコル	IGN.	RES.	MIG.	TRA.
実行時コスト	無し	ckpt. 頻度	ckpt. 頻度	通信頻度
復旧時コスト	無し	中	大	小
耐プロセス障害	無し	有り	有り	有り
耐ノード障害	無し	無し	有り	有り
冗長ノード数	無し	無し	障害頻度	大

トのノード間転送分コストが必要になる．ノードに障害が発生した場合も復旧可能であるが、すでに他のプロセスが割り振られているノードにマイグレーションすると負荷が増す．そのため、復旧には冗長ノードが必要となる．

TRANSFER 障害の発生したプロセスのレプリケーションプロセスをプライマリにする．レプリケーションプロセスを作っておく必要がある．正常実行時のコストはメッセージを複数送信することによるもので通信頻度に応じて大きくなる．障害復旧はプライマリプロセスとレプリケーションプロセスが同等であるため、ほとんどコストがかからない．ノードに障害が発生した場合でも復旧可能である．しかし、レプリケーションプロセスをプライマリプロセスと同一ノードで行うと、プライマリプロセスの負荷となるため、冗長ノードは復旧時だけでなく常に必要である．

図 1 はリカバリプロトコルの特徴をまとめたものである．故障の程度にあわせて適切なリ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．しかし、これらのリカバリプロトコルを適切に選択するには MPI のフォールトを適切に分類する必要がある．

3.2 フォールトモデル

前節で述べた 4 プロトコルにおいて、復旧できる障害の種類を考慮した上で MPI のフォールトを以下の 3 モデルに分類する．

Network Fault 冗長化ネットワークの一部の障害、あるいはパケットロス等によるネットワーク性能の著しい低下状態．ただしネットワークが完全に使用不能になってしまった場合は後述の Physical Fault として扱う．Network Fault では完全に計算が中断するわけではないため、IGNORE を選択することも可能である．計算を続行するためには MIGRATION/TRANSFER により他のノードにプロセスを移す必要がある．

Process Fault 並列ジョブを構成するプロセスが何らかの原因により異常終了してしまった状態．計算が中断してしまうために耐プロセス障害の能力を持つ RESTART/MIGRATION/TRANSFER を用いる必要がある．

Physical Fault ハードウェアの障害により、物理的にノードが使えない状態．また、ネットワーク

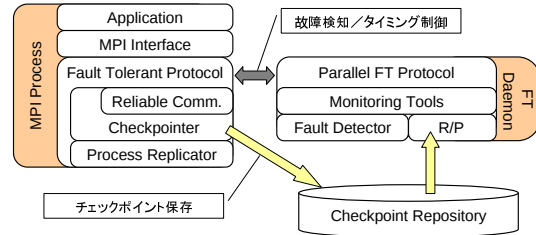


図 1 CuckooMPI コンポーネント図

が完全に切断されノードが応答不能になった場合も Physical Fault として扱う．復旧には耐ノード障害の能力を持つ MIGRATION を用いる必要がある．

また、MPI 構成プロセスだけでなく、耐故障機能に障害が起こることも考えられる．これに対処するために耐故障機能障害を以下の 4 モデルに分類する．

Migration/Transfer Fault MIGRATION および TRANSFER の結果、冗長要素がなくなった状態．以後 MIGRATION/TRANSFER が選択された場合は、計算を中断し新たな冗長要素の追加を待つか、すでに他のプロセスが実行されているノードに MIGRATION/TRANSFER を行う必要がある．後者の場合、負荷分散の問題から並列ジョブの性能は著しく低下する．

FTSystem Fault 何らかの理由により通常の計算に支障はないが、耐故障機能が使用不能になる状態．例としては、チェックポイントのタイミングを発行するデーモンの障害や、チェックポイントレポジトリの障害、障害発生を監視するモニターの障害などがあげられる．復旧手段は故障したコンポーネントにより様々で、復旧するまで耐故障性機能は使えない．

Double Fault 故障復旧中にさらに故障が発生した状態．最も簡単な対処法としては新たに検出されたフォールトモデルにしたがって故障復旧の作業を最初からやり直すことである．

CuckooMPI は以上の 6 つのフォールトモデルに対してリカバリプロトコルを選択するコンポーネントを組み合わせることで、実行環境、ソフトウェアアーキテクチャに柔軟なリカバリモデルを提供する．

3.3 コンポーネント

図 1 は、CuckooMPI のコンポーネントの全体図である．CuckooMPI は、実際に計算を行う MPI 構成プロセスと、耐故障機能を制御するデーモンからなる．各コンポーネントは以下のような機能を持つ．

Application ユーザが用意する実際のアプリケーションプログラム

MPI Interface MPI を単純な I/O システムコールで実現する部分

Fault Tolerant Protocol/Parallel FT Protocol

並列ジョブの耐故障機能を逐次プロセスの耐故障機能で実現するためのタイミング調整などを行う。実際にはチェックポイントングを取るときの同期やリスタートタイミングの指定、レプリケーションプロセスのプライマリへの移行タイミングの指定を行う。

Reliable Comm./Checkpointor/Replicator

これらは逐次プロセスの耐故障化に必要な機能で、既存研究として存在する、高信頼ネットワーク、チェックポイント、プロセスレプリケータを流用する。これらのコンポーネントを OS やハードウェア毎に入れ替えることにより、実行環境に柔軟な耐故障性 MPI を実現可能である。

Monitoring Tools CuckooMPI における故障を検知するためのコンポーネント。Monitoring Tools は検知した故障を前述した 6 つのフォールトモデルに分類し、適切な Fault Detector に通知する

Fault Detector/Recovery Protocol Fault Detector は Monitoring Tools が判別したフォールトに対してさらに詳細な分析を行い、リカバリプロトコルを決定する。Recovery Protocol は Fault Detector が決定した方法に従い実際にリカバリを行う。この部分を実行環境やソフトウェアアーキテクチャ毎に入れ替えることにより、柔軟なりカバリモデルを持った耐故障性 MPI を実現可能になる。

4. 実 装

プロトタイプとして、一般の研究室などにあるローエンドなコモディティクラスタを対象としたコンポーネントを実装した。実装の概要を以下に述べる。

MPI Interface ナイブな MPI 実装である MPICH-P4 を改造して用いた。変更点はまず、ジョブ実行時に Fault Tolerant Daemon からジョブを一意に判別できるよう mpirun においてユニークな ID (MPIID) を取得できるようにした。また、P4 ライブラリによってスレーブプロセスが起動される際にも、MPIID を伝えるよう改造した。

さらに、MPICH-P4 のチャンネルである P4 ライブラリは rank 0 のプロセスとの接続は MPI_Init 時に作成すが、ほかの接続に関しては以下のように通信を確立する。

- (1) 計算プロセスは、相手のプロセスとの通信のためのソケットを開く。
- (2) 計算プロセスは、P4 のプロセスステابلを確認し、相手のプロセスのリスナへ向けて自分のポート番号とともに通信要求を送信する。
- (3) 通信要求を出した計算プロセスは、相手が

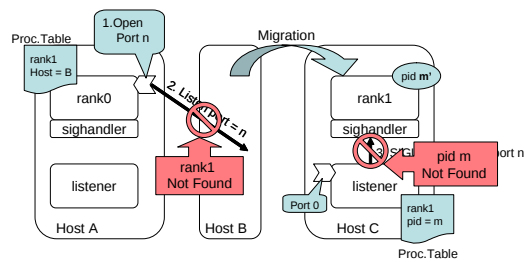


図 2 マイグレーション後のコネクション

らのコネクションを持つ。

- (4) 通信を要求された側のリスナは、計算プロセスに向けて SIGUSR1 を送信する。
- (5) 計算プロセスはシグナルハンドラへ入る。リスナは通信要求を計算プロセスへ渡し、計算プロセスは要求元の計算プロセスとコネクションを張る。

しかしチェックポイントング/リスタートを行うと、プロセス ID が変更され SIGUSR1 が届かなくなる。また、通信路が張られる前にプロセスマイグレーションが行われると、プロセスステابلのホスト名から変わってしまうため、新しい接続が作れなくなる (図 2)。プロセスステابلの変更は難しいため、MPI_Init() 終了時に全対全通信を行い通信路を全て作成してしまう実装を行った。

Fault Tolerant Protocol/Parallel FT Protocol

Coordinated Checkpointing のみ実装を行った。これは同期によりチェックポイント間の通信の一貫性を取る方法で、現在の実装では全てのプロセスがチェックポイントを終了するまでプロセスの実行を停止する

Reliable Comm./Checkpointor/Replicator

Reliable Communication として Rocks^{5),6)} を用いた。これは通信バッファをコピーしておき、ネットワーク復旧後に足りないデータを再送する仕組みを持つ。また Checkpointor としては ckpt⁷⁾ を用いた。Replicator は実装していない。

Monitoring Tools

Fault Tolerant Daemon を 2 つに分け、システム毎に存在するものと各ノード毎に存在する Sub FT Daemon とした。ノード間においては HB により生存をチェックし、プロセスは kill -0 シグナルを用いて生存をチェックする。(図 3)

Fault Detector/Recovery Protocol

ローエンドコモディティクラスタでは設置環境の不備からハードウェアの故障が起きやすい。また、ネットワークの冗長化も想定できない。さらに冗長ノードが豊富に使用できるわけではないのでレプリケーション方式は向いていない。以上を考え以下の実装を行った。

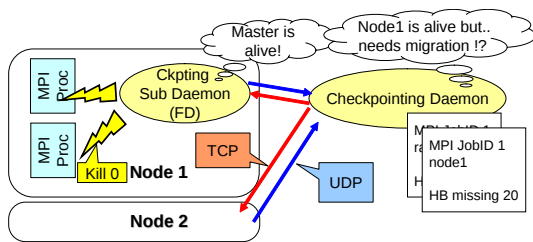


図 3 モニタリング

表 2 実験環境

CPU	Athlon 1900+×2
Memory	768MB
Network	100base-T/1000base-T
OS	Linux 2.4.18

Network Fault MIGRATION を選択

Process Fault RESTART, 同じノードで何度も繰り返されるならば MIGRATION を選択

Physical Fault MIGRATION を選択. MIGRATION が無効である場合は IGNORE を選択し, 異常終了させる.

Migration Fault MIGRATION を無効化. ノードが不安定なため過負荷により他のプロセスを巻き込んで故障が起こる可能性があるため.

5. 実験と考察

実装したプロトタイプの実験評価を行う.

オリジナルの MPICH と比較すると実行時は高信頼通信路である Rocks のオーバーヘッドがかかる. Rocks 使用によるオーバーヘッドは, 通信バッファのコピーである. このオーバーヘッドは通信回数 N に対して $O(N)$ であるため, Rocks の影響はプロセス数の増加に対して一定の割合であると考えられる.

5.1 実験環境

実験環境として, 東京工業大学松岡研究室の PrestoIII クラスタを用いた. 各ノードは表 2 の構成である.

5.2 通常実行時ベンチマーク

Cuckoo MPI が実用的な性能で実行されるか確認するために, NPB-CG CLASS A⁸⁾ を用いたベンチマークを行った. 条件は以下になっている.

- 問題サイズ CLASS A
- プロセス数は 2 のべき乗個で, 2 から 32 プロセス
- 通信 1000Base-T
- 実行中の意図的な障害発生, 復旧は行わない.

結果 (図 4), 最大で 10% 程度の性能向上を示している. これは, プログラム前に全対全の接続を確立してしまうために, 実行時の接続手続きが必要なく, 計算の外乱が少なくなるために性能が向上したと考えられ

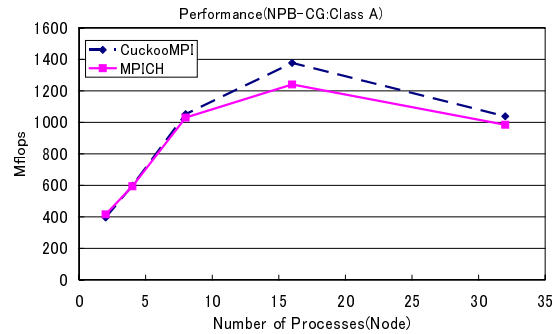


図 4 通常実行時ベンチマーク

る. 2 プロセスの時は MPICH においても MPIInit 時にソケットが接続されるために MPICH の方が高い性能を示している. より通信サイズの大きな CLASS B などを用いるとコピー時間のオーバーヘッドがこの性能向上を上回り性能低下が起きると予想できる. しかし, Rocks の実装は, 送信バッファがあふれるとバッファに書き込めるようになるまで write を再実行し続けるようになっている. 相手が非同期通信を仮定して同様に大規模なデータを送信すると, 双方が write を繰り返し処理が止まってしまう. 現状この部分が解決できていないため実験を行うことができなかった.

5.3 フォールト/リカバリモデルの適用実験

フォールト/リカバリモデル中心のフレームワーク作成における優位性示すための実験の指針を示す.

故障発生器を用いてネットワークにパケットロスを実ミュレーションする. さらに, ネットワークがつながっている限り実行を続けるもの (従来の耐故障性 MPI) とパケットロスが多いときにマイグレーションを行う (フォールト/リカバリモデル適用) 2 つの Network Fault Detector を用意する. これによりパケットロスが頻繁に起こる環境 (HW 安定性が低い) では, 後者の方が性能が向上すると予測でき, 環境に合わせた Recovery Model が必要であると実証できる.

6. 関連研究

6.1 LAM/MPI

LAM/MPI は MPI 実装の一つとして LAM⁴⁾ を用いたものである. LAM/MPI は SSI (System Service Interface) と呼ばれる 4 種のコンポーネントをもち, 以下のような機能を持っている.

Boot リモートノードへの起動手段の提供

RPI MPI の一対一通信の提供

Coll MPI の集団通信アルゴリズムの提供

CR チェックポインティング/リスタートを提供

耐故障性についてはコーディネイテッドチェックポインティングが実装されているが, 専用の CR SSI, RPI SSI を使う必要があり, 現状では RPI は TCP

のみの実装である。CR には Berkeley Lab Checkpoint/Restart (BLCR)⁹⁾ というカーネルレベルチェックポイントを用いており、マイグレーションができない。また、故障検知機構も持っていない。しかしながら、これは今後複数の CR コンポーネントが開発されるにつれ改善される可能性がある。耐故障機能を使用するにあたりアプリケーションプログラムをかきかえる必要はない。

6.2 FT-MPI

FT-MPI³⁾ は他の多くの耐故障性 MPI と違い、ユーザ透過な耐故障性を提供しない。ユーザが MPI プログラムに直接、故障への対処を記述することにより耐故障性を発揮する。

通常の MPI 規格では、MPI ジョブ中のひとつのプロセスがダウンしたり、通信に失敗が起これば、MPI ジョブ全体がクラッシュする。FT-MPI はコミュニケータおよびプロセスの状態拡張しており、コミュニケータのエラーの検知、および障害への柔軟な対処方法をユーザに提供する。

しかしながらプログラマがエラー処理に関するコードを追加しなければならず、非常に負荷が高い。

7. まとめと今後の課題

本研究では、フォールト/リカバリモデル中心のコンポーネントフレームワークをもった耐故障性 MPI、CuckooMPI を提案した。CuckooMPI は以下の機能を備えている。

柔軟性 様々な MPI のエラーに対し柔軟な対応が可能であること

透過性 耐故障機能を導入したことにより MPI ユーザに負荷をかけない

可搬性 MPI が実行される多くの環境で使用可能なシステムである

さらに、CuckooMPI のプロトタイプ実装を行い、そのオーバーヘッドを測定したところ、最大で 10% 程度の性能向上が見られた。これは実行開始時にプロセス間に全対全のコネクションを作成してしまうことが原因であると考えられる。また、フォールト/リカバリモデル中心のコンポーネントフレームワークの優位性を示すための実験の指針について述べた。今後の課題としては以下のものがあげられる。

フォールト/リカバリモデル中心であることの優位性
前節最後に示した実験の指針に従い、フォールト/リカバリモデル中心であることの優位性を示す。
コンポーネントの追加実装 現状は、それぞれのコンポーネントにおいて 1 種の実装しかできていない。今後 Uncoordinated Checkpointing のためのコンポーネントや複数のモニタリング手法、またローエンドコモディティクラスタ以外への適応のための様々な Fault Detector/Recovery Protocol

を実装していく必要がある。

謝 辞

本研究の一部は科学技術振興事業団・戦略的創造研究「低電力化とモデリング技術によるメガスケールコンピューティング」による。

参 考 文 献

- 1) Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: High-performance, portable implementation of the MPI Message Passing Interface Standard, *Parallel Computing*, Vol. 22, No. 6, pp. 789–828 (1996).
- 2) 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).
- 3) Fagg, G. and a Dongarra: FT-MPI: FaultTolerant MPI, Supporting Dynamic Applications in a Dynamic World (2000). Euro PVM/MPI User's Group Meeting 2000, Springer-Verlag, Berlin, Germany, 2000, pp. 346–353.
- 4) Burns, G., Daoud, R. and Vaigl, J.: LAM: An Open Cluster Environment for MPI, *Proceedings of Supercomputing Symposium*, pp. 379–386 (1994).
- 5) Zandy, V. C. and Miller, B. P.: Reliable Sockets, Technical report, University of Wisconsin-Madison Computer Sciences (2001).
- 6) Zandy, V. C. and Miller, B. P.: Reliable Network Connections, *Proc. of the 8th Mobicom*, pp. 95–106 (2002).
- 7) Zandy, V. C.: ckpt: A process checkpoint library (2002). <http://www.cs.wisc.edu/~zandy/ckpt>.
- 8) Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, D., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V. and Weeratunga, S. K.: The NAS Parallel Benchmarks, *The International Journal of Supercomputer Applications*, Vol. 5, No. 3, pp. 63–73 (1991).
- 9) Duell, J., H. P. and Roman, E.: The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart., Technical report, Berkeley Lab Technical Report (publication LBNL-54941) (2003).