クラスタ監視機能付きMPI通信ライブラリ

酒　井　将　人[†]　石　川　　　裕[†]

MPI 通信ライブラリに障害の場所を特定する監視機能を持つシステムを提案する．従来の障害検知システムで用いられる管理サーバやハートビートメッセージを必要としないという特徴をもつ．これによって，余分なコストをかけず，障害が発生していない状態ではパフォーマンスの低下がないシステムを目指している．また，障害に関する事柄をまとめ，障害を抽象化し，将来の拡張を容易にすることを目的とした障害モデルを提案する．OS から報告されたエラーをもとに可能性がある障害を得て，障害モデルで設定されている方法を用いて障害が特定される．

MPI library with cluster monitoring system

Masato Sakai[†] and Yutaka Ishikawa[†]

We propose a new MPI library with cluster monitoring functions that detect locations of faults. The system dose not require the management server and the heart beat message used in a traditional fault detection system. So, under the normal operation, the system dose not decrease the performance with no additional cost. We also propose a fault model in order to abstract faults and to be capable of fault extentions. A fault is detected by obtaining candidates for fault based on an error report from OS, and using the method that the fault model specifies.

1. はじめに

大規模な PC クラスタシステムにおいて，システム全体での障害発生率は，単体の PC に比べて非常に高くなる．MPI を用いた並列処理の代表的なアプリケーションである大規模な科学技術計算では，実行時間が数日から数ヵ月にも及ぶことがある．そのため，計算中のノードのどこかで障害が発生した場合は，アプリケーションを再実行する必要がある．このような，障害が発生した時にアプリケーションを最初から再実行する必要がある事態へ対処する研究は数多く存在する．

MPI を用いたアプリケーションに耐障害性をもたせる研究の多くは，障害が発生してから計算を再開するまでにかかる時間を短くすること，耐障害性を得るために追加した動作のオーバヘッドを小さくすることが目標になっている．つまり，障害が発生した後の処理とその処理を行うための準備としての処理の 2 つのことに重点をおいていると言える．しかし，障害の検知について扱っている研究は少なく，障害検知を正しくかつ計算用の処理に影響を与えない形で実行するための研究は行われていない．もし，検知が正確に行われれば，適切なリカバリプロトコルの選択や，何が原因で実行が止まってしまったのかがわかる詳細なエラーを出力して終了することが可能である．

本稿では，上記の主張のうち，障害の検知を正しくかつ迅速に，さらに MPI のアプリケーションの性能を下げること無く実行できるシステムを提案する．提案するシステムは，障害が発生していないときには，MPI の通信性能に影響を与えないことを目標にしている．障害が発生したときの計算性能や復旧までにかかる時間は考えないものとする．

2. 障害検知

2.1 障害の種類

ハードウェアに起きる障害とその障害を検知することができる規格については**表 1** の通りである．IPMI[8)] は，特定のハードウェアや OS に依存することなくハードウェアを監視することを可能にした標準のインターフェース仕様である．一般的には管理ソフトウェアの下位インターフェースとして機能している．ノード内部の情報だけでなく，ネットワーク経由で監視する方法も用意されている．SNMP[9)] は，UDP を用いてネットワーク経由で機器を監視するためのプロトコルで，RFC1157 で定義されている．リモートからルータやスイッチの状態，ネットワークのトラフィックの状態を取得することができる．ノードに関する情報は IPMI によって，ノード間のネットワーク部分の情報は SNMP によって取得することが可能であり，これ

† 東京大学
The University of Tokyo

表 1 障害と検知の規格

障害の場所	検知できる規格	収集できるデータ
CPU	IPMI	内部エラー，プロセッサの速度が自動的に抑制された，訂正不可能な CPU エラー，温度
CPU Fan	IPMI	回転数
メモリ	IPMI	訂正不可能な ECC エラー
ディスク	IPMI	回復不可能なデバイス障害，回復不可能なデバイスコントローラの障害
電源	IPMI	電源装置の障害，電力が供給されていない
NIC	IPMI	障害を検知した，電源が供給されていない
スイッチ	SNMP	電源，ファン，ポートの正常/異常．ポートごとの送受信データサイズ．ポートごとのケーブル着脱状態.

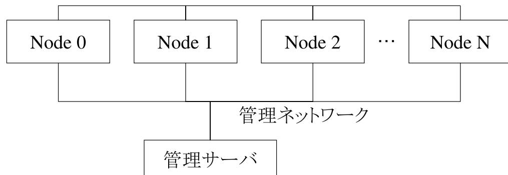

図 1 障害検知システムその 1

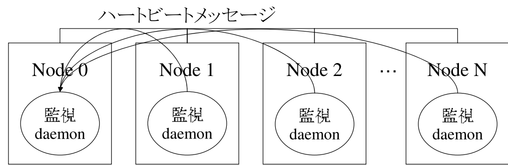

図 2 障害検知システムその 2

らを用いたツールも存在している．

ソフトウェアに関する障害としては，RSH や NFS 等の MPI アプリケーションを実行するために必要なサービスが正常に利用できない時があげられる．実際にどのサービスが必要であるかということについては，MPI ライブラリの実装やクラスタの構成に依存する．

2.2 既存の障害検知システム

クラスタの障害を検知するシステムは大きく分けて 2 通りある．クラスタノードとは別に監視専用サーバを用意して外部から監視する方法とクラスタノード内に監視プロセスを用意して内部から監視する方法とである．

外部から監視する場合の実装は，**図 1** のように，監視サーバからクラスタの各ノードに対して情報を取得するリクエストを送信し，得られた応答を解析して障害を検知する方法が一般的である．この時，管理サーバと各ノード間のネットワークには，管理専用ネットワークが用意される．問題点としては，新しくサーバを用意することにコストがかかるということや管理サーバの耐障害性についても考える必要があることがあげられる．一方，内部から監視する場合の実装は，**図 2** のように，監視プロセスがノード内の情報を取得し，その情報を解析してノード内の障害を検知する．さらに，ハートビートメッセージを互いに送信しあい，他のノードに障害が発生していないかを確認するという方法が一般的である．問題点としては，ノードの内部で収集と解析を行っているので MPI の計算プロセスの実行を阻害すること，また，ハートビートメッセージがネットワークのバンド幅を消費することがあげられる．

3. 提案システム

3.1 システムの概要

MPI アプリケーションが期待される実行性能で実行されている時には，障害が発生していないものとする．ネットワークの通信性能やノードの計算性能が著しく低下している，あるノードへメッセージが送信できない，あるいは，いつまで待ってもあるノードからメッセージが受信できない，といった状態を障害の発生とする．このとき，MPI 通信ライブラリの中で障害を検知できるのは，アプリケーションが MPI の関数を呼んだ時である．

従来のシステムでは，ミドルウェアやアプリケーションなどのユーザレベルで動作するプログラムにおいて，OS から報告されるエラーコードから障害発生の可能性を判断する．または，OS のログをみて障害発生の可能性を判断するといった方法を取っている．しかし，この方法では障害が発生した場所を正確に特定することはできない．例えば，リモートのノードに対して TCP でコネクションを張り，`write` でメッセージを送ろうとした時に “no route to host” というエラーが発生したとすると，従来の検知方法では，ケーブルが抜けたのか，相手の OS が落ちたのか，電源が落ちたのか判断ができない．

そこで本システムでは，ライブラリの通信関数の中で，OS の通信用 API がエラーを返した時に障害が発生した可能性があるとして，エラーの内容に応じた障害発見用の関数を実行するものとする．その結果，どの機器に障害が発生したのかという情報を取得できるようにする．

図 3 にシステムアーキテクチャを示す．Detection Manager は，エラーの内容と原因となる障害を特定する機能を持つ．MPI ライブラリは，Detection Interface を通じて，障害が起きた場所を知ることができる．Handler は，障害があったかどうかを確認するハンドラで，障害に対応させる Detection Manager に登録される．

アプリケーションの実行時には，Application，MPI

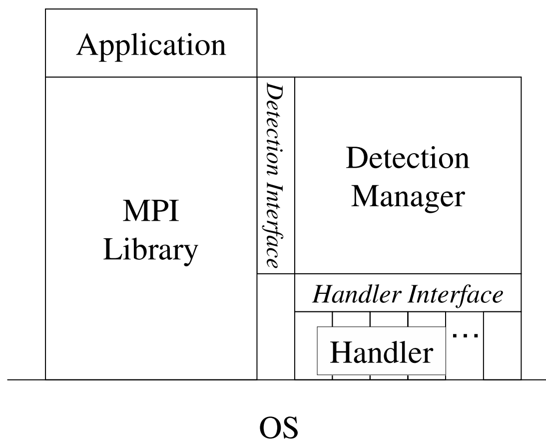

図 3 アーキテクチャ

Library，Detection Manager，Handler が 1 つのプロセスとして動作する．

3.2 ソフトウェアレベルでの障害の特定

ここでは，アプリケーションが OS から報告されるエラーを受け取った時に，どうやって障害が発生した可能性がある場所を特定するかについて述べる．

一般的に，エラーには原因となりうる障害が複数存在する．エラーがわかっただけでは，どこに障害が起きたのか特定することは不可能である．エラーを受け取った時にできることは，障害が発生した可能性がある場所の範囲を知ることである．その範囲内に含まれる機器で，どこで障害が発生したのかを確定するためには，それぞれの障害に対して確認を行っていかなければならない．

TCP/IP を用いて通信を行うアプリケーションにおいて，通信にかかわる処理を実行中に報告されるエラーと発生の可能性がある障害については次の通りである．

ENETDOWN ネットワークデバイスが使用可能な状態ではなかった場合に発生する．デバイスを起動していない可能性がある．

ENETUNREASH コネクション確立時に，相手ノードまでの経路が見つからなかった．経路上のどこかに障害が発生した場合に発生する．ケーブルが抜けた・スイッチの故障・相手ノードの NIC の障害や，相手ノードの CPU・電源など，マシンが落ちてしまう障害が発生している可能性がある．

ECONNRESET ESTABLISHED の状態の時に，コネクション確立先のノードから RST を受信した場合に発生する．相手ノードのメモリや CPU に，プロセスが異常終了してしまう障害が発生しているという可能性がある．

ETIMEDOUT 何回かメッセージを送信しても相手から反応がない場合に発生する．スイッチの故障・相手ノードのケーブルが抜けた・相手ノードの NIC の障害や，相手ノードが落ちてしまう障害が発生している可能性がある．

ECONNREFUSED SYN-SENT の状態の時に，RST を受信したときに発生する．**ECONNRESET** と同様の障害が発生している可能性がある．

EHOSTUNREACH メッセージの送信時に送信先までの経路が見つからない場合に TCP の送信がタイムアウトすると発生する．**ENETUNREACH** と同様の障害が発生している可能性がある．

このようにして，エラーから，そのエラーの原因として可能性のある障害の集合を求め，発生した障害を特定していくことが可能である．

3.3 障害モデル

障害の特定を行う上で次にあげる事柄を障害ごとにまとめる必要がある．

- どのようにして情報を集めるのか．
- 障害発生とする基準を何にするか．
- 障害が発生したら何が起きるのか．

障害の特定を行うために，障害発生の可能性がある対象の情報を集める必要がある．情報を集める方法は，障害によって異なるのが普通である．次に，集めた情報はそのままでは意味が無く，何をもって障害とするのか，基準となるものが無ければ障害を特定することはできない．最後に，ある障害が引金となって別の障害が発生することがある．例えば，CPU Fan の故障によって CPU に障害が発生しマシン全体が落ちる，といった障害の連鎖が起こりうる．

さらに，今後新しいハードウェアが追加されたり，新しいサービスが必要となってくることは，容易に想像ができる．こうした状況に備えて，新しい障害に対応するための拡張が容易に行えるような仕組みが必要である．

以上のことから，障害ごとに抽象化して扱いやすくするために，障害モデルを提案する．障害モデルは，障害 1 つずつに対応するように作成される．つまり，CPU の障害を扱うモデルは CPU 障害モデルであり，メモリの障害を扱うモデルはメモリ障害モデルである．

障害モデルは以下から構成される．

名前 障害モデルが表現している障害の名前であり，CPU，メモリ，温度などである．

障害の詳細 発生している障害の詳細な情報であり，CPU では内部エラーや温度が高いなど，メモリでは ECC エラー，CPU Fan では回転数といった内容である．

検知方法 障害発生の可能性がある対象の情報を集める機能．IPMI や SNMP のリクエストを送信，OS のログを確認，コマンドを実行，さまざまな方法で情報を集める．

隠れる障害 この障害と同じエラーを発生させる障害であり，この障害が発生していることにより検知

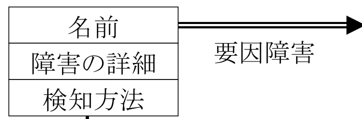

図 4 障害モデルイメージ

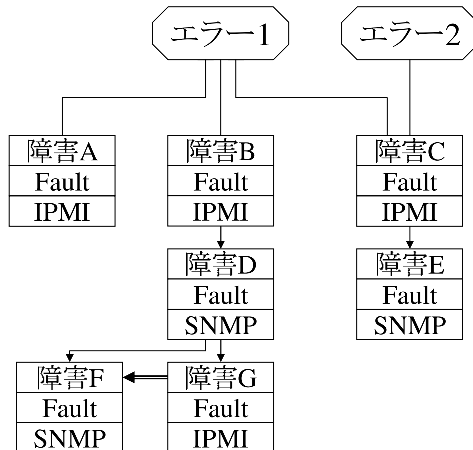

図 5 障害モデル関係図

が不可能になる障害．例えば，NIC に障害が発生しているときには，スイッチなどネットワーク越しに障害を検知する機器について，障害の検知が不可能になる．

要因障害 この障害が原因となって引き起こされる障害．例えば，CPU Fan に障害が発生した場合，CPU の温度が上昇し CPU にも障害が発生する．

障害モデルを**図 4** に示し，エラーと障害の関係を**図 5** に示す．図 5 では，“エラー 1，2” から出ている線はエラーの原因となる障害を示している．また，“障害 B” が発生した場合，“障害 D，F，G” の障害発生の有無を確認することができないことと，“障害 G” が発生することが原因で “障害 F” が発生することを表現している．

障害モデルを用いることによって，障害ごとに必要な事柄がまとめられ，どこに新しい障害が追加されるべきなのか理解しやすく，新しい障害に対応したシステムを容易に再構築することができる．

3.4 動　　作

動作の具体例として，**図 6** のような状況において Node 0 からの `write` が，“no route to host” というエラーで失敗したときを考える．このエラーの場合，表 1 により，次の（1）から（6）に示す障害と，さらに（6）について 6 種類の障害の可能性がある．

（1） Node 0 の NIC に異常が発生
（2） スイッチにおいて，Node 0 からのケーブルコネクタに異常が発生

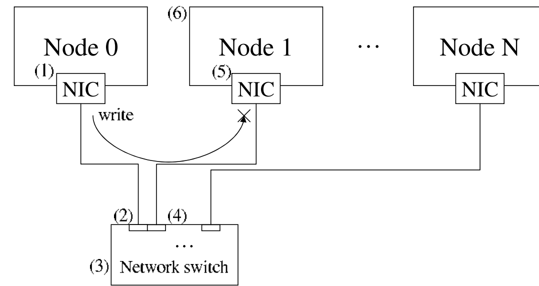

図 6 動作の具体例

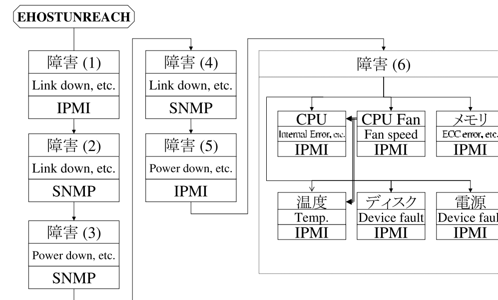

図 7 障害モデルによる表現

（3） スイッチに異常が発生
（4） スイッチにおいて，Node 1 からのケーブルコネクタに異常が発生
（5） Node 1 の NIC に異常が発生
（6） Node 1 に異常が発生
- CPU
- CPU Fan
- メモリ
- 温度
- ディスク
- 電源

エラーとこれらの障害の関係を障害モデルを用いて表すと**図 7** のようになる．

Detection Manager は，MPI ライブラリから **EHOSTUNREACH** というエラーについて障害を特定することを求められたら，“障害（1）” から順番に障害の発生を確認していく．

結果は，障害の発生が確認できたものは障害が発生しているとして，確認を行うことができなかったものは，未知の障害が存在するとして MPI ライブラリに通知される．

3.5 実　　装

3.5.1 Detection Interface

MPI ライブラリに対して，Detection Manager を生成する関数と障害の確認要求に関する関数とを提供する．

確認要求に関しては，要求の生成および削除の関数として `detReqCreate` と `detReqDelete` を用意する．

```
ret = write( ... );
if( ret == -1 && need_detect(errno) ) {
  DetReq *dreq = detReqCreate( errno );
  /* errno == EHOSTUNREACH */
  detQuery( dreq );
  for( dreq->faults )  {
    ...
  }
  detReqDelete( dreq );
}
```

図 8 コードサンプル

作成した要求を用いて問い合わせを行う関数として `detQuery` を用意する．要求には，次にあげる内容を含む．

エラー ID OS や MPI ライブラリに依存しない一意の値が割り当てられた ID である．リクエストの生成時に値がセットされる．

障害 ID のリスト エラー ID と同じく，OS や MPI ライブラリに依存しない一意の値が割り当てられた ID である．問い合わせ後に値がセットされる．

MPI ライブラリからは，**図 8** のように使用される．まず，`write` がエラーを返した時に，エラーが障害発生を示す内容だった場合，エラーをもとに要求を生成する．生成した要求を用いて障害の特定を行う．最後に，特定された障害について順番に処理を行うことが出来る．

3.5.2 Detection Manager

Detection Manager では，障害情報の収集と障害発生の確認を行う．

ここで，Detection Interface を通して問い合わされた要求をもとに，`FaultModel` 構造体を用いて図 5 で表現されるような構造が作成される．`FaultModel` 構造体の主な変数として `FaultModelElem` 構造体がある．この 2 つの構造体の具体的例を示すと，CPU という機器を表現する `FaultModel` 構造体の中に，CPU の熱や内部エラーといった CPU に関する個々の事象をあらわす `FaultModelElem` 構造体が存在する，という関係である．`FaultModelElem` 構造体は，機器の情報を収集する関数と，収集した情報から障害が発生しているか判断するための情報を持つ．

`FaultModelElem` 構造体の主要な変数は次の通りである．

`det_threshold_high` 閾値として障害判別の基準として用いる値．温度などがこの値を上まわっているときに障害が発生していると判断する．

`det_threshold_low` 閾値として障害判別の基準として用いる値．温度などがこの値を下まわっているときに障害が発生していると判断する．

`det_handler` 機器の各事象の情報を取得する関数へのポインタ．

3.5.3 Handler Interface

Detection Manager に対して，IPMI や SNMP などの実際にデータを収集する関数を提供する．

3.6 Handler

Handler では，IPMI や SNMP などの実際にデータを収集する関数が用意されている．Detection Manager は，Handler Interface を通してこの関数を使用することにより，目的のデータを得ることができる．

4. 関 連 研 究

MPI に耐障害性を持たせるための研究は多く存在する．

MPI/FT[1] は，アプリケーションのモデル（Master-slave,SPMD）に応じて冗長性やチェックポイントの有無を決定する．MPI/FT は，self-checking thread（SCT）がノード内の計算スレッドと通信用スレッドが動いていることを確認するという internal heartbeats と，SCT が他ノードの SCT からのハートビートメッセージを受け取って，MPI のプロセスが動いていることを確認するという external heartbeats との 2 つを用いて障害を検知している．障害が全くない場合でも障害検知のための通信を行っており，性能の低下を招いている．また，障害の検知をアプリケーションレベルでしか行っておらず，障害の詳細については知ることができない．

FT-MPI[4] は，communicator レベルで障害を検知する．fail したランクがあると MPI の関数の戻り値によって障害の情報が他のランクに送られる．FT-MPI は，HARNESS[2] システムのプラグインとして実装されている．FT-MPI は，ノード間の通信に障害が発生した場合と，OS が検知した障害を HARNESS Layer を通して検知した場合に障害が発生したと判断する．障害が全くない場合でも障害検知のために OS の監視を行っており，性能の低下を招いている．障害の検知をアプリケーションレベルと OS レベルでしか行っておらず，障害の詳細については知ることができない．

MPI/FT や FT-MPI と比較して，本システムは障害の特定が行える点が異なる．さらに，障害が全くない場合において計算とは関係の無い処理を必要としないことが優位な点である．

MPICH-V2[3] は，障害を検知するアプリケーションが存在することを仮定している．よって，MPI のアプリケーションと障害検知アプリケーションが同時に動作していることになる．本システムでは，別途に検知用アプリケーションを必要としていない．

LA-MPI[5] は，プロセスの障害を検知するのではなく，TCP/IP のような信頼性のあるデータ転送ではなくて，より性能が高いが信頼性の無いデータ転送を用いても信頼性のあるメッセージの配送が可能になること目指している．そのために，通信レイヤーを Mem-

ory and Message Management Layer と Send and Receive Layer に分け，前者でパケットの再送やルート選択を行い，後者はエラーを返すようになっている．メッセージの送受信の性能はよいが，通信の障害しか検知できない．障害の可能性を知る方法が，本システムと同じく通信のエラーが発生した時であるが，検知できる障害の幅広さと障害の詳細についてが本システムの優位な点である．

CuckooMPI[10] は，リカバリモデルや耐障害機能の部分をコンポーネントとして提供し，環境やアプリケーションに最適な耐障害性 MPI を用いることを可能にしている．障害が発生すると，Fault Detector が障害のモデルに応じてリカバリプロトコルを選択する．障害の検知については詳しく述べられていない．

MPI とは関係ないが，障害を検知する機能を持ったソフトウェアとして，IBM の Tivoli Enterprise Console[6] および IBM Tivoli Monitoring[7] がある．前者は，情報の受信およびルールにしたがって障害が発生していないかをチェックする Event server と，ノード内のアプリケーションやシステムリソースの状態を収集し，その情報を Event server に送信する Event adapter からなる．後者は，監視用のプロセスが CIM とよばれる規格にもとづいてノード内の障害を検知する．しかし，Windows 以外の OS では CIM の実装が不十分であるから，あまり詳細な情報は得られず監視できるリソースは少ない．

5. おわりに

既存の耐障害性 MPI ライブラリでは，障害検知を正確に行うことは重視されていなかった．本稿では，障害検知を正確に行えることを目標としたシステムを提案した．その中で，障害を特定することに関して必要な事柄をまとめ，かつ，新しく障害の対象が増えた時でも容易にシステムを拡張できるように，障害モデルを提案した．

今後は，提案したシステム・障害モデルを実装する．そして，障害が発生していない時には本来の性能が出せることを，障害が発生した時にはその障害が特定できることを確認する．

謝　　辞

本研究の一部は，文部科学省「eSociety 基盤ソフトウェアの総合開発」の委託による．

参 考 文 献

1) Batchu, R., Dandass, Y. S., Skjellum, A. and Beddhu, M.: MPI/FT: A Model-Based Approach to Low-Overhead Fault Tolerant Message-Passing Middleware, *Cluster Computing*, Vol.7, No.4, pp.303–315 (2004).
2) Beck, M., Dongarra, J.J., Fagg, G.E., Geist, G.A., Gray, P., Kohl, J., Migliardi, M., Moore, K., Moore, T., Papadopoulous, P., Scott, S.L. and Sunderam, V.: HARNESS: A Next Generation Distributed Virtual Machine, *Future Generation Computer Systems*, Vol.15, No.5–6, pp. 571–582 (1999).
3) Bouteiller, A., Cappello, F., Hérault, T., Krawezik, G., Lemarinier, P. and Magnitte, F.: MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on the Pessimistic Sender Based Message Logging, *In proceedings of The IEEE/ACM SC2003 Conference*, Phoenix USA (2003).
4) Fagg, G.E., Gabriel, E., Chen, Z., Angskun, T., Bosilca, G., Bukovsky, A. and Dongarra, J. J.: Fault Tolerant Communication Library and Applications for High Performance Computing, *Proceedings of the 4th Los Alamos Computer Science Institute Symposium (LACSI'03)* (2003).
5) Graham, R.L., Choi, S.-E., Daniel, D.J., Desai, N. N., Minnich, R. G., Rasmussen, C. E., Risinger, L.D. and Sukalski, M.W.: A Network-Failure-Tolerant Message-Passing System for Terascale Clusters, *International Journal of Parallel Programing*, Vol.31, No.4, pp.285–303 (2003).
6) IBM: IBM Tivoli Enterprise Console. http://www-306.ibm.com/software/tivoli/products/enterprise-console/.
7) IBM: IBM Tivoli Monitoring. http://www-306.ibm.com/software/tivoli/products/monitor/.
8) Intel, Hewlett-Packrd, NEC and Dell: Intelligent Platform Management Interface Specification Second Generation v2.0 (2004).
9) SNMP: http://www.ietf.org/rfc/rfc1157.txt.
10) 貫本英之，松岡　聡：ポータブルな耐障害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，先端的計算基盤シンポジウム SACSIS 2005 論文誌，Vol.2005, No.5, pp.228–229 (2005).