社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006−OS−103 (17)
2006／8／1

CIMを用いた障害検知システム

酒井将人† 石川 裕†

システムの管理インターフェースとしてCIM (Common Information Model) が策定されている．CIMに新しいクラスを追加しそのクラスを用いた障害検知システムを提案する．既存の障害検知システムには3つの問題が存在する．1つは設定が複雑であること，2つ目は性能を低下させること，最後は検知結果が正しいとは限らないことである．著者らは，これらの問題に対してMPI通信ライブラリに追加する障害検知機能を既に提案した．既提案を元にして障害検知システムを実現すると，検査方法の設定が複雑になってしまうという問題が生じた．この問題を，検査方法を機器情報と関連付けてデータベースに保存することにより解決する．解決方法を実現するために新しくCIMクラスを追加した．これにより検査方法の設定が容易になる．本稿では，新しく追加したクラスについて述べ，追加したクラスを用いた障害検知システムの設計と実装を示す．

A Fault Detection System with Common Information Model

Masato Sakai† and Yutaka Ishikawa†

Common Information Model (CIM) is designed as a system management interface. We introduce new CIM classes and propose a fault detection system with the classes. There are three issues in existing fault detection system. First, it is complex work to configure a detection system. Second, network and computational performance is decreased by periodic monitoring operations. Last, detection systems may return false detection. We have already proposed a fault detection component adding to an MPI Communication library for solving these issues. Using this system, we have found that it becomes complex work to configure how to monitor devices. We resolve this issue by storing a monitoring method and the relation between the method and devices in database. We add new CIM classes to store these two object in database. By this solution, it is easy to describe the configuration about monitoring tools. In this paper, we describe new classes and, show design and implementation about the new fault detection system with the new classes.

1. はじめに

近年，科学技術計算やビジネス用途などで大規模なクラスタシステムが広く利用されている．このようなクラスタは，CPU/メモリ/ディスク/ネットワークスイッチなどの数多くのデバイス類によって構成されている．構成機器の数が増えるとシステム全体としての故障率は増大する．実行時間が非常に長い科学計算では故障発生後も計算を続けられるように，ビジネス用途ではサービスのダウンタイムを短くするために，障害検知を正しくおこなうことが重要になっている．

障害検知のための研究や製品は多く存在する．IBM Tivoli Monitoring[6] やNagios[1] はクラスタのホストやスイッチを定期的に監視する．JAMM Monitoring System[8] やGridMonitor[9] は，Grid環境において障害検知を含めたさまざまな用途のためにGrid環境内の情報を収集する．他にも，ハイパフォーマンスコンピューティングの分野において耐障害性並列計算用通信ライブラリのために検知のオーバーヘッドを抑えた障害検知機能についての研究が存在する．MPI/FT[3] では，MPIアプリケーションの通信パターン毎に適した監視方法を選択することで，オーバーヘッドの少ない障害検知機能を実現している．

しかし，これらの障害検知システムには3つの問題が存在する．1つは，障害検知システムの設定が非常に複雑であるという問題である．なぜなら，デバイスやサービスといったクラスタシステムを構成するさまざまな要素間の関係をユーザが考慮する必要があるからである．ここでいう関係とは，例えばネットワークトポロジやサービス間の依存関係である．

2つ目は，検知処理がパフォーマンスの低下を招くという問題である．障害が発生していない状況においても周期的に発行されるリモートにあるデバイスやサービスへの検査用メッセージがネットワークのバンド幅を低下させる．また，監視対象のホストにおいて

† 東京大学
The University of Tokyo

検査用に情報を収集する処理がCPUやメモリといったリソースを消費することになる．

3つ目は，検知エンジンが障害の原因を正しく特定できていない可能性があるという問題である．例えば，リモートホストとその経路上にあるスイッチを監視している場合に，スイッチで故障が発生すると，スイッチとホストの両方で障害が発生したという結果を返してしまう．

以上の問題に対して著者らは論文10)において，ハイパフォーマンスコンピューティング分野用に耐障害MPI通信ライブラリのための障害検知機能を提案した．しかし，この提案を元に一般的な用途向けに障害検知システムを設計したところ，障害検知システムの設定を容易にするはずが，場合によっては設定がより複雑になる．本稿では，この問題を解決する．以降，第2節で既提案手法の概要を示す．第3節で既提案手法の問題点を挙げ，それに対応する解決方法を挙げる．第4節で解決方法について説明する．第5節，第6節で障害検知システムの設計と実装を示す．

2. 既提案手法の問題

第1節において挙げた既存の検知システムの問題点に対して，著者らは論文10)において，ハイパフォーマンスコンピューティング分野用に耐障害MPI通信ライブラリのための障害検知エンジンを提案した．提案した障害検知エンジンでは，最初の問題に対して，ユーザが考慮する設定の構成要素（障害/実際の検知対象/検知方法/ネットワークトポロジ等）間の関係を減らすことによって設定を用意にするというアプローチで解決した．具体的には，障害/障害間の依存関係/障害を検査する方法の3つを記述した設定ファイルと，CIM (Common Information Model)[4]を用いデバイスやネットワークトポロジを表現したデータベースの2つに設定を分離した．CIMはDMTF (Distributed Management Task Force) で標準化が推められているシステム機器管理インターフェース標準である．さまざまなデバイスを統一されたインターフェースで管理することを目的としている．障害検知エンジンは検査対象をデータベースから参照し，その対象へ検査ツールを実行するという方法で障害検知を行う．どのようにCIMを利用したかは次節で述べる．

2番目の問題に対して，OS等から報告される情報を引き金として検知処理を開始するアプローチで解決した．検知エンジンは報告がない限り動作することがない．よって，通信と計算のパフォーマンスに影響を与えない．そのために，報告（エラー情報）とその原因となる障害の関係を設定ファイルに記述する．障害検知エンジンは，この関係をもとに，報告された情報から検査するべき障害のリストを生成する．ネットワークトポロジを設定から分離させているため，この関係を記述する際にネットワークトポロジを意識する必要なく記述することが可能である．

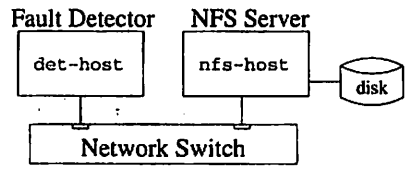

図1 機器構成例

最後の問題に対して，障害間に依存関係を持たせることで誤検知を防ぐ．ある障害が発生すると，依存関係がある別の障害の検査が不可能になるという関係を導入した．この関係はユーザが定義する必要なく，障害検知エンジンがデータベースにあるデバイスやネットワークトポロジから自動生成する．

3. 既提案におけるCIM利用の問題

CIMでは，機器の構成/ネットワークトポロジを表現することが可能である．CIMを用いて表現した構成をデータベースに保存してそのデータベースを利用することによって，“あるホストまでの経路上に存在するスイッチ”といった情報をソフトウェアが取得することが可能になる．図1の機器構成例を用いて例を示す．図1において，`det-host`は障害を検知するホストであり，検知エンジンが動作している．`nfs-host`はNFSサービスを提供している．diskは`nfs-host`に設置されているディスクである．そして，`det-host`と`nfs-host`は，Network Switchにつながっている．この例において，`nfs-host`のホスト名やIPアドレスをもとにして，検知エンジンが動いている`det-host`から`nfs-host`の経路上にあるスイッチを列挙することが可能である．機器の構成やネットワークトポロジに由来する依存関係を検知エンジンが自動で補完することが可能である．その結果，障害と検査用ツールとそれらの間の関係を記述した設定ファイルと，CIMを用いたハードウェアとネットワークトポロジを表現したデータベースを準備するだけで動作する障害検知システムを作ることは可能である．

検査対象となるデバイスやソフトウェアがローカル/リモートのどちらにあっても動くような検査方法を設定することや，さまざまなデバイスの種類に対応した検査方法を設定することを考えた場合に，上記の設計では問題が生じる．例えば，ディスクの状態を調べたい場合に，ローカルにある場合とリモートにある場合では使用するコマンドや必要となる引数が異なる．検査方法の設定として，図2にディスクの障害に対する設定の概要を示す．この例は，

- ディスクがローカルの機器の場合とリモートの機器の場合では検査方法が異なること

```
if ローカルのディスク then
  S.M.A.R.T. を使った検査
else if リモートのディスク then
  if IPMI が有効 then
    IPMI を使った検査
  else if SNMP が有効 then
    SNMP を使った検査
  end if
end if
```

図 2 検査方法の概要

- リモートの機器の場合において，ディスクが設置されているホストに対して，IPMI[7] が利用できる場合と，IPMI は利用できないが SNMP[2] が利用できる場合では検査方法がことなること

という状況に利用できる設定である．この例のように，論文 10) での提案を用いた場合，検査方法の設定が条件分岐を含んだものになる．さらに，検知エンジンが条件を評価するために，どのパラメータを用いるべきかをヒントとして検査方法の設定の中に記述する必要もあった．このように，検査方法を障害の情報として持たせた設計では，多種多様な障害やデバイスやソフトウェアに対応するように検査方法を記述することが非常に困難な作業になってしまった．

本稿では，検査方法と検査対象を関連付けて設定する方法によりこの問題を解決し，論文 10) における問題点を解決した障害検知システムを提案する．検査対象に対して有効な検査方法を設定するという設計により，検査方法の設定が直感的であり容易になっている．しかし，CIM の標準クラスの中には，検査方法を表現するクラスおよび機器と検査方法の間の関連を表現するクラスが存在しない．故に，検査対象に検査方法を関連付けるために新しい CIM のクラスを定義する．

4. CIM クラスの追加

第 3 節で述べたように，検査対象がローカルに存在する場合とリモートに存在する場合の両方を表現するクラスを追加する．追加するクラスを UML のクラス図を用いて表現すると図 3 となる．図 3 において，3 段に分割された四角形がクラスを示し，上段から，クラス名/変数/メソッドが書かれている．変数は「変数名:変数の型」と記述される．

FDM_Check 検査方法をあらわす抽象クラスである．変数として，検査用のコマンド名をあらわす文字列型の `command` と検査結果を判定するための整数列挙型の `status` が存在する．メソッドは存在しない．**FDM_LocalhostCheck** は，**FDM_Check** のサブクラスであり，検査対象がローカルホストにある場合の検査方法をあらわす具象クラスである．独自の変数を持たず，メソッドも存在しない．同様に **FDM_RemotehostCheck** は，**FDM_Check** のサブクラスであり，検査対象がリモートホストにある場合の検査方法をあらわす具象クラスである．独自の変数を持たず，メソッドも存在しない．

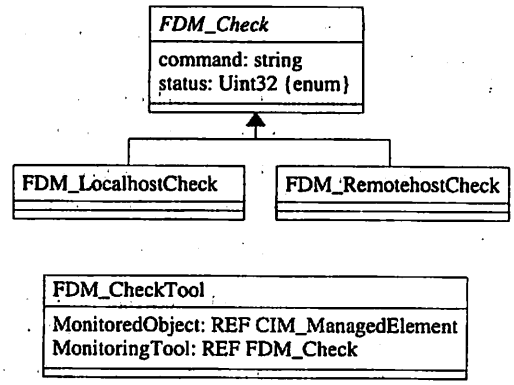

図 3 追加したクラス

FDM_CheckTool 検査対象と検査方法の関連をあらわす具象クラスである．変数として，関連の中で検査対象を指す **CIM_ManagedElement** クラスへの参照型の `MonitoredObject` と，利用する検査方法を指す **FDM_Check** クラスへの参照型の `MonitoringTool` が存在する．メソッドは存在しない．実際のデータベース上では，`MonitoredObject` は CIM の標準のクラスまたはそれを継承して追加したクラスのインスタンスへの参照を示す．`MonitoringTool` は **FDM_LocalcheckTool** か **FDM_RemotecheckTool** のどちらかのクラスのインスタンスへの参照を示す．

これらのクラスの利用例を図 1 の機器構成例を用いて示す．図 4 に，追加したクラスを用いて，`nfs-host` のホストとディスクと NFS サービスおよびそれらの検査方法のインスタンスの関係を示す．本図では UML オブジェクト図で表現されている．図 4 において 2 段に分割された四角形がインスタンスを示している．上段には，「インスタンス名：クラス名」を書くが，図 4 ではすべてのインスタンスにおいてインスタンス名を省略している．下段は，変数の値を示している．図 4 では代表的な変数だけを記している．`nfs-host` と名前がつけられている **CIM_UnitaryComputerSystem** というクラスのインスタンスがホストをあらわしている．**CIM_LogicalDisk** というクラスのインスタンスがディスクをあらわし，`NFS` と名前がつけられている **CIM_Service** というクラスのインスタンスが NFS サービスをあらわす．

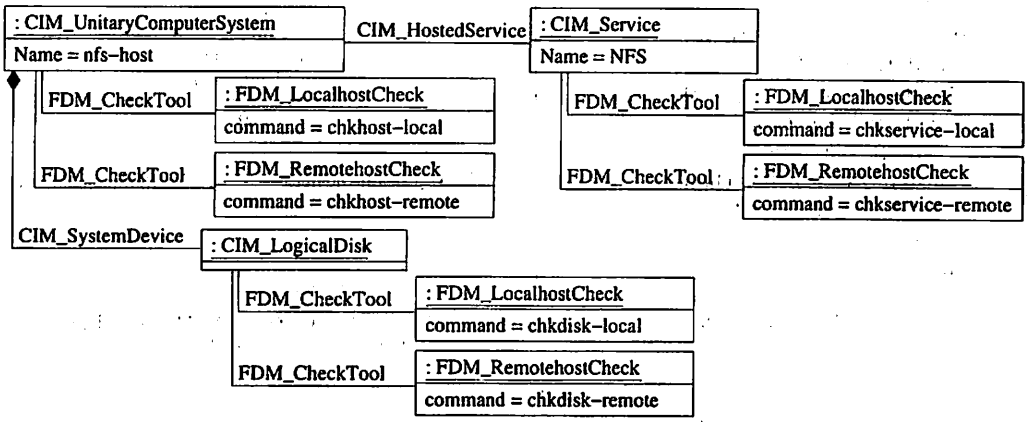

図 4 nfs-host 計算機の構成に対応する CIM データベース

CIM_HostedService はサービスがコンピュータシステム上で提供されていることをあらわすアソシエーションである．アソシエーションとはインスタンス間に関連が存在することを示すものである．**CIM_HostedService** によって NFS サービスがホストで提供されていることを表現している．**CIM_SystemDevice** はデバイスがコンピュータシステムに部品として装着されていることをあらわすコンポジションである．コンポジションとはインスタンス間に依存関係が存在することを示し，両インスタンスのライフサイクルがほぼ一致する場合に用いられるものである．**CIM_SystemDevice** によってディスクがホストの構成部品のひとつであることを表現している．

検査方法については，検知エンジンから見て，ホストがローカルホストである場合に `chkhost-local` を用い，リモートホストである場合に `chkhost-remote` を用いることを示している．サービスやディスクの場合も同様にそれぞれローカルホストにある場合とリモートホストにある場合の両方に 1 つずつ検査方法が示されいてる．

これらの追加したクラスを用いることによって，ソフトウェアでの自動処理が可能になる．すなわち，管理者が検知エンジンの設定を行うときに考慮する関係が減ったことを示している．

5. 障害検知システム

OS 等からの異常を知らせる報告をアプリケーションが受け取った時にその原因を自動的に調べる障害検知システムの設計を提案する．また，障害検知システムの設定が容易にすることを目的とした CIM の拡張方法を提案する．提案した障害検知システムにおいて，"アプリケーションが観測する異常" をエラー，その "原因" を障害とよぶ．エラーの発生後にはエラーの原因として考えられる障害だけでなく，その障害を調査する上で正しく動作している必要があるデバイスやソフトウェアも調査対象に加え障害検知を行う．

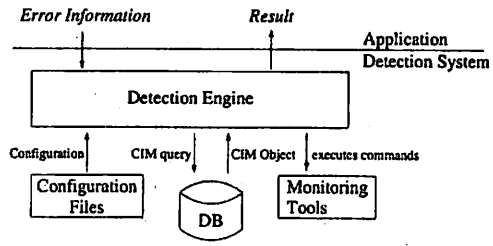

図 5 障害検知エンジンの概要

図 5 に障害検知エンジンの概要を示す．アプリケーションからエラー情報を受け取って，障害が観測されたデバイス/ソフトウェアを結果として返す．設定ファイルにはエラーと障害の関係および障害が発生するデバイスを特定するために検知エンジンが利用する情報が記述されている．データベースにはデバイスやソフトウェアの構成，どのツールを使って検査を行うかという情報，デバイスやソフトウェアと検査方法の関係の 3 種類の情報が保存されている．検査用ツールは検知エンジンが障害発生を検査するために使用するツール類のことである．

5.1 障害検知システムの設定例

障害検知システムに読み込まれる設定ファイルには以下の内容が記述される．

- エラーとその原因となる障害の関係
- 各障害の詳細な設定
  - 障害が発生するデバイスまたはソフトウェアをデータベースから取り出すためのクエリ
  - 正常である必要があるデバイスまたはソフト

```
<error id="HostUnreach">
  <fault name="RemoteHostDown" />
</error>
```

図 6 エラーと障害の関係

```
<fault name="RemoteHostDown">
  <identify>
    <param name="HOSTNAME" />
    <query>
select OBJECTPATH(CIM_UnitaryComputerSystem)
         AS Path
  from CIM_UnitaryComputerSystem
 where CIM_UnitaryCOmputerSystem =
         '$HOSTNAME'
    </query>
  </identify>
</fault>
```

図 7 障害の詳細

ウェアの一覧をデータベースから取り出すためのクエリ

実際には，エラーと障害の関係と障害の詳細設定は分けて記述される．

設定ファイルは XML で記述する．図 6 と図 7 に例を示す．図 6 はエラーと原因となる障害の関係を定義している．例では，HostUnreach という id で特定される error の原因として，RemoteHostDown が考えられるということをあらわしてる．ユーザはエラーの id とその原因となる障害を定義するだけであり，障害の内容について深く知る必要はない．図 7 では障害の詳細を定義している．例では，RemoteHostDown という名前をもつ障害について定義している．この障害の詳細には，検査対象となるオブジェクトをデータベースから取得する方法が identify によって定義されている．identify の中には，データベースに発行するクエリと対象となるオブジェクトを一意に定めるためのパラメータが定義される．データベースに発行するクエリは query によって定義され，パラメータは param によって定義される．パラメータの名前は仕様により定義されている文字列であり，意味は固定である．この例では，HOSTNAME というホスト名を意味するパラメータが定義されている．param は複数定義されることがあるが，query は 1 つしか定義されない．

障害検知システムを利用するためにユーザが設定するものは，設定ファイルである図 6，図 7 と，図 4 に示すようなデータベースである．

5.2 インターフェース

アプリケーションから検知エンジンを利用する方法を図 8 に示す．この例は，あるホストと通信を行おうとした際に EHOSTUNREACH というエラーが発生した場合における障害検知までのコードを示している．はじ

```
if( errno == EHOSTUNREACH ) {
  DetConfig *conf
     = createDetConfig("HostUnreach");
  setParam(conf, HOSTNAME,
           FDM_HOST, "nfs-host");
  DetResult *ret = performCheck(conf);
}
```

図 8 検知エンジンの利用方法

表 1 パラメータタイプの例

パラメータタイプ	意味	例
FDM_HOST	ホスト名	"nfs-host"
FDM_FILE	ファイル名	"/etc/hosts"
FDM_SERVICE	サービス名	"httpd"
FDM_SOCKET	ソケット構造体	*sockaddr

めに，設定をあらわす DetConfig オブジェクトを生成する．つぎに，生成したオブジェクトにパラメータを与える．ここでは，ホスト名が必要なのでホスト名をパラメータとして与える．setParam() の第 2 引数は，図 7 に示したパラメータを指定している．第 3 引数は，パラメータのタイプを指定する定数であり，表 1 に示すようなものがある．表 1 の例とは，setParam() の第 4 引数に指定する値の例である．最後に，設定オブジェクト引数として検知エンジンの検査を開始する．結果を DetResult オブジェクトとして受け取る．DetResult には，障害を検知した対象の数と検査が不可能だった対象の数が設定されている．また，障害を検知した対象を示すデータベース上のオブジェクトのキーとなる値のリストと検査が不可能だった対象を示すオブジェクトのキーとなる値のリストも持っている．

6. 実　　装

6.1 障害の情報を取得

設定ファイルに記述されたエラーと障害の関係と障害の詳細な設定を読み込み，検査する必要がある障害の情報を検査候補リストに追加する．障害の情報とは，図 7 の identify 要素の中身を指している．

はじめに，入力されたエラー情報をもとに必要となる関係を取り出す．例えば図 6 の関係は，"HostUnreach" というエラー情報を受けとった時に使用される．つぎに，取り出した関係の記述をもとに，検査対象となる障害の詳細な設定を取り出す．例えば，取り出した関係が図 6 の場合，原因となる障害は RemoteHostDown であると記述されているので，障害の詳細の定義の中から RemoteHostDown と名前がついた障害を検査候補のリストに追加する．

以上の処理を XSLT （XSL Tranformation） を使って行う．XSLT とは，任意の XML 文書を読み込み，それを加工して出力する変換用の言語である．変

```
if target-object は CIM_UnitaryComputerSystem のインスタンス then
  target-host ⇐ target-object
else
  target-host ⇐ target-object からホストを取得するクエリの実行結果
end if
local-host ⇐ 自ホストを取得するクエリの実行結果
if local-host と target-host が等しい then
  target-object はローカルに存在する
else
  target-object はリモートに存在する
end if
```

図 9 位置を判定するアルゴリズム

換ルールは，設定ファイルに内容によらないものなのでユーザがその記述を気にすることは無い．

6.2 検査対象の特定

前節において検査候補として，障害の情報を取り出した．取り出した情報の中にあるパラメータとクエリを用いて，検査対象を示す CIM のインスタンスを取り出す．例えば，RemoteHostDown という障害の場合，クエリ内の${HOSTNAME}をインターフェースを通して HOSTNAME に関連付けられた値に置き換えてデータベースにクエリを発行する．その結果，HOSTNAME がもつ値をホスト名に持つホストのインスタンスの参照を得ることができる．

6.3 位置の判定

ローカルホストのデバイスとリモートホストのデバイスのそれぞれの検査方法を定義する方法を用意したが，実際に検査を行うときには，対象となるデバイスがローカルにあるのかリモートにあるのか特定しなければ定義した検査方法を利用する事はできない．ここでは監視対象のデバイスの位置を特定する方法について述べる．図 1 において，Fault Detector から NFS Server ディスクを調べる状況を仮定する．また，インスタンスの例として図 4 を用いる．

位置の判定アルゴリズムを図 9 に示す．target-object は検査対象を示し，target-host は検査対象が設置されているホストを示し，loca-host は検知エンジンが動作しているホストを示している．target-object からホストを取得するクエリを CQL (CIM Query Language)[5] で記述した例を，図 10 に示す．ここで OBJECTPATH とは，データベース上のインスタンスの参照を返す CQL の関数である．図 10 は，path to a monitored disk が図 4 の CIM_LogicalDisk のインスタンスの参照であれば，CIM_SystemDevice で関連付けられてる nfs-host と名前のついた CIM_UnitaryComputerSystem のインスタンスの参照を返すクエリである．さらに，自ホストを取得するクエリを

```
select OBJECTPATH(CIM_UnitaryComputerSystem)
         AS Path
  from CIM_UnitaryComputerSystem,
       CIM_LogialDevice,
       CIM_SystemDevice
 where CIM_SystemDevice.GroupComponent =
       OBJECTPATH(CIM_UnitaryComputerSystem)
   and CIM_SystemDevice.PartComponent =
       'path to a monitored disk'
```

図 10 デバイスからホストを取得する CQL

```
select OBJECTPATH(CIM_UnitaryComputerSystem)
         AS Path
  from CIM_UnitaryComputerSystem
 where CIM_UnitaruComputerSystem.Name =
       'host name'
```

図 11 ホストを取り出す CQL

図 11 に示す．先に挙げた仮定のもとでは，host name に det-host を指定することで det-host を示すインスタンスの参照を得ることができる．

6.4 検査ツールの実行

検査対象の特定と位置の判定が終わると検査用の設定を取得することが可能になる．ここでは検査用の設定を取得し検査ツールを実行する方法について述べる．前節と同じ仮定のもとで説明する．

図 4 に示す通り，検査対象となるオブジェクトにはそれぞれローカル用とリモート用の両方の検査用の設定が関連づけられているので，その設定をデータベースから取得する．リモート用の設定を取り出す例を図 12 に示す．path to a monitored object に検査対象であるディスクのインスタンスの参照を指定することで，図 4 の中の，command という変数が chkdisk-remote である，FDM_RemotehostCheck のインスタンスの参照を得ることができる．

検査方法インスタンスの取得後は，コマンドを取得

```
select OBJECTPATH(FDM_RemotehostCheck)
         AS Path
  from FDM_RemotehostCheck,
       FDM_CheckTool,
       CIM_ManagedElement
 where FDM_CheckTool.MonitoredObject =
         'path to a monitored object'
   and FDM_CheckTool.MonitoringTool =
         OBJECTPATH(FDM_RemotehostCheck)
```

図 12 検査用オブジェクトの取得（リモート用）

し実行する．コマンドの実行結果・終了コードをもって対象に障害が発生しているかどうかの判定を行う．

検査対象となるオブジェクトを特定後，そのオブジェクトに関連付いている検査方法を盲目的に実行して障害発生の確認を行う．実行時に検査ツールの検査内容を考慮しない．1つ以上のツールが障害発生を示す結果を返せば，検査対象に障害が発生していると判断する．障害発生箇所の切り分けは，データベース上に保存する時点で既に行われているので，検査ツールの段階で問題箇所の切り分けをする必要はない．

7. 関連研究

7.1 障害検知システム

IBM Tivoli Monitoring[6] は，システムのボトルネックになっている機器や将来障害が発生するであろう機器を検知するシステムである．監視の周期や監視対象の情報は，“Resource model” と呼ばれる論理モデルに定義される．検知エンジンはその Resource model をもとに動作する．検知エンジンの設定は，あらかじめ用意された Resource model の中から必要なものを選択するという方法で設定する．新しいデバイスや新しい障害に対応する場合には，そのための Resource model を生成する必要がある．そのために，情報収集する機能を Java で実装し，検査対象や検査周期についての設定を記述する必要があり，拡張が容易な仕様であるとはいえない．また，周期的にデータ収集を行っていることから，CPU やメモリ等のリソースを消費している．さらに，誤検知を防ぐ仕組みが存在しない．

Nagios[1] は，サーバやワークステーションやネットワークスイッチおよびホストで提供されているサービスを監視するシステムである．このシステムの設定には，検査対象と検査方法と検査周期を設定する．Nagios には，検査対象に障害が発生している状態と，検査対象まで到達できないという状態を区別する方法がある．監視設定に含まれる “parent” というオプションを用いて実現してる．ある監視対象のチェックコマンドが異常を示すものであれば，正常である結果が得られるまで parent で指された監視対象を調べる．ネットワークトポロジを考慮した設定を記述する必要があり，設定が容易であるとは言えない．さらに，Nagios も周期的に検査を行っているので，リソースを消費している．

7.2 CIM の拡張

文献 11) では，障害メッセージの分析のために CIM を拡張した例を示している．得られたメッセージを分析し，メッセージ間の因果関係をユーザに示すことが可能になる拡張である．メッセージの因果関係により根本となる原因をユーザに示すことができる．しかし，メッセージが得られないような障害，例えばハードウエアの故障によるホストダウンなどを検知するために役に立つような拡張ではない．

8. まとめと今後の課題

CIM を用いた障害検知システムを提案した．既存の障害検知システムに存在する，1) 複雑な設定，2) 性能の低下，3) 原因が特定不能，という 3 つの問題を解決した．検知システムの複雑な設定という問題に対して，検知する障害と機器構成を分離して記述する方法を提案した．機器構成は，記述に CIM を用い CIM のデータベースに保存した．次に，定期的な検知処理による通信や計算の性能の低下という問題に対して，OS 等からのエラー報告を引き金に検知処理を実行する方法を提案した．最後に，障害の原因が特定不能という問題に対して，障害間に関係を持たせることで検知を防ぐことを提案した．上記 3 つの問題に対して論文 10) において同様の提案をした．しかし，既提案では検査方法の設定が複雑になるという問題が生じた．そのため，検査方法の設定がより直感的になるように，機器と検査方法を関連付けてデータベースに保存することを新たに提案した．検査方法と，機器と検査方法の関連を記述するために新しく CIM クラスを追加した．

今後の課題として，障害間の関係を自動生成する機能の実装が完成していないので，その機能を実装する．また，現在の設定ではデータベースの作成が単純とはいえ膨大な作業量になる．提案した障害検知システムを実用化するためこれを一部自動化させる必要がある．

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) : Nagios, http://www.nagios.org/.

2) : SNMP, http://www.ietf.org/rfc/rfc1157.txt.

3) 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).
4) DMTF: Common Information Model (CIM) Standards, `http://www.dmtf.org/standards/cim/`.
5) DMTF: CIM Qeuery Language Specification, `http://www.dmtf.org/standards/published_documents/DSP0202.pdf` (2004).
6) IBM: IBM Tivoli Monitoring. `http://www-306.ibm.com/software/tivoli/products/monitor/`.
7) Intel, Hewlett-Packrd, NEC and Dell: Intelligent Platform Management Interface Specification Second Generation v2.0 (2004).
8) Tierney, B., Crowley, B., Gunter, D., Holding, M., Lee, J. and Thompson, M.: A Monitoring Sensor Management System for Grid Environments, *HPDC*, pp.97–104 (2000).
9) Yu, D. and Baker, R.: GridMonitor: Integration of Massive Facility Fabric Monitoring with Meta Data Service in Grid Environment (2002).
10) 酒井将人, 石川 裕：クラスタ監視機能付き MPI 通信ライブラリ, HPC103, HPC (2005).
11) 田中 智, 宇和田弘美：CIM 知識ベースを活用したシステム運用管理のインテリジェント化, 技術創発3, NRI (2004).