データ分散化とオブジェクト再構築に基づく分散処理システムの高信頼化方式

齊藤雅彦[†] 村田悟[†]
島田優[††] 横山孝典[†]

オブジェクト指向分散処理システムの高信頼性を達成するに当たっては，同一処理を実行するオブジェクト（レプリカ）を並列に動作させ，計算機故障に備える方法を用いることが多い．しかし，この方式では，複数個のレプリカの存在，および，レプリカ間通信によって，通常時オーバヘッドが大きくなるという問題点がある．本稿では，オーバヘッドをできる限り小さくして高信頼性を達成するLazy Fault Toleranceを提案，評価する．本方式は，基本的には，データ冗長化に基づく高信頼化方式であるが，オブジェクトの所有データが他オブジェクトに固有のデータであれば，データを該当オブジェクトとの間で冗長所有するという特徴を有する．計算機故障が発生すると，各オブジェクトに分散されたデータを収集して，故障計算機で動作していたオブジェクトを別計算機上に再構築する．特に，システム内のハードウェア/ソフトウェア情報を纏めて制御するシステム管理オブジェクトに対しては，分散処理の実行過程で，データを各計算機に分散配置することが可能である．このため，本方式では，再構築時間が増大するものの，極めて小さな通常時オーバヘッドでオブジェクトの高信頼性を実現できる．

Method for Fault Tolerant Distributed Systems Based on Data Distribution and Object Reconstruction

MASAHIKO SAITO,[†] SATORU MURATA,[†] MASARU SHIMADA[††] and TAKANORI YOKOYAMA[†]

We present a new method called Lazy Fault Tolerance for refining the fault tolerance of object-oriented distributed systems. Lazy Fault Tolerance uses data redundancy instead of object redundancy. For system management objects, the data are distributed over all computers when a system management object requests computers to act locally. The system management object will restore its data by gathering all information from all computers if the computer on which it operates fails. Thus, Lazy Fault Tolerance does not need replicas which are commonly used to achieve fault tolerance of object-oriented distributed systems. Using our method, we can greatly reduce the performance degradation caused by the execution of replicas, although it takes a longer time to restore the system when a computer fails.

1. はじめに

パソコン，ワークステーションの高性能，低価格化を背景にした分散処理システムの普及が著しい．次世代パソコンOS（Operating System）の登場やインタフェースの標準化，オブジェクト指向プログラミングの導入により，これらの小型計算機を主構成要素とする分散処理システムが，今後の分散処理の主流になると考える．

分散処理システムは電力系統制御，列車制御等の情報制御分野にも適用されつつある．このような分野では，特に小型計算機を構成要素とする分散処理システムにおいて，システムの高信頼性が重要な特徴となる．構成要素である計算機の故障・停止が多い半面，複数の低価格計算機を組み合せてシステムを構築し，より信頼性の高い計算機システムを実現できるという期待がある．すなわち，分散処理システム内のいずれかの計算機が故障・停止しても，残った計算機で縮退運転を行えることが，分散処理導入の動機の一つとなる．しかし，縮退運転を行うためには，故障した計算機上で動作していたオブジェクトを他の計算機上に移行させ，互いに協調動作していたオブジェクトに故障の影

† （株）日立製作所 日立研究所
Hitachi Research Laboratory, Hitachi, Ltd.
†† （株）日立製作所 知的所有権本部
Intellectual Property Office, Hitachi, Ltd.

響を与えないようにする必要がある．特に，分散処理システム内のハードウェア/ソフトウェア情報を纏めて管理するシステム管理オブジェクトを特定の計算機上に配置することが多く，この計算機が故障・停止すれば，システム全体に影響が及ぶことになる．

上記の問題に対処するため，一般に，重要なオブジェクトに対しては他計算機上にオブジェクトの複製（レプリカ）を作成して動作させる方法が用いられる．本体オブジェクトの動作する計算機が故障・停止した場合には，レプリカにその処理を継続させる．この方式を用いると，故障時の復旧処理は容易かつ高速に行うことができるが，レプリカの存在およびレプリカ間通信によって，通常時（故障が発生していない状態）のオーバヘッドが大きくなるといった問題がある．

このため，本研究では，特に信頼性の影響が大きいシステム管理オブジェクトを対象とし，計算機故障からの復旧には時間を要するが，通常時の速度低下をできる限り防ぐ Lazy Fault Tolerance を考案，評価した．Lazy Fault Tolerance はデータを分散配置することによってオブジェクトを高信頼化させる方法であり，計算機故障・停止が発生した状況で，冗長化されたデータを収集してオブジェクトを再構築する．Lazy Fault Tolerance を用いると，オブジェクトのレプリカを用いずに，極めて小さな通常時オーバヘッドで高信頼性を達成することができる．

2. 対象システム

本研究では，分散処理システムの高信頼性を達成するに当たって，以下の二つの仮定を置く．

Fail-Silent 性 分散処理システムの各計算機は Fail-Silent である[1]．複数の計算機が協調動作している環境において，一つの計算機故障が他の計算機，ネットワークデバイスに影響を及ぼすことを未然に防ぐことは難しく，完全な Fail-Silent 性の実現は容易ではない．しかし，ここでは，計算機自体の 2 重化，通信部分の 2 重化等によって，各計算機が故障を十分隠蔽でき，故障の他者への影響を無視できると仮定する．

オブジェクト指向分散処理 対象となるシステムはオブジェクト指向に基づく分散処理システムである．複数の計算機上で複数のアプリケーションオブジェクトが互いにメッセージを送受して協調動作を行う．従来の情報制御分野では，計算機間で共有データを有し，これに基づいて制御を行うこともあったが，拡張性・保守性に乏しく，将来的にはオブジェクト指向を取り入れた分散処理システムが広く採用されていくものと考える．

我々は，既に，拡張性・保守性に優れた分散処理システムを構築することを目的として，UNIX☆等の標準プラットホーム上に統一的なオブジェクト指向インタフェースを提供する基本ソフトウェア・システム nOR（network-wide Object-based Reflective architecture）を開発している[2)~4)]．本研究では，システム nOR においてオブジェクトの高信頼性を実現することを目的とした．

システム nOR は，複数のオブジェクトを纏めてグループ（これを「世界」と呼ぶ）化し，これを別の世界のオブジェクトとして抽象化する多重世界モデル[2),4)]に基づいて構築した基本ソフトウェアである．すなわち，オブジェクト指向における拡張性・保守性と階層型システムにおける管理の容易さとを併せ持つシステムとなっている．多重世界モデルを支援するため，システム nOR では，処理と管理を分離して扱うことができるリフレクティブアーキテクチャ[5)]を導入した．リフレクティブアーキテクチャは，ソフトウェアを，アプリケーション本来の処理を実行するベースレベルと，ベースレベルのプログラムを管理するメタレベルに分離する．

図 1 にシステム nOR のハードウェア/ソフトウェア構成を示した．複数の計算機から成るサブネット☆☆が複数接続されたハードウェア構成を有する．このような分散処理システム上で動作するアプリケーションオブジェクトを，メタレベルに属する以下の 4 種類のソフトウェアが支援する．

(a) メタオブジェクト（meta-object）：オブジェクトと 1 対 1 に対応し，各々のオブジェクトのエラー処理，オブジェクト間通信の仲介，バージョンアップ等の処理を行う．

(b) 計算機管理オブジェクト（computer manager）：個々の計算機上のハードウェア管理，オブジェクト生成・終了管理を行う．分散処理 OS の一部であると見なすこともできる．

(c) ネットワーク管理オブジェクト（network manager）：計算機管理オブジェクトの提供する情報を基に，サブネット内のオブジェクト生成・終了管理，名称管理，負荷分散を行う．

(d) 世界管理オブジェクト（world manager）：論理的グループである「世界」とサブネットとの対応を管理し，世界内でのオブジェクト生成・終了管理，名称

☆ UNIX は X/Open カンパニリミテッドがライセンスしている米国並びに他の国における登録商標です．

☆☆ ここでは，互いにブロードキャストを行える範囲内にある計算機群をサブネットと定義する．

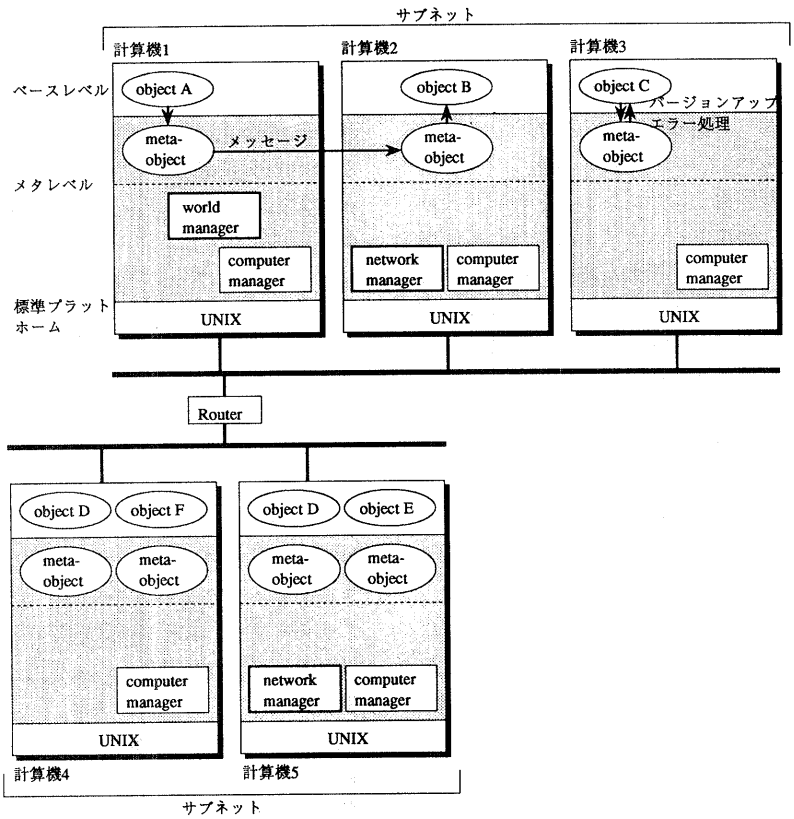


図1 分散処理基本ソフトウェア・システム nOR
Fig. 1 Distributed system nOR.

管理を行う.

このようなシステム構成においては，アプリケーションオブジェクト以上に，ネットワーク管理オブジェクト，世界管理オブジェクトといったシステム全体を管理するソフトウェア☆の高信頼化も重要な課題となる.

3. 従来の高信頼化方式

このようなシステムの高信頼性を達成する方法として，重要なオブジェクトに対して，複数の計算機上にオブジェクトの複製（レプリカ）を作成して動作させる方法が，オブジェクト指向分散処理では，一般的である[7]. この方式は Delta-4[8)~11)]，知的分散システム[12)]等で用いられている. レプリカを用いると，本体およびレプリカの動作している計算機いずれもが停止しない限り，プログラムの処理を継続することができる. また，レプリカは本体と全く同一の処理，データを保持するため，本体停止時の復旧を高速に行うことができる. しかし，この方法は以下の問題点を有する.

(a) オブジェクトの本体およびレプリカの動作する計算機がいずれも故障・停止した場合には，その処理が中断する. これに対処するためには，複数のレプリカを異なる計算機上に配置する必要があり，下記 (b)(c) の問題点を一層深刻にさせる.

(b) レプリカの処理自体が計算機負荷を増大させる.

(c) 常に本体とレプリカとの間でデータの一貫性を保つ必要がある. Delta-4 で行われている分類[11)]に従うと，本体とレプリカとの間でのデータ一貫性保証の方法として，本体とレプリカをほぼ完全に同期して動作させる方法（Active Replication）[10),13)]と，一定間隔ごとに本体からレプリカにデータを転送する方法（Passive Replication）[8),14),15)]がある. いずれの方法に関しても以下のような一貫性保証に関する問題点が存在する.

Active Replication 一定間隔ごとに待合せを

☆ ネットワーク管理オブジェクト，世界管理オブジェクトのように，システム内のソフトウェア/ハードウェア情報を纏めて，システム全体の管理を行うソフトウェアをシステム管理オブジェクトと呼ぶ. 例えば，DCE (Distributed Computing Environment)[6)]では，CDS (Cell Directory System) 等のソフトウェアがこれに相当する.

行って本体とレプリカ間の同期を保証する必要がある．特に，各々が動作する計算機の速度が異なる場合，一方の処理速度が他方に対する隘路となる．

Passive Replication 本体からレプリカにデータを転送する処理が通常時の計算機負荷を増大させる．特に，複数のレプリカを使用する場合に，一貫性保証処理が非常に複雑となる．

すなわち，レプリカを用いる方式は基本的に故障からの早期復旧を目的とした高信頼化方式であり，この代償として，通常時の実行速度を犠牲にしているとも言える．しかしながら，対象とする分散処理システムによっては，通常時の速度低下をできる限り防ぐことが高信頼化における重要な課題となる．本研究では，このような観点に基づいてオブジェクトの高信頼化を達成する Lazy Fault Tolerance を考案，評価した．

Lazy Fault Tolerance は，計算機故障からの復旧にはある程度の時間を要するが，通常時の速度低下をできる限り防ぎ，また，複数の計算機の同時停止にある程度対処できる高信頼化方式である．本方式は，分散配置されたデータからオブジェクトを再構築する再構築型故障回復を主体とするが，それを補助する方式として，故障認識の主体を分散処理システム内で巡回させる巡回型故障認識を併せ持つ．巡回型故障認識は，分散処理システム内のいかなる1個以上の計算機故障の組み合せも認識可能とする故障認識方式であり，複数の計算機の同時停止からの復旧を支援する．再構築型故障回復は，レプリカを使用する代わりにオブジェクトの所有データを分散配置して冗長化させる高信頼化方式であり，特に，システム内のソフトウェア/ハードウェア情報を纏めて管理するシステム管理オブジェクトに有効である．プロトタイプによる性能評価では，通常時オーバヘッドを3%以下に抑えつつ，システム管理オブジェクトの信頼性を高めることが可能となった．

4. 巡回型故障認識

パソコン，ワークステーション等を構成要素とする分散処理システムでは，(a) システム内のいずれの計算機も同程度の確率で故障する，(b) 人為的な計算機停止も考慮すれば二つ以上の計算機の同時停止の確率も無視できない，という前提を設ける必要がある．このため，Lazy Fault Tolerance では，分散処理システム内で故障認識の主体を時間経過と共に巡回させ，いかなる計算機の故障・停止も認識できる巡回型故障認識をその補助機能の一つとして導入した．

巡回型故障認識は，基本的には，各々の計算機に存在する計算機管理オブジェクトのうち，マスタとなる計算機管理オブジェクトが，システム内の他の計算機（スレーブ）に対してマルチキャストを行い，これに対する返答を確認することにより，故障認識を行う方法である．分散処理システムが複数のサブネットから構成される場合，故障認識はサブネット単位で行う．スレーブ計算機から一定時間経過しても応答がなければ，マスタは計算機故障であると判断する．時間経過に従って，マスタとなる権利を分散処理システム内で巡回させることにより，マスタ計算機の故障にも対処できるようにする．以下，巡回型故障認識の動作の流れを説明する．

1. 初 期 設 定

各計算機管理オブジェクトは変数 NextTime によって故障確認を行う時刻を管理する．サブネット内の計算機数を N 台とすると，各計算機に $0\sim(N-1)$ の範囲で互いに異なる整数値を与え，これを計算機番号 num とする．現時刻を TimeNow，故障確認間隔の最小単位を ΔT とすると，以下のように，次に自己がマスタになる時間を暫定的に算出する．

$$\mathrm{NextTime} = \mathrm{TimeNow} + (\mathrm{num}+1)^{*}\Delta T$$

2. 故 障 確 認

各計算機管理オブジェクトは，現時刻 TimeNow が以下の条件を満たすならば，自己がマスタであると判断し，故障確認メッセージをマルチキャストする．

$$\mathrm{NextTime} \leq \mathrm{TimeNow} < \mathrm{NextTime} + \Delta T$$

これに対して各スレーブは自己の存在している計算機が生存していることを通知する．マスタはサブネット内に存在するすべての計算機から生存通知が届くか，一定時間が経過するまで待つ．返答が届かなかった計算機に対しては，再度故障確認メッセージを送信し，再び返答がなければ，故障と判断する．

3. 故障確認時刻管理

各計算機管理オブジェクトは，故障確認メッセージの送受信時に，マスタと自己との論理的距離から，自己が次にマスタになるべき時刻を決定する．ここで，マスタとの論理的距離 dist は，自己の計算機番号（num），故障確認メッセージで送られてきたマスタの計算機番号（master）から，以下のように計算する．

マスタ　：$\mathrm{dist} = N$

スレーブ：$\mathrm{num} > \mathrm{master}$ のとき

$$\mathrm{dist} = \mathrm{num} - \mathrm{master}\,[\mathrm{mod}\,N]$$

$\mathrm{num} < \mathrm{master}$ のとき

$$\mathrm{dist} = \mathrm{num} - \mathrm{master} + N\,[\mathrm{mod}\,N]$$

これにより，サブネット内計算機は各々互いに異なる論理的距離を導出できる．各々の計算機管理オブジェ

クトは，マスタとの論理的距離と，故障確認間隔の最小単位（ΔT），現在時刻（TimeNow）から，以下の計算式により，自己が次にマスタになるべき時刻を決定する．

$$\text{NextTime} = \text{TimeNow} + \text{dist}^{*}\Delta T$$

この時刻は故障確認メッセージを送受信するごとに再計算する．すなわち，マスタが故障確認メッセージをマルチキャストしてから，変数NextTimeは以下のように定期的に再計算される．

$$\text{NextTime} - \text{TimeNow} = N^{*}\Delta T, (N-1)^{*}\Delta T, \ldots, 2^{*}\Delta T, \Delta T$$

これによって，計算機（num − 1）からの故障確認メッセージを受信してから ΔT 時間後に計算機の故障確認を行うことができる．計算機（num − 2）が故障確認を行った時点で，$2^{*}\Delta T$ 時間後に計算機の故障確認を行うことを計算しているため，仮に，計算機（num − 1）が故障した場合でも，計算機の故障確認を続行することができる．

以上に示す巡回型故障認識を用いることにより，仮に次回マスタになる予定であった計算機が停止しても，計算機それぞれがマスタになる時刻を記憶しているため，故障認識自体が行われなくなることはない．すなわち，停止する計算機のいかんにかかわらず，計算機停止を認識することができ，また，マスタを含む複数の計算機の同時停止にも対処することができる．

5. 再構築型故障回復

5.1 通常時処理

Lazy Fault Toleranceはオブジェクトを冗長化させるのではなく，データを冗長化させることによって高信頼化を実現する方法である．通常時にプログラムの所有データを分散配置して冗長化し，故障が発生すると冗長化されたデータからオブジェクトを再構築する．本方式は，オブジェクトの所有データが次に示す自己固有データ，他者固有データのいずれかに属し，かつ，システム管理オブジェクトにとっては，所有データの大半が他者固有データであることに基づく方法である．

自己固有データ オブジェクトが自ら設定，変更するデータ．

他者固有データ 他のオブジェクトが設定，変更するデータであって，通信等によって入手したデータ．なお，他者固有データには，計算機性能，ネットワークアドレス等のように，いずれかの計算機に固有の値も含まれるものとする．これを特に，計算機固有データと呼ぶ．ある計算機が故障・停止した場合，その計算機固有データはシステム管理にとって不要となる．

一般に，アプリケーションオブジェクトの所有データは自己固有データに，システム管理オブジェクトの所有データは他者固有データに分類される．例えば，システムnORでは，ネットワーク管理オブジェクトは計算機性能，ネットワークアドレス等の計算機固有データと，個々の計算機上で動作しているオブジェクトの管理データ（オブジェクトの名称，通信アドレス，動作状態等）を各計算機管理オブジェクトから入手し，サブネット内でのオブジェクト生成・終了管理，名称管理，負荷分散等を行う．同様に，世界管理オブジェクトはサブネット内計算機の総合性能，サブネット内オブジェクトの管理データ等を各ネットワーク管理オブジェクトから入手し，サブネット間に跨るオブジェクト生成・終了管理，名称管理等を行う．

オブジェクトの所有データが他者固有データである場合，データの本来帰属するオブジェクトとの間でデータを冗長化させることができる．すなわち，ネットワーク管理オブジェクトの管理データは計算機管理オブジェクトとの間で冗長所有され，世界管理オブジェクトの管理データはネットワーク管理オブジェクトとの間で冗長化できる．

この冗長化の利点は，分散処理の実行過程でデータを分散配置できることである．例えば，**図2**に，ネットワーク管理オブジェクトが新たなオブジェクトを生成する過程を示した．ネットワーク管理オブジェクトは，各計算機の性能，負荷を考慮してオブジェクト生成先の計算機を決定し，その計算機管理オブジェクトに対してオブジェクト生成を依頼する．計算機管理オブジェクトは，メタオブジェクトを起動し（メタオブジェクトがオブジェクトを起動する），名称，通信アドレス，動作状態等から成るオブジェクト管理データを記憶する．この後，オブジェクト管理データをオブジェクト生成完了メッセージと共にネットワーク管理オブジェクトに通知すれば，オブジェクト管理データの冗長化が達成される．これによる通常時オーバヘッドはほぼ無に等しい．

さらに，この場合，計算機管理オブジェクトとネットワーク管理オブジェクトとの間で，データの一貫性は自動的に保証される．次節で述べるように，計算機故障時には，ネットワーク管理オブジェクトは計算機管理オブジェクトからデータを収集して管理データを再構築する．したがって，計算機管理オブジェクトの所有データがネットワーク管理オブジェクトの所有データよりも常に新しいか等しいことが保証されていれば，正しくネットワーク管理オブジェクトを再構築できる．そして，この条件は，オブジェクト管理デー

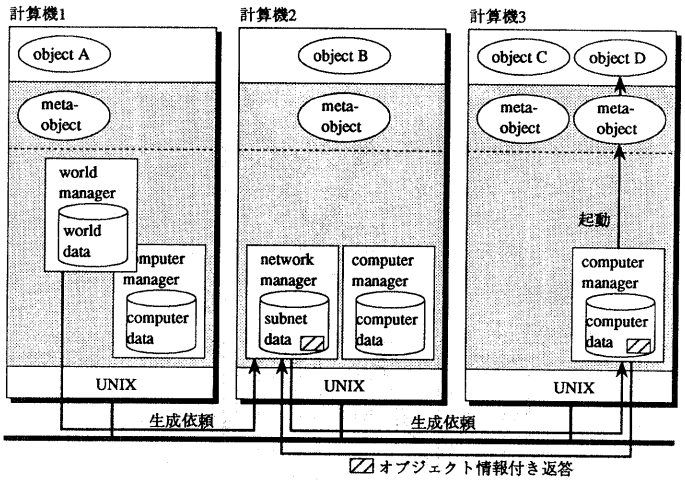


図2 システム管理オブジェクトのデータ冗長化
Fig. 2 Data redundancy of system management object.

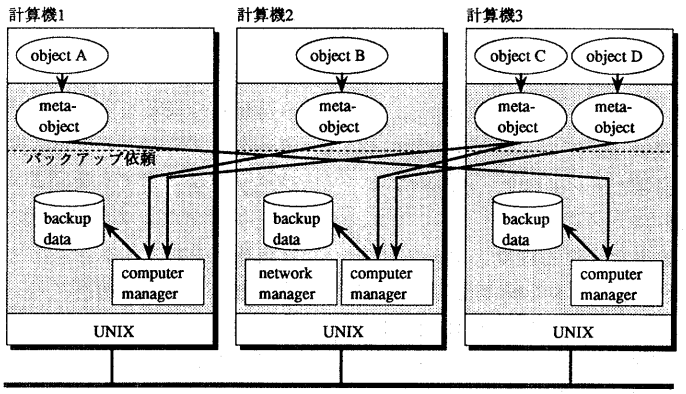


図3 アプリケーションオブジェクトのデータ冗長化
Fig. 3 Data redundancy of objects.

タをオブジェクト生成完了メッセージと共に通知することによって達成されている．

オブジェクトの所有データが自己固有データである場合，他者固有データと同様の冗長化は不可能であるため，Primary Backup[16]を用いて冗長化を施す．ただし，高信頼化の対象がアプリケーションオブジェクトである場合，図3に示すように，メタオブジェクトが強制的にオブジェクトの所有データを他計算機上に保存する．この際，アプリケーションプログラマは定期的にバックアップを行ってほしいデータ，およびバックアップ間隔を指定できる．メタオブジェクトは指定された間隔ごとにオブジェクトの内部データを取り出し，他計算機の計算機管理オブジェクトにバックアップを依頼する．同様に，メタオブジェクトを介して送受信されるメッセージもバックアップデータとして格納される．

なお，システム管理オブジェクトの所有データにも，ネットワークの遅延時間，サブネット内の計算機数等，自己固有データとなる管理データが存在する．しかし，これらのデータは，一般に，システム構成が変更されない限り変化しないデータであり，バックアップによる性能低下を無視することができる．

5.2 故障時処理

分散処理システム内の計算機が故障・停止した場合には，前節で述べた冗長データに基づいて，停止した計算機上のオブジェクトを他計算機上に再構築する．

システム管理オブジェクトの動作している計算機が停止した場合には，冗長化された管理データをすべて収集することによって，システム管理オブジェクトを再構築することができる．図4にネットワーク管理オブジェクトの再構築の手順を示す．なお，世界管理オブジェクトの再構築も同様に行うことが可能である．

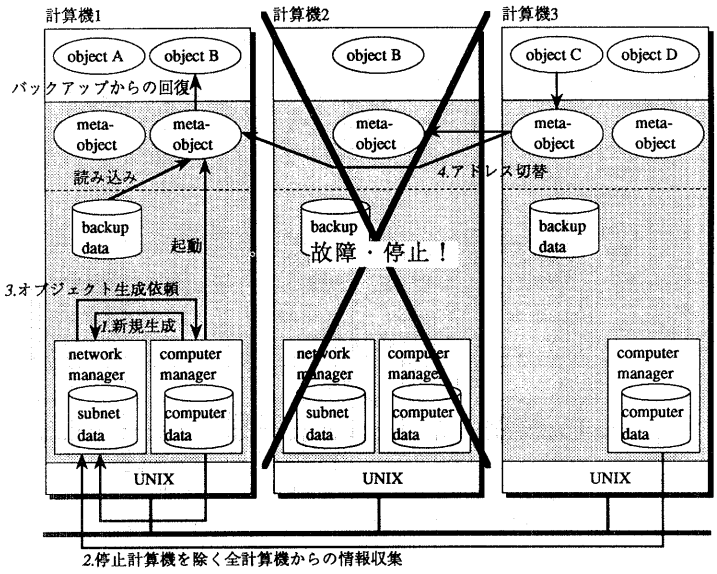


図4 システム管理オブジェクトの再構築
Fig. 4 Reconstruction of system management object.

1. ネットワーク管理オブジェクトの再起動

故障を認識した計算機管理オブジェクトは自己の計算機上にネットワーク管理オブジェクトを再起動する．

2. 管理データの収集

新たに起動されたネットワーク管理オブジェクトはシステム構成に即した管理データを構築しなければならない．ネットワーク管理オブジェクトは自己のサブネット内の各計算機管理オブジェクトに要求して，個々の計算機に関する情報を入手する．

ここで，停止した計算機は以降システム内に存在しないものとして扱われるので，停止計算機からハードウェア情報を収集できなくても問題はない．

3. オブジェクト再起動

新たに生成されたネットワーク管理オブジェクトは，停止した計算機上で動作していたオブジェクトを他計算機上に再起動させる．停止計算機でネットワーク管理オブジェクトと同時に停止したオブジェクトの情報は，各計算機管理オブジェクトからバックアップデータに関する情報を収集することによって入手できる．例えば，アプリケーションのバックアップを図3のように行っていた場合，計算機1から収集したバックアップデータの情報により，計算機2（停止した計算機）上で，オブジェクトBが動作していたことが判る．この情報により，計算機1上でオブジェクトBを再起動させることができる．データを収集したネットワーク管理オブジェクトは，バックアップデータを保持する計算機管理オブジェクトに対して，オブジェクト再起動の指示を出す．計算機管理オブジェクトは指定メタオブジェクトを再起動し，メタオブジェクトにバックアップデータを通知する．

4. クライアント側の処理

再構築されたオブジェクトは，計算機故障前と異なる通信アドレスを所有する．このオブジェクトに対して，別のオブジェクトが通信しようとする場合，通信エラーが発生する．この場合，送信側メタオブジェクトは，ネットワーク管理オブジェクトに受信側オブジェクトの新しい情報を尋ね，その通信アドレスを入手する．これによって，メタオブジェクトはアプリケーションに受信側オブジェクトの再構築を認識させることなく，メッセージ送信を再開することができる．

Lazy Fault Tolerance の特徴として，必要とするデータを各計算機から収集してネットワーク管理オブジェクトを再構築するため，1台以上のいかなる計算機が停止した場合でも，その状況に応じて，ネットワーク管理オブジェクトを再構築できることが挙げられる．ただし，アプリケーションオブジェクトの情報をバックアップデータから入手しているため，オブジェクトおよびそのバックアップデータが存在する計算機いずれもが停止した場合，オブジェクトを再起動させることはできない．

6. 性能評価

Lazy Fault Tolerance は，レプリカを用いる方式と異なり，分散処理実行の過程でデータの冗長化を行う

表1 性能評価環境
Table 1 Condition of evaluation.

システム構成	HP9000/710×2台 + HP9000/755×1台
ネットワーク	EtherNet（上記計算機以外に，約200台の計算機がこのネットワークに接続されている）
基本ソフトウェア	HP-UX
通信プリミティブ	NCS (Network Computing System) のRPC
アプリケーション	電力系統需給予測および周波数制御の一部とその表示部
オブジェクト生成に伴い転送されるオブジェクト管理データの内容	名称，ID，通信アドレス，オブジェクトの存在する計算機の情報，動作状態，関連するネットワークおよび世界の情報，実行ファイルの位置，バージョン，機能（メソッド）等 計228バイト + （機能数） × 20バイト NCSのRPCによる平均転送時間20.52ミリ秒
アプリケーションのバックアップの指定	メッセージのみ自動バックアップ
巡回故障認識の間隔	10秒

表2 オーバヘッドの評価
Table 2 Evaluation of overhead.

オーバヘッド	Lazy Fault Tolerance*		Primary Backup
	オブジェクト生成	オブジェクト終了	
自計算機	0.42 ms:2.82% (14.90 ms)	0.10 ms:2.01% (4.98 ms)	—
他計算機	0.49 ms:0.41% (119.15 ms)	0.12 ms:0.30% (39.94 ms)	25.30 ms**

*( ) 内はオブジェクト生成/終了の全処理時間
**管理データを別計算機上に保存するために必要な時間

表3 再構築時間の評価
Table 3 Evaluation of the reconstruction time.

故障認識&再構築時間	故障認識時間			再構築時間	計
	ロスタイム $\Delta T = 10$ s	1回目タイムアウト	2回目タイムアウト		
best case	0	5 s	10 s	4.55 s	19.55 s
worst case	20 s	5 s	10 s	4.55 s	39.55 s

ことができるため，通常時オーバヘッドを極めて小さな値とすることができる．この反面，故障時には，複数のオブジェクトからデータを収集してシステム管理オブジェクトを再構築するため，故障回復に長い時間を要する．本研究では，どの程度の通常時オーバヘッドでデータ冗長化を行えるか，並びに，どの程度の再構築時間でシステム管理オブジェクトを復旧させることができるかを測定した．

性能測定では，3台のワークステーションから成るプロトタイプを構成し，評価を行った．**表1**に評価を行ったシステムの動作環境を示す．

表2にネットワーク管理オブジェクトにおける，Lazy Fault Tolerance適用によるオーバヘッドを示した．ここでは，他計算機へオブジェクト生成/終了を依頼する場合，および，自計算機上でオブジェクト生成/終了を行う場合について測定し，通常時オーバヘッドとその処理時間に占める割合とを示している．自己の計算機上で処理を行う場合には計算機間通信が存在しないため，比較的オーバヘッドが大きくなるが，それでも，3%以内に抑えられている．他計算機に処理を依頼する場合には，オーバヘッドは0.5%以下であり，いずれも無視できる範囲であると考える．

また，同様にデータを冗長化して高信頼化を達成する方式であるが，アプリケーションの高信頼化に用いられているPrimary Backupを，ネットワーク管理オブジェクトに適用した場合の性能測定結果も表2に示した．Lazy Fault Toleranceのオーバヘッドは，分散処理の実行過程でデータを分散配置できるという利点により，極めて小さくなることが確認できる．

表3には，ネットワーク管理オブジェクトを他計算

機上に再構築するための時間を示した．これは，ネットワーク管理オブジェクトの動作している計算機の電源を停止させ，残り2台で縮退運転できるようになるまでの時間を示している．故障認識完了後，実際に再構築が終了するまでの時間が4.55秒であり，たとえ故障認識時間を0とすることができても，この間ネットワーク管理オブジェクトの処理が中断する．参考として，ネットワーク管理オブジェクトにPrimary Backupを適用した場合の故障回復時間を測定したが，これは平均0.35秒となり，故障回復時間の長さがLazy Fault Toleranceの問題点であることが理解できる．

データを各計算機から収集/統合し，また，整合性チェックを行う過程が故障回復処理の大半を占める．このため，ネットワーク管理オブジェクトの再構築時間は基本的に計算機数に比例するが，計算機数が非常に多い場合，ネットワーク競合により再構築時間がさらに長くなると考える．なお，将来的には，再構築処理を最適化/並列化して高速化する予定であるが，この場合には，通信プリミティブのNCS[17]RPCが約250バイトのオブジェクト管理データを転送するのに20ミリ秒程度の遅延時間を有すること，および，メタオブジェクトをUNIXのプロセスで構成していることなどが隘路となると考える．

本プロトタイプの一つの欠点として，故障認識処理自体が再構築処理に比べて3～7倍の時間を要していることが挙げられる．これは，通信ネットワークとしてEtherNetを使用していることが主たる要因である．EtherNetの不確実性のため，故障が発生したことを15秒のタイムアウト（5秒+10秒の2回のタイムアウト）で認識している．また，巡回型故障認識で故障認識間隔を10秒とすると，マスタ計算機が故障・停止した場合には最悪20秒のロスタイムが発生する（2台以上の計算機の同時停止を考慮すると，さらにロスタイムが大きくなる可能性がある）．このため，プロトタイプでは，Lazy Fault Toleranceは故障発生から復旧までに約20～40秒の時間を要する方式であることが確認された．故障回復時間が重要視されるプログラムに対しては，レプリカ等を使用して復旧を高速化する必要がある．

将来的には，故障回復時間を短縮する上で隘路となる故障認識のハードウェア化，故障認識における通信のEtherNetからの分離，高速な通信プリミティブの使用，および，メタオブジェクトのLight Weight Process化等を図り，情報制御分野での実用に耐えるシステムにする必要がある．

7. おわりに

本研究では，計算機故障からの復旧にはある程度の時間を要するが，通常時の速度低下をできる限り防ぐLazy Fault Toleranceを考案，評価した．本方式は，データの分散配置によってオブジェクトを高信頼化させる方法であり，計算機故障という異常が発生した状況で，冗長化されたデータに基づき，オブジェクトを再構築する．本方式は，システム管理オブジェクトの高信頼化に特に適した方法である．

Lazy Fault Toleranceは通常時に行う処理をできる限り省略しているため，通常時のオーバヘッドを極めて小さな値とすることができる．プロトタイプでは通常時オーバヘッドを3%以下に抑えることが可能であった．しかし，システム管理オブジェクトの動作している計算機が故障した場合には，必要とする管理データを分散処理システム内の各オブジェクトから収集するため，故障復旧に時間を要する．この復旧時間は3台構成の分散処理システムで4.5秒程度であった．

謝辞 本研究に対して，多くの助言，協力をいただいた当社日立研究所の武石春海氏，佐川朋子氏，上脇正氏，納谷英光氏，恒冨邦彦氏，University of Illinois at Urbana-Champaign, Open Systems Laboratoryの Daniel C. Sturman 氏，Wooyong Kim 氏に深謝いたします．

参考文献

1) Schneider, F.B.: Byzantine Generals in Action: Implementing Fail-stop Processors, *ACM Trans. on Computer Systems*, Vol.2, No.2, pp.145–154 (1984).
2) 横山孝典ほか：リフレクティブアーキテクチャに基づく分散処理環境(1)—分散処理モデル—，第43回情報処理学会全国大会論文集，Vol.5, pp.159–160 (1991).
3) 上脇 正ほか：リフレクティブアーキテクチャに基づく分散処理環境(2)—メタシステム—，第43回情報処理学会全国大会論文集，Vol.5, pp.161–162 (1991).
4) 齊藤雅彦ほか：リフレクティブアーキテクチャに基づく分散処理環境(3)—オブジェクト管理と通信管理—，第43回情報処理学会全国大会論文集，Vol.5, pp.163–164 (1991).
5) Maes, P.: Concepts and Experiments in Computational Reflection, *OOPSLA-88 Proceedings*, pp.147–155 (1988).
6) Open Software Foundation: *Introduction to OSF DCE*, Prentice Hall, Inc. (1992).
7) Birman, K.P. and Joseph, T.A.: *Exploiting*

Replication in Distributed Systems, Chapter 15 in Distributed Systems, Mullender, S. ed., pp.319–367, ACM Press, Addison-Wesley, New York (1989).
8) Speirs, N.A. and Barrett, P.A.: Using Passive Replications in Delta-4 to Provide Dependable Distributed Computing, *Proc. 19th Int'l Symp. Fault-Tolerant Computing Systems*, pp.184–190 (1989).
9) Barrett, P.A. et al.: The Delta-4 Extra Performance Architecture (XPA), *Proc. of the 20th Int'l Symp. Fault-Tolerant Computing Systems*, pp.481–488 (1990).
10) Chereque, M. et al.: Active Replication in Delta-4, *Proc. 22nd Int'l Conf. Fault-Tolerant Computing Systems*, pp.28–37 (1992).
11) Powell, D.: Distributed Fault Tolerance: Lessons from Delta-4, *IEEE Micro*, pp.36–47 (Feb. 1994).
12) 関俊 文ほか：オブジェクト指向分散システムにおける放送待機冗長処理方式，電気学会情報処理研究会資料，IP-94-1 (1994).
13) Ahamad, M. et al.: Fault Tolerant Computing in Object Based Distributed Operating Systems, *Proc. of the 6th Symposium on Reliable Distributed Systems*, pp.115–125 (1987).
14) Birman, K.P. et al.: Implementing Fault Tolerant Distributed Objects, *IEEE Trans. on Software Engineering*, Vol.11, No.6, pp.502–508 (1985).
15) Babaogle, O.: Fault Tolerant Computing Based on Mach, *ACM Operating Review*, Vol.24, No.1, pp.27–39 (1990).
16) Borg, A. et al.: A Message System Supporting Fault-Tolerance, *Proc. of the 9th ACM Symposium on Operating Systems Principles*, pp.90–99 (1983).
17) Zahn, L. et al.: *Network Computing Architecture*, Prentice Hall, Inc. (1990).

(平成6年8月19日受付)
(平成7年7月 7 日採録)

齊藤 雅彦（正会員）

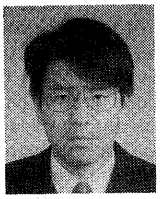
1964年京都生．1988年京都大学大学院工学研究科情報工学専攻修了．同年（株）日立製作所日立研究所入社．並列処理オペレーティングシステム，VLIWコンパイラ，分散処理プログラミング環境等の研究に従事．並列/分散処理アーキテクチャ，オブジェクト指向，リアルタイム処理等の分野に興味を持つ．電子情報通信学会会員．

村田 悟（正会員）

昭和41年生．平成3年京都大学大学院工学研究科修士課程修了．同年（株）日立製作所入社．オブジェクト指向分散環境の開発に携わった後，自動列車運転システムの開発に従事．大規模ソフトウェアの開発技法に興味をもつ．電気学会会員．

島田 優（正会員）

昭和35年生．昭和59年東北大学工学部通信工学科卒業．昭和61年同大学院工学研究科情報工学科前期課程修了．同年（株）日立製作所入社．現在，知的所有権本部に所属．



横山 孝典（正会員）

1959年生．1981年東北大学工学部通信工学科卒業．1983年同大学院工学研究科電気及通信工学専攻修士課程修了．同年（株）日立製作所日立研究所入社．1987年から1990年まで（財）新世代コンピュータ技術開発機構出向．ユーザインタフェース，人工知能，並列処理，分散処理，ソフトウェア工学の研究開発に従事．電子情報通信学会，IEEE，ACM各会員．