プロセスの2重化によるOSの高信頼化手法

岸本光弘[†] 中島淳[†,☆] 大橋勝之[†]
金沢裕治[†] 土屋芳浩[†] 今井祐二[†]

従来オペレーティングシステム (OS) の高信頼化は，試験やレビューを網羅的に行い内在するソフトのバグ等の故障を排除することで追求されてきた．しかしOSはその規模や複雑さが膨大なため，故障を完全に取り除くことは現実には不可能であった．一方，誤りの発生を内部で検出し回復する耐故障技術が知られており，いくつかの特殊用途のOSが耐故障技術に基づいて設計実現されている．しかし，耐故障技術をオープンなOSに適用するためには，様々な問題を解決しなければならない．本論文では，既存のオープンなOSの代表であるUNIXに対し，耐故障技術の1つである“プロセスペア方式”を適用する方法を検討した．誤りの検出，隔離，回復を実現するための故障管理機構を考案するとともに，既存プログラムの修正軽減のため，耐故障性を備えた通信機構と内部状態の安定格納機能をライブラリとして用意した．そして，実際にファイルシステムに対し耐故障性を付与する修正を行い，ハードウェアやソフトウェアに誤りが発生しても，ユーザに知られることなく回復できることを確認した．

Highly Available Operating System by Process Replication

Mitsuhiro Kishimoto,[†] Jun Nakajima,[†,☆] Katsuyuki Ohashi,[†]
Yuzi Kanazawa,[†] Yoshihiro Tsuchiya[†] and Yuji Imai[†]

In order to make an operating system highly available, careful design reviews and exhaustive tests are used to eliminate internal software faults. However, its quantity and complexity prevent these fault avoidance methods from finding all faults. Fault tolerant methods can detect and recover internal errors as they occur. A few fault tolerant Operating Systems were designed and developed. But they are propriety and require expensive special hardware. Applying fault tolerant methods to existent open operating systems raise many difficult issues. This paper describes how to apply the “process pair method”, one of the well-known fault tolerant methods, to an open operating system, UNIX. A fault management mechanism is developed to detect, isolate, and recover errors. In order to minimize modification in existent programs for fault tolerance, an enhanced communication library and an stable storage library are introduced. We have prototyped fault tolerant file system and demonstrated that it can mask hardware errors and software errors from users.

1. はじめに

オープンシステムの普及にともない，従来メインフレームとそのオペレーティング・システム (OS) 上に構築されてきた企業の基幹業務システムを，オープンシステム，特にUNIX[☆☆]上に構築する事例が増加している．しかし，UNIXはメインフレームのOSと比較すると，一般に1桁程度信頼性が低いといわれている[☆☆☆]．

信頼性の低さはUNIXが歴史的にエンジニアリング分野で成長してきたことに起因している．たとえば“lasy panic”と呼ばれるコードが存在する．lasy panicとは，予期せぬ資源不足のような対処が難しい事態がOS内部で発生した場合，安易にOS全体を異常終了してしまうものである．商用のUNIXでは網羅的なレビューや試験により，このような不適切なコードを除去し，設計やコーディングのバグ等の故障を排除する努力が払われている．これらは故障回避（フォールト・アボイダンス）と総称される技術である．しかし，

† 富士通研究所
Fujitsu Laboratories Ltd.
☆ 現在，SCO
Presently with The Santa Cruz Operation, Inc.

☆☆ UNIXはX/Openカンパニーリミテッドが独占的にライセンスしている米国ならびに他の国における登録商標です．

☆☆☆ 信頼性を計る尺度である可用性（＝運用時間/(運用時間＋停止時間)）でいえば，メインフレームのOSが99.9〜99.99%であるのに対し，現在の商用UNIXは99〜99.9%程度である[1]．

OSは複雑なシステムであり故障を根絶することは非常に難しい．特にオープンなOSは次々に新機能が追加され成長していくので，故障回避技術だけによる高信頼化努力には限界がある．

故障から誤りが発生した際に，誤りを内部で検出/隔離/回復して，発生した誤りを隠蔽する技術を耐故障技術（フォールト・トレランス技術）と呼ぶ．前述の故障回避技術に加え耐故障技術を適用することにより，信頼性の大幅な向上が期待できる．

しかし，広く普及しているオープンなOSは耐故障技術が適用しにくい構造を持っており，すでに存在する膨大なプログラムに耐故障性を付与する方法はまだ検討されていない．本論文は耐故障技術の1つである“プロセスペア方式”を，既存の代表的なオープンなOSであるUNIXに対して適用する方法について述べる．はじめに，高信頼OSの概要を説明する．次に故障管理，耐故障通信機構，内部状態引継ぎについて説明する．そして耐故障ファイルシステムの構築法とプロトタイプでの結果を示す．最後に考察および今後の課題を述べる．

2. 関連研究

耐故障技術には，ハードウェアで実現するものとソフトウェアで実現するものがある．ハードウェアによる耐故障技術では，CPUやI/O装置を多重化し，ハードウェアで発生した誤りをソフトウェアに見えなくする．実際にハードウェア・フォールト・トレラント・マシンがいくつか商品化されている☆．これらはOSの大幅修正が不要という利点があるが，逆に実際に障害の原因の大半を占めるソフトウェアの故障に対処することができない．

ハードウェアの耐故障技術に加え，ソフトウェアによる耐故障技術を実装したフォールト・トレラント・コンピュータ（FTC）が商品化されている☆☆．いくつかのFTCはプロセスペア方式により独自OSを耐故障化している[2),3)]．FTCは，99.999%以上の可用性や24時間連続運転，さらにハード，ソフトの活性増設，交換といった高度な機能を実現している．しかしFTCは，専用の高価な二重化ハードウェアを用い，耐故障性を念頭に置いて設計した独自のプロプライアタリOSが必要となるので，既存のオープンなOSに耐故障性を付与する際の参考にはならない．

UNIXをベースとしたソフトウェアによる耐故障OSとしてTARGON/32が知られている[4),5)]．TARGON/32はマイクロカーネル構成であり，OSをマイクロカーネルと複数のサーバに分割している．そして各サーバをプロセスペア方式を用いて耐故障化している．ハードウェアやマイクロカーネルで誤りが発生すると，サーバをバックアップに切り替えることで処理を継続する．TARGON/32はプロセスペア方式の1つである自動チェックポインティング方式を採用している．自動チェックポインティング方式はユーザプロセスを透過的にプロセスペア化し耐故障化も実現できるという利点がある．しかし更新データをページ単位で大量にチェックポイントする必要があり，プロセスペア間でアトミック3ウェイメッセージ交換による同期が必要である．さらにサーバが決定的に動作しなければならないので，サーバをマルチスレッド化することができない．またTARGON/32がカバーしている故障の範囲は，ハードウェア故障とマイクロカーネルの故障までであり，サーバの故障は含まれていない．マイクロカーネルのプログラム規模に比べ，UNIXサブシステムなどのサーバの合計のプログラム規模は約4倍なので，マスクできる範囲は数分の1にすぎない．

ISIS[6)]は，クライアントサーバ型のユーザプログラムにおいて，複数のサーバプロセスを用意し，耐故障性を備えた通信ライブラリを使用することで耐故障性を実現している．ユーザは複数のサーバプロセスの管理をISISに任せることができる．しかしISISはユーザプログラムを高信頼化するものなので本論文で解決しようとしているオープンなOSの高信頼化のためには利用できない．ユーザプログラム自身の故障にも対処できカバー範囲が広い反面，耐故障性を付与するためユーザプログラムの修正が必要である．

現用待機構成や相互待機構成の高信頼クラスタシステムも実績のある耐故障システムと考えることができる．クラスタシステムは独立した複数のコンピュータから構成されており，あるコンピュータで誤りが発生すると，別のコンピュータで業務を引き継ぐものである．クラスタシステムはシステム全体としては高信頼になっているが，ノードのレベルでは，クラッシュが発生し，短時間とはいえダウンが外部に露見してしまい比較的長い回復時間が必要となる．

GOLDRUSH[7)]は，Chorusマイクロカーネルを用いており，リスタート方式によりファイルシステムを高信頼化するシステムである．リスタート方式を用いることで，既存のOSを軽微な修正で高信頼化を実現している．しかしリスタート方式はディスク等の永続

☆ TANDEM社のIntegrity S4000シリーズ，Stratus社のContinuumシリーズなど

☆☆ Tandem社のHimalayaシリーズ，富士通のSUREシリーズなど．

記憶メディアが必要であり，故障を回復する機能単位の依存関係が簡単な場合しか利用できない．たとえばファイルシステムを耐故障化することはできるが，プロセス間通信機能の耐故障化では利用できない．

本論文では，ユーザプログラムの変更を行わずに高信頼化するため，OS層での高信頼化を実現する．耐故障性を考慮せずに開発され，広範囲に利用されているオープンなOSのすべての機能に適用可能で，かつ特別なハードウェアを必要としない方式を提案する．本論文で提案する方式により，拡張性や保守性を損なわずにわずかな修正でプロセスペア化することができ可用性が向上する．

3. 高信頼OSの概要

3.1 高信頼OSの設計目標

システム運用時の障害の大きな部分を占めるソフトウェアの故障による誤りを隠蔽するためには，ハードウェアの多重化ではなくソフトウェアによる耐故障技術を実装する必要がある．そのためにはOSの耐故障化が有効である．

(1) **可用性の向上**
プロセスペア方式を用いてOSを耐故障化し，OS内部で発生した誤りの95%以上をマスクすることで可用性を1桁以上向上させることを目標とする．

しかし既存のオープンなOSを耐故障化するためには，さらに以下の目標を達成しなければいけない．

(2) **ユーザプログラムは修正しない**
オープンなOSはその上で動作する膨大な数のユーザのプログラムを持っている．ユーザのプログラム修正やリコンパイルを必要としないOSの高信頼化が必須である．

そこでOSを修正して耐故障性を付与する際に，ユーザのプログラムとのインタフェースであるシステムコールは変更せず，既存のユーザプログラムをすべてそのまま実行できるようにする．

(3) **OSの保守性と拡張性の維持**
既存OSに対し耐故障技術を適応するにはOS自身のソースプログラムを修正する必要がある．しかしライブラリやコマンド部分を除いても，UNIXのソースプログラムは500Kステップ以上あるので，修正量を減らす工夫が必要である．また，プログラムは耐故障化されたあとも，継続的に保守され機能が追加されていく．プログラムの保守性や将来の拡張性を維持するために，既存の処理論理を変えずに耐故障化する必要がある．

そのため，すべての誤りを統一的に扱える故障管理機構を用意する．また完成度の高いOSのプログラムは，パラメータエラーや資源不足の処理のため良く構造化された処理論理になっている．そのため，耐故障化に必要な共通機能をライブラリとして提供すれば，既存処理論理を変更せずに耐故障化が可能である．

(4) **特殊なハードウェアを前提としない**
移植性の良さはオープンなOSの特徴の1つである．高信頼化が移植性の良さを損なってはいけない．

そのため，耐故障技術の実現に特別なハードウェアを前提とせず，耐故障化に必要な機能は原則としてソフトウェアで実現することにする．このとき，実行時の定常性能の低下を非常に小さく抑える必要がある．

3.2 マイクロカーネル構成

モノリシック構造のOSは，OS内のデータや関数を複雑に共有しているため，誤りが発生したとき，誤りの波及範囲を特定し，モジュールを隔離したり回復したりするのが簡単ではない．近代的なOSの中には，モジュール性を高め分散システムへの拡張が容易なマイクロカーネル構造を採用しているものがあり，UNIXにもいくつかのマイクロカーネル構造の実装がある[4),8),9)]．

マイクロカーネル構造のOSでは，マイクロカーネルは仮想記憶やスケジューリング，プロセス間通信等の基本的な機能のみを提供し，それ以外の高級なサービスはマイクロカーネル上で動作する上位のサブシステムが提供する．さらにこのサブシステムは，機能ごとに複数のプロセスにモジュール分けされ，プロセス間では明示的なメッセージを用いて通信を行っている．サブシステムを構成するシステム・プロセスを以降サーバと呼ぶ．マイクロカーネル構造のOSではデータや関数はサーバごとの管理となり，サーバを故障の隔離や回復の単位として利用することができるので，耐故障技術の採用に適した構造になっている．

我々は，マイクロカーネル構造を採用したUNIXの1つであるChorusマイクロカーネルとその上で動作するマルチサーバUNIXサブシステム[8)]を対象に耐故障技術の適用を試みた（図1参照）[☆]．ソフトウェアの構造は，下からマイクロカーネルの層，UNIXサブシステムの層，ユーザプロセスの層からなる3階層となる．図中のPMはプロセスマネージャ，FMはファイルマネージャ，STMはストリームマネージャ，IPCMはIPCマネージャ，KMはIPCキーマネージャであり，すべてUNIXサブシステムを構成するサーバで

☆ Chorusでは実行体のことをアクタと呼ぶが，以降では一般的な呼称であるプロセスを用いる．

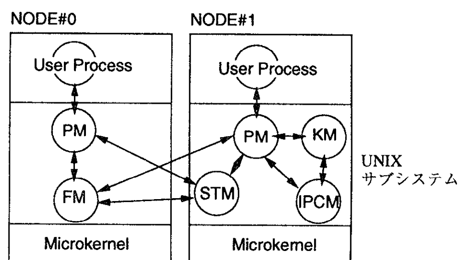


図1 Chorus microkernel 上で動作する UNIX サブシステム
Fig. 1 UNIX subsystem over chorus microkernel.

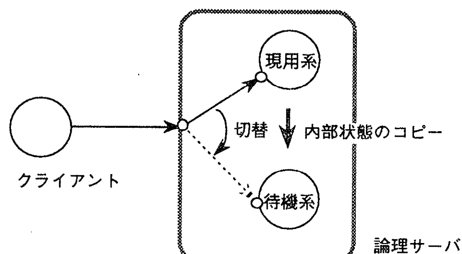


図2 プロセスペア方式
Fig. 2 Process pair method.

ある．

図1に示すように，システムは複数のノードから構成されている．ノードごとに別のマイクロカーネルが動作しているが，UNIX サブシステムは全ノードにまたがって動作している．そのため UNIX のユーザは個々のノードを意識する必要はなく，複数のノードが1つのシステムに見える．あるノードで誤りが発生してノードが停止しても，残りのノードでシステムを継続運用することができる．

3.3 プロセスペア方式

ソフトウェアによる耐故障技術に，プロセスペア方式[2),3),10)~13)]がある．プロセスペア方式は，サーバを現用系プロセス（以降，現用系と呼ぶ）と待機系プロセス（以降，待機系と呼ぶ）のペアで二重化し，誤りからの回復を可能にしている（**図2** 参照)．通常の運用時には，現用系がクライアントからの要求を処理し待機系は動作しない．現用系は内部状態を待機系にコピーして，誤り発生時の引継ぎに備える．状態のコピーには様々な方式があるが，通常実行時のオーバヘッドが小さく故障伝播の可能性が低い遅延引継ぎおよびエッセンス引継ぎ[3)]を採用した．

現用系に故障が発生すると，待機系が処理を引き継ぎ，新たに現用系に切り替わる．そして新たな待機系を起動し，将来の故障に備える．ノードのダウンに対処するために，現用系と待機系は必ず別のノードに配置する必要がある．

プロセスペア方式により，サーバのソフトウェアの故障（バグ）に起因する誤りや，サーバが動作しているノードのハードウェア故障による誤り，そのノードのマイクロカーネルのバグによる誤りを隠蔽することができる．サーバのバグによる誤りの中で再現性の高い誤りは，新現用系で再実行したときも再現してしまう．したがってプロセスペア方式はレビューやテストなどの故障回避技術との併用が前提となる．しかし，再現性の高いバグはレビューやテストによって容易に除去することができるので，品質保証のされた商用の OS では再現性の高いバグは実質的に排除されていると考えることができる．故障回避技術で発見できないバグは，複雑な条件下でのみ発生する再現性の乏しいバグであり，プロセスペアによって隠蔽することが可能である．文献2) では同一のバグによる誤りが新現用系で再現する確率は4%以下と推定されている．

状態コピーを行わずに現用系と待機系の間で同一の内部状態を保持する方法として，ホットスペア方式がある．ホットスペア方式ではクライアントからの処理要求を現用系と待機系の両方が実際に処理を行い，それぞれの内部状態を更新した後，現用系だけがクライアントに結果を返すというものである[6),14)]．

ホットスペア方式は，以下のような問題点があるので採用しなかった．

- 現用系と待機系の両方で実際の処理を行うので，処理量が倍増する．
- マルチスレッド化されたサーバでは，現用系と待機系の処理結果の同一性が保証できない．
- 結果の同一性を保証しようとすると，プロセスの多重化によって隠蔽できるはずのタイミング依存のソフトウェアバグが顕在化してしまう（現用系と待機系の両方で誤りが発生してしまう)．

3.4 高信頼 OS のアーキテクチャ

前述の設計目標を実現するため，マイクロカーネル化されたマルチサーバ UNIX をサーバ単位にプロセスペア方式を適用して耐故障化する．これによりハードウェアおよびソフトウェアの故障による誤りに対処することができる．さらに，プログラムの保守性や拡張性を損なわずにサーバを耐故障化するため，サーバとは独立に故障管理の機構を用意した．サーバは故障管理機構に対し，誤りの発生を通知したり，指示に従って回復処理をするだけで済み，個別の誤りの詳細を知る必要がない．**図3** に示すような，1つの新規サーバと2つのライブラリを用意している．

(1) **故障管理サーバ（Fault Manager, FTM)**
誤りの検出から回復までの一連の処理を指示する機能

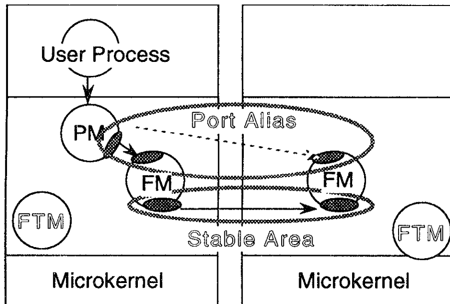


図3 高信頼 OS のアーキテクチャ
Fig. 3 Architecture of highly available OS.

をサーバとして実装する．サーバはノードごとに配置し，自分自身も耐故障性を持つ．

(2) **耐故障通信機構（Port Alias）**
耐故障性を備えたプロセス間通信機構で，クライアントからのリクエストの宛先の自動切替え処理や，故障発生時の再送処理，冗長なリクエストを検出し処理する機能を提供する．

(3) **データ安定格納機構（Stable Area）**
現用系の内部データを格納する機構で，現用系に故障が発生したときは，待機系が内容を読み出すことができる．

これら3つの機能について，次章以降で詳しく説明する．

4. 故障管理

4.1 故障管理モデル

ハードウェアの故障や OS の故障などいろいろな原因から誤りが発生するが，堅牢な故障管理機構を実現するためには，すべての誤りを同一の機構で統一的に管理することが重要である．故障対処のダイアグラムを図4に示す．

サーバ単位にプロセスペア方式を用いてプロセスの2重化を行っているので，サーバを故障回復の単位としている．しかし以下に述べる理由から故障隔離はサーバを単位とすることができず，ノードを単位としている．

Chorus のマイクロカーネルでは，実行性能向上のためマイクロカーネルおよび UNIX サブシステムに所属するサーバはカーネルアドレス空間を共有している．そのため1つのサーバで誤りが発生した場合，他のサーバやマイクロカーネルのデータを破壊する可能性がある．さらに，同一ノード内の軽量通信はコンテクストを切り替えず，呼び出し元と呼び出し先のサーバが実行スタックを共有するので，サーバ単位で分離することができない．

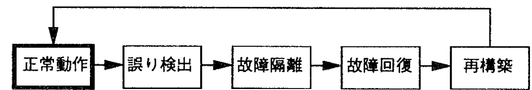


図4 故障管理モデル
Fig. 4 Fault management model.

そこで，あるサーバの故障が誤りを発生した際には，そのサーバが動作していたノード全体を停止させ，システムから切り離す．もし複数のサーバが停止ノードに含まれていた場合には，複数のサーバ回復処理を並行して開始する．

図4に示した故障管理の各ステップの内容を以下で詳しく説明する．

(1) **誤り検出**
マイクロカーネルおよびサーバには誤り検出コードが埋め込まれており，回復不可能な誤りを発見したら即時ノードをパニックさせることでフェイルファーストを実現する．ノード間ではハートビートメッセージを定期的に交換し生存確認を行っているので，ハートビートに応答しないノードがあると，そのノードで誤りが発生したことが検出される．

システムのパーティショニングを防止するために，システム全体でノードの生存情報の一貫性を保証している．また誤りの発生したノードが停止処理を完了できない可能性があるので，外部から強制的にノードを停止する機構を用意する．

(2) **故障隔離**
前述のように，実行性能とのトレードオフによりノードを故障隔離の単位としている．

(3) **故障回復**
故障回復はサーバ単位に行う．回復指示を受けた待機系が誤り発生直前の現用系の状態を再現する．待機系が消失した場合は，回復処理は不要であり，すぐにシステム再構成を行う．

複数のサーバが並行して回復処理を実行する場合がある．このときあるサーバの回復を行うために他のサーバに問合せをするなど，サーバ間の回復処理に依存関係があると，回復処理中にデッドロックが発生することがある．デッドロックを防止するためには，各サーバの故障回復処理は他のサーバに依存せずに完了できる必要がある．そこでサーバは自分の回復処理に必要なすべての情報をデータ安定格納機構に保存しておく．

故障回復後は待機系のない現用系のみの状態になるが，新しい現用系はサービス中断時間を短くするため，故障回復が完了した時点からサービスを再開する．

(4) システム再構成

故障管理サーバは，将来発生する誤りに対処するために新しい待機系を生成し配置する．サーバによってはハードウェアの接続状態などの条件から，待機系の配置可能なノードが限定される場合がある．

ダウンしたノードは自動的に再起動を試みる．再起動に成功した場合は再びシステムに組み込まれる．再起動に失敗すると，ある回数再起動を繰り返した後，再起動を放棄する．

現在はノード実行中にサーバを動的にシステム空間にローディングする機構がないため，新待機系の生成はノードの再起動時に行っている．

4.2 ユーザプログラムへの故障の見え方

上述した故障管理機構により，ハードウェアおよびOSの故障は隠蔽され，ユーザプログラムからは見えない．しかし運悪くユーザプログラムを実行しているノードがダウンしたときはユーザプログラムも異常終了してしまう．

4.3 故障管理機能の品質

システムの他の部分とは異なり，故障管理機構自身の誤りは致命的で回復することができない．そのため故障管理機構は，他より高い品質が要求される．高い品質を実現するために次のような配慮をしている．

- 個々の誤りに個別に対処するのではなく，すべての誤りを統一的に扱うことにより，故障管理機構が単純で小規模になる．そのため設計，コーディング，試験すべてのフェーズで品質を上げることができる．
- 複雑な機構は避け，なるべく単純な機構とする．たとえば2つのノードでほぼ同時に誤りが起こる確率は無視できる程度に小さいが，二重誤りにも対処できる故障管理機構は，一重誤りしか対処しない機構に比べ，倍以上複雑になり，その結果故障管理機構の品質が下がってしまう．したがって一重誤りにしか対処しない故障管理機構の方が，システム全体としては結果的に高い可用性を実現できる．
- 故障管理機構はシステムの他の部分より緻密なレビューと試験を実施する．システム開発全体で使用できる時間と資源が限られている中で，より多くの時間と資源を故障管理機構の品質確保に割り当てる．

4.4 Fault Manager (FTM)

説明してきた故障管理の機構は，Fault Manager (FTM) というサーバで実現している．FTMは，(1) 現用系・待機系の管理，(2) 引継ぎの指示と完了の同期，(3) 新待機系の生成配置，(4) 資源回収のためのサーバ消失通知の機能を提供している．

FTMは故障検出機構から他ノードの消失通知を受け取る．そして保持しているサーバの配置情報を基に消失ノードで動作していた現用系を特定し，対応する待機系に引継ぎ指示を送る．単一のノード消失により複数の現用系が消失した場合は，同時に複数の待機系に引継ぎ指示を送る．待機系から引継ぎ完了通知を受けると，その待機系を新しい現用系に昇格させ，新しい待機系を配置するノードを決定し新待機系を生成する．

また，プロセスペア化されていないサーバが消失したときは，そのサーバがこれまで使っていた他のサーバ内のデータを回収しなければならない．そのためにFTMはサーバの消失通知を送る．必要なサーバにのみ消失通知を送るため，FTMはサーバ間の依存関係を記録している．そのためFTM自身もノード故障に耐える必要がある．プロセスペアの仕組みを提供するFTMは，プロセスペアによって耐故障化することはできない．そこでFTMの複製を各ノードに複製配置し，全FTMを通信を単位にロックステップで実行する．FTMが動作するのは主に誤りが発生したときなので，ロックステップによる実行速度の低下は問題にならない．

5. 耐故障性を備えた通信機構

5.1 故障時の通信の問題

UNIXサブシステムを構成するプロセスは，マイクロカーネルが提供するメッセージ通信機構 (IPC) を用いて他のプロセスと通信している．IPCの宛先はポートと呼ばれ，ポートはいずれかのプロセスに所有されている．プロセスペア方式により複数プロセスから構成される論理サーバを作ると，クライアントから論理サーバにIPCで処理依頼を送るとき，次のような問題が発生する．

(1) 宛先の変化

サーバ側で引継ぎが起こるたびに，クライアントは通信先を新現用系のポートに切り替える必要がある．

(2) メッセージの消失

現用系の障害によって，クライアントから送信中だったメッセージが失われることがある．

(3) 冗長なメッセージの発生

メッセージの消失はクライアント側でメッセージを再送することで解決することができる．しかし誤りの発生タイミングによっては，すでにサーバで処理が完了したメッセージをクライアントが再送してしまう場合

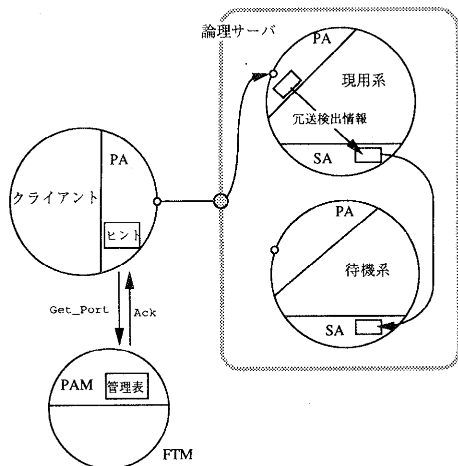


図5 耐故障通信ライブラリ（Port Alias）
Fig. 5 IPC library (Port Alias).

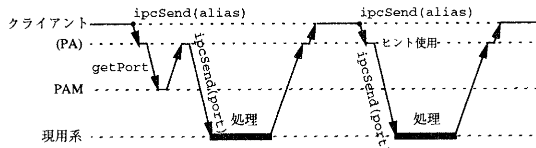


図6 Port Aliasの送信フロー
Fig. 6 Port Alias.

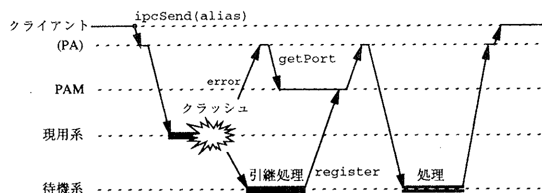


図7 サーバ引継ぎ時の送信フロー
Fig. 7 Takeover.

もある．すでに処理が完了しているメッセージの再送を冗送と呼ぶ．

サーバで行う処理が，ディスクの読み込みや書き出しのように，べき等（何回実行しても結果や副作用が同じ）の場合には，冗送を実行してもつねに正しい結果となる．しかし，ファイルの消去操作のようにべき等でない処理の場合は，冗送を実行すると誤動作することがある．たとえば，ファイル消去操作では，サーバが消去作業を完了し消去成功の返信メッセージを送るときに誤りが起こった場合，新現用系がクライアントからの再送メッセージを実行すると，消すべきファイルが存在しない旨のエラーを返してしまう．

5.2 通信ライブラリによる解決

上記のIPCの問題を解決するため，引継ぎを越えて存在する永続的なIPC送受信の宛先としてエリアスを導入する．クライアントはポートの代わりにエリアスを宛先として通信を行う．エリアスは耐故障通信で必要とされる以下の機能を提供する．この耐故障性を備えたプロセス間通信機構（Port Alias, PA）はライブラリとして提供し，プロセスペア化するすべてのサーバで使用することができる（図5参照）．

(1) 宛先の自動切替え

エリアスと対応するポートとの関係をPort Alias Manager（PAM）で管理している．PAMはFTMの一部として実装する．クライアントでエリアスに対する送信が起こると，現在の現用系のポートをPAMに問い合わせ，得られたポートに対してリクエストを送信する（図6参照）．クライアントではオーバヘッド削減のため，1度問い合わせたエリアスとポートの対をヒントとして保管し，以降の送信ではヒントを用いる．サーバの引継ぎが起こるとヒント情報は不正となるが，宛先不明でいったん通信に失敗した後，自動的にPAMから新現用系のポートを取得し再送信する．

(2) 自動再送信

PAは現用系のクラッシュに備えて，送信メッセージのコピーを保持している．サーバの現用系がクラッシュすると，実行中のIPCは異常終了する．クライアント側のPAは上記の宛先の切替処理を行った後，保持していたメッセージを新現用系宛に自動的に再送する（図7参照）．

(3) 冗長なメッセージの検出と処理

クライアント側のPAは，冗送を検出するため送信メッセージごとにユニークな識別子を添付する．サーバはべき等でない処理が完了し冗送の検出が必要になった時点で，冗送検出の開始をPAに依頼する．その際に，冗送が来たときに返すべき返信メッセージも指定する．以後，PAはメッセージを受信するたびに，メッセージから識別子を取り出して，冗送であるかどうかを検査し，冗送の場合には登録された返信メッセージを自動的に返す．

冗送検出と自動返信は，待機系が引き継ぐ必要がある処理なので，冗送検出情報と返信メッセージは後述するStable Area（SA）に一括保管される．

返信メッセージがクライアントまで到着し，以後再送（冗送）が起こらないことが確定すると，クライアントが冗送検出終了を通知する．サーバはこの通知を受けて冗送検出情報と返信メッセージを破棄する．冗送検出終了通知は，通常は次のメッセージにピギーバックすることで余分な通信を抑止する．

このようにPAが故障発生時の問題を解決しているので，クライアントとサーバは，引継ぎ時の通信に関

連した複雑な処理をほとんど意識する必要がない．利用者に対して誤りを自動訂正する従来の通信機構と比べると，PA ではカーネル提供の IPC と APC を合わせることで修正量を削減し，マルチキャストではなく失敗時だけ再送することで余分な通信を減らし，冗送検出情報も引き継ぐことで新現用系でも冗送検出を可能としている．

6. サーバの内部状態引継ぎ

6.1 内部状態の引継ぎ方式

データ安定格納機構（Stable Area, SA）は，プロセスペアの現用系と待機系の間で処理の引継ぎに必要なデータを受け渡すための機構である．現用系は，待機系が再生できない最低限の引継ぎデータだけを SA に保存し（エッセンス引継ぎ），現用系が故障すると，待機系が SA からデータを読み出して処理を引き継ぐ（遅延引継ぎ）[3]．

現用系の内部状態を引き継ぐためには，以下の機能が必要である．

(1) **永続記憶の実現**

現用系が SA に保存したデータを，現用系が消失したあとで待機系が利用できる必要がある．また待機系が存在しないときに現用系が保存したデータも，待機系生成後に，待機系から利用可能になる必要がある．SXO[3]では，永続記憶としてバッテリバックアップされた二重化メモリを用いているが，永続記憶を特殊なハードを用いずに実現する．

(2) **データ保存操作のアトミック性保証**

現用系がデータを SA に保存操作中に誤りが発生することがある．たとえば双方向リストの操作などは，一連のポインタ更新をアトミックに行う必要がある．すなわち，SA への一連のデータ保存中に故障が発生した場合は，一連のデータの保存がすべて完了しているか，1 つも保存されていないかのいずれかにならなければいけない．

(3) **性能保証**

SA のへの保存操作によるサーバの性能低下が許容範囲内である必要がある．さらに専用ハードウェア（バッテリバックアップされた二重化メモリ等）が利用できるときは，外部インタフェースを変更せずに性能低下をほぼなくすことができる必要がある．

6.2 二重書きによる永続記憶の実現

現用系と待機系の両方に SA 用メモリ領域を確保し，現用系で SA への書き込み依頼があると，現用系側のメモリ領域だけでなく待機系側にもデータを格納することで永続記憶を実現する．現用系がクラッシュして

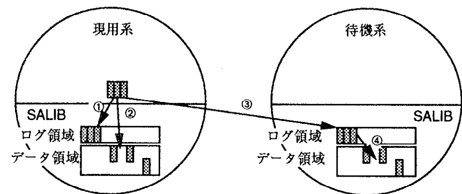


図 8 データ安定格納機構（Stable Area）
Fig. 8 Stable Area.

消失したときは，待機系は自分が保持しているメモリ領域からデータを読み出す．また現用系のみの状態から新待機系が用意された場合は，現用系の持つ SA 用メモリ領域全部を待機系のメモリ領域にコピーを行う（図 8 参照）．

6.3 ログを用いたアトミック操作の実現

アトミックな書き込みを実現するため，SA にはデータ領域とは別にログ領域を設ける．SA への書き込み依頼は，データ領域への書き込みに先だってログ領域に書き込み，そのあとデータ領域へ書き込まれる（図 8 参照）．ログ領域への書き込み中にクラッシュが起きた場合は，すべての書き込みデータは破棄される．データ領域への書き込み中にクラッシュが起きた場合は，引継ぎ時にログ領域のデータをデータ領域に反映することによりすべてのデータの書き込み完了が保証される．

6.4 性能向上の工夫

永続記憶およびアトミック性の保証を，前節までで説明した方法で素直に実現すると，SA へのデータ格納のたびに現用系から待機系にノード間通信が発生する．ノード間が低速ネットワークで結合されている場合には，処理のオーバヘッドが非常に大きくなり，性能が著しく低下してしまう．そこで，待機系へのデータ転送（同期処理）の実行を明示的に要求するための命令を用意し，SA の使用者が同期を明示的に指示するまで転送を遅延させることにする．複数のデータ転送をまとめることができるので転送回数を削減することができ，オーバヘッドを大幅に軽減できる．図 8 において，通常の SA 格納命令は，①および②のみ行いノード間通信は発生しない．同期処理が依頼されて初めて現用系のログ領域から待機系のログ領域への通信処理③を行う．待機系のログ領域への書き込みが終了すると，同期操作は終了する．待機系側でのログの展開処理④を非同期に実行することで応答速度をさらに向上させている．

6.5 ライブラリによる実装

データ安定格納機構もライブラリ（SALIB）として実現している．したがって使用者はデータの格納指示

と同期指示を出すだけで，必要なデータの引継ぎを実行することができる．データ領域の管理法には，格納アドレスとデータの長さを指定するものと，あらかじめセルの大きさと数を指定して配列を初期化しておき，セルの管理までライブラリが提供するものの2種類を提供している．

7. プログラムの修正の削減

既存のOSプログラムに耐故障性を付与するためには，将来の保守性や拡張性を確保するため，プログラムの修正を量と質の両面から削減しなければならないことを述べてきた．削減のための仕組みの1つは，FTMによって実現された故障を統一的に扱うことのできる管理の仕組みであり，もう1つが通信とデータ安定格納のためのライブラリである．

上記の機構に加え，既存プログラムを耐故障化するための手順を確立した．サーバを耐故障化する際にはまずデータ安定格納機構に格納するデータを特定する．次にプログラム上でデータを実際に格納すべき場所を特定する．スレッドセイフのプログラムの場合，資源のロックを獲得したり解放する場所が参考になる．そしてサーバ間のデータの受渡しプロトコルを調べ，サーバ間の依存関係と相手が故障したときに，どの資源を回収する必要があるかを調べる．プログラムの実際の修正はサーバの処理内容に依存するが，以上のような手順でサーバの耐故障化を進めることで，サーバの処理論理を変更せずに耐故障化が実現できる．

8. プロトタイプとその評価

8.1 プロトタイプの構成

これまでに述べてきた機能を使い，UNIXのファイル管理を行うFile Managerと関連する2つのサーバをプロセスペア方式で耐故障化した．耐故障化FMを以降FTFMと呼ぶ．作成した高信頼UNIXのプロトタイプの構成を**図9**に示す．

プロトタイプは，10Base5で接続された3台のPCからなるクラスタで，3ノードが1つのUNIXになっている．使用したPCは，i486/33 MHz，36 MBメモリ，IDE disk，Adaptec ISA SCSA HBAである．各ノードをkey node，slave 1，slave 2と呼ぶ．slave 1とslave 2は，1本のSCSIバスでSCSI diskを共有している．ユーザのプロセスはkey node上で動作するが，SCSIディスクのファイル操作を行うシステムコールは，すべてslave 1またはslave 2にあるFTFMにより処理される．試作システムではFTFM以外のサーバが故障するとシステムダウンになるのでFTFM以外はkey nodeに配置した．プロトタイプではkey nodeの故障は考えないことにする．

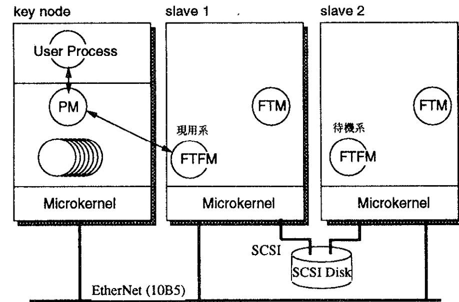


図9 プロトタイプの構成
Fig. 9 Configuration of prototype system.

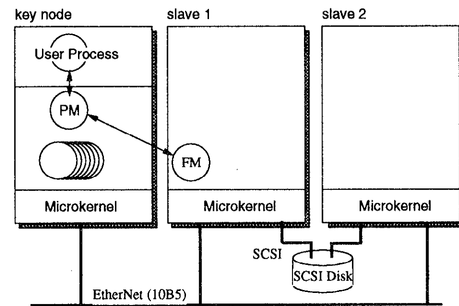


図10 対照システムの構成
Fig. 10 Configuration of contrastive pre-modified system.

表1 プログラムの変更行数の割合（%）
Table 1 Ratio of modified line of code (%).

	追加	削除	新規
全体	0.5	0.02	8.0
FTFM	0.8	0.05	4.5

高信頼化オーバヘッドを調べるための対照システムとして，修正前のマイクロカーネル版UNIXシステム（**図10**）でも測定を行った．対照システムではSCSI diskにアクセスするFMはslave 1で動作する．

8.2 コードの変更量

システムの高信頼化に必要な修正量を客観的に測る指針として，追加/削除/変更したプログラムの行数を調べた．修正前の行数に対する比率（%）で表したものが**表1**である．表中の“新規”はファイル単位で追加した新規機能を，“追加”および“削除”は既存のファイルに追加/削除した行数の比率を表している．

OS全体での結果を見ると，FTMやPALIB, SALIBといった新規機能が大部分で，それらを入れても変更量は10%以下であった．また，既存のサーバであるFMに対する変更もプロセスペアの初期化処理や引継ぎ時の処理など新規分が多く，純粋に既存コードを追加/削除した行数は1%以下であることが分かる．

また既存コードの修正内容も，SA へのデータの格納操作の追加等が中心で，FM の処理のアルゴリズムのにまでかかわる変更はほとんど必要なく比較的容易であった．これは，もともとの FM がマルチスレッドに対応するため，内部データを操作する際に，排他をとってアトミックに更新しているためである．

8.3 耐故障性の確認実験

プロトタイプの耐故障性を確認するために次のような実験を行った．(1) ハードウェアの故障をエミュレートするため，slave の電源を動作中に落す．(2) マイクロカーネルのソフト故障をエミュレートするために，動作中に slave のリセットボタンを押す．(3) 故障発生ツールを使って OS の内部で人工的なソフト故障を起す．いずれのケースにおいても，エラーはユーザには見えず，引継ぎ時間の分だけ処理が遅れただけであった．また，slave 1，slave 2 を交互に落としても期待される動作をした．

8.4 性能比較

高信頼化にともなうオーバヘッドを調べるため，対照システムとの性能比較を行った．典型的なマルチユーザ使用環境をシミュレートするベンチマークである SDET と Kenbus [15] を用いて性能を測定した．測定結果を**表 2** に示す．2 つのベンチマークとも single user での測定結果である．

Kenbus においてはオーバヘッドはほとんど見られない．これは Kenbus がユーザとの対話性能を測定しているため，ファイルシステムの性能劣化が結果に現れなかったのであろう．一方 SDET はファイルシステムに負荷が掛かるベンチマークなので，高信頼化システムの性能は 50%以上劣化していることが分かる．

システムの内部動作を詳しく調べた結果，処理時間の増加分 178 秒（= 378 − 200）のうち，約 30 秒が PA によるオーバヘッドであり，残りが SA と，SA が発行する通信のオーバヘッドであることが分かった．

さらに，SA の通信の内容を調べた結果，省略可能なデータを転送していることが分かり，転送データ量は約半分に減少できる見通しが立った．また，PA の処理も最適化の余地があるので，最終的には SDET で 300 秒以下（変更前の性能の 7 割程度）まで性能低下を軽減できると予想している．

8.5 故障回復時間

誤り検出から通常処理再開までにかかる時間を故障回復時間と定義し，必要な時間を測定した．現実の使用環境での回復時間測定のため，SDET ベンチマークの実行中に slave 1 の電源を落し，人為的なハードウェア故障を発生させた．複数回の測定の結果，故障回復

表 2 ベンチマークプログラムを用いた性能測定
Table 2 Performance measurement via benchmark program.

	修正前	高信頼修正後
SDET (sec)	200	378
Kenbus (scripts/hour)	13.84	13.81

には 10 秒から 13 秒程度必要であることが分かった．

故障回復処理を詳しく分析した結果，引継ぎデータを少し増やすことで，救済時間をさらに短縮できることが分かった．

9. おわりに

本論文では，プロセスペア方式をオープンな OS である UNIX に適用して高信頼化する際の問題点とそれに対する解決法を示した．

既存の OS のプログラムの拡張性や保守性を損なわずに耐故障性を付与するための仕掛けとして，故障管理機能，耐故障通信機構，データ安定格納機構が有用であることを示した．プロトタイプではファイル管理機能を担当する FM 等 3 つのサーバだけを耐故障化したが，提案している方式は UNIX サブシステムを構成するすべてのサーバを耐故障化できる汎用の仕組みとなっている．

プロトタイプを用いた実験により，ノード故障を隠蔽できることを実証した．耐故障化によるオーバヘッドにはまだまだ改善の余地があるが，詳細な解析により，オーバヘッド軽減の見通しが得られている．

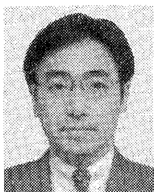
ファイルシステム以外の OS のすべてのサーバをプロセスペア化し，性能や可用性向上の定量的な評価を行うことは今後の課題である．

参 考 文 献

1) High Availability Working Group: Requirement for High Availability Technology, Survey Report, Unix International (1993).
2) Lee, I. and Iyer, R.K.: Faults, Symptoms, and Software Fault Tolerance in the Tandem GUARDIAN90 Operating System, *Proc. 23rd Annual International Symposium on Fault-Tolerant Computing*, pp.20–29, Toulouse, France, IEEE CS (1993).
3) Muramatsu, H., et al.: Operating System SXO for Continuous Operation, *Algorithms, Software, Architecture Information Processing 92*, Leeuwen, J.v. (Ed.), pp.615–621, IFIP, Elsevier Science (1992).
4) Borg, A., et al.: Fault Tolerant Under UNIX, *ACM Trans. Computer Systems*, Vol.7, No.1,

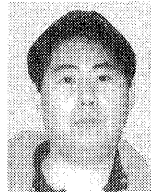
pp.1–24 (1989).
5) Babaoglu, O.: Fault-Tolerant Computing Based on Mach, *Operating Systems Review*, Vol.24, No.1, pp.27–39 (1990).
6) Marzullo, K., et al.: Tools for Monitoring and Controlling Distributed Application Management, *IEEE Computer*, Vol.24, No.8, pp. 42–51 (1991).
7) Kittur, S., et al.: Fault Tolerance in a Distributed CHORUS/MiX System, *Proc. USENIX 1996 Annual Technical Conference*, San Diego, CA, pp.219–228, USENIX (1996).
8) Batlivala, N., et al.: Experience with SVR4 Over CHORUS, *Proc. Usenix Workshop on Micro-kernels and Other Kernel Architectures*, Seattle, WA, pp.223–241, USENIX (1992).
9) Rashid, R., et al.: Mach: A Foundation For Open Systems, *Proc. 2nd Workshop on Workstation Operating Systems*, Pacific Grove, California, IEEE CS (1989).
10) Gray, J. and Reuter, A.: *Transaction Processing: Concepts and Techniques*, chapter 3, pp.119–148, Morgan Kaufmann (1993).
11) Gray, J.: A Census of Tandem System Availability Between 1985 and 1990, *IEEE Reliability*, Vol.39, No.4, pp.409–419 (1990).
12) Gray, J.: Why Do Computer Stop and What Can Be Done About It?, Technical Report, TR85.7, Tandem Computer (1985).
13) Iyer, R.K. and Lee, I.: Software Fault Tolerance in Computer Operating Systems, *Software Fault Tolerance*, Lyu, M.R. (Ed.), pp.249–278, John Wiley and Sons (1995).
14) Major, D., et al.: An Overview of the NetWare Operating System, *USENIX Winter 1994 Technical Concerence Proceedings*, pp.355–372, San Francisco, CA, USENIX (1994).
15) Open Systems Group: SPEC SDM release 1.0 benchmarks, Specification, SPEC Consortium (1993).

(平成9年2月7日受付)
(平成9年9月10日採録)



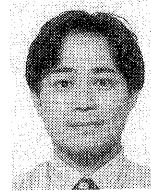
岸本 光弘（正会員）

1958年生．1981年東北大学工学部通信工学科卒業．1983年同大大学院修士課程修了．同年（株）富士通研究所入社．現在，コンピュータシステム研究部主任研究員．並列・分散OSの高性能化，高信頼化の研究開発に従事．1997年から東北大学大学院情報科学科博士課程に在学中．IEEE会員．



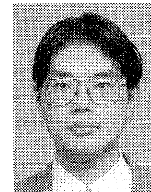
中島 淳

1961年生．1986年東京大学計数工学科卒業．同年（株）富士通研究所入社．UNIXをベースにしたマイクロカーネル，分散技術の研究開発に従事．1996年SCO（米国New Jersey）に移り，現在SVR5の開発，特にプロセス管理を中心にccNUMA，64 bitサポートなどに従事．



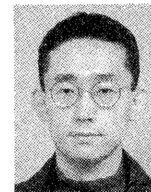
大橋 勝之（正会員）

1965年生．1987年静岡大学工学部電子工学科卒業．同年（株）富士通研究所入社．オンライン手書き文字認識技術，ペン入力の携帯型マルチメディア情報機器の研究開発を経て，現在は並列/分散システムの高可用化/負荷分散制御技術の研究開発に従事．



金沢 裕治（正会員）

1965年生．1988年東京大学工学部計数工学科卒業．1990年同大大学院修士課程修了．同年（株）富士通研究所入社．並列・分散OSの高性能化，高信頼化の研究開発を経て，現在はVLSI配置システムの研究に従事．



土屋 芳浩

1965年生．1989年東京大学工学部計数工学科卒業．同年（株）富士通研究所入社．マイクロカーネルOS，ファイルシステムの研究開発に従事．情報と人間のかかわりに興味を持つ．ACM，USENIX各会員．

今井 祐二（正会員）

1968年生．1990年名古屋大学工学部情報工学科卒業．1992年同大大学院工学研究科情報工学専攻博士前期課程修了．同年（株）富士通研究所入社．現在，コンピュータシステム研究部に所属．並列・分散OSの高性能化，高信頼化の研究開発に従事．