オペレーティング・システム　56－6
(1992. 8. 19)

永続オブジェクトによるオペレーティングシステムの構成

平瀬 吉也[†]　大久保 英嗣[†]　大野 豊[†]　白川 洋充[††]

[†]立命館大学理工学部情報工学科
[††]近畿大学理工学部経営工学科

分散システムの最大の課題は，並行性と信頼性の向上であるが，従来の分散オペレーティングシステムの研究は，並行性の向上だけに目が向けられてきた．この理由は，オペレーティングシステムを分散環境で利用してきた歴史が浅いためである．一方，分散システムの重要な応用領域であるデータベースシステムの分野では，着実に信頼性向上の技術が積み重ねられてきた．本研究の目的は，この技術をオペレーティングシステムの構成法に取り入れることである．即ち，永続オブジェクトによりオペレーティングシステムを構成することで，信頼性のあるオペレーティングシステムの実現を目指すものである．

An Operating System on Persistent Objects

Yoshiya Hirase[†]　Eiji Okubo[†]　Yutaka Ohno[†]　Hiromitsu Shirakawa[††]

[†]Department of Computer Science and Systems Engineering,
Faculty of Science and Engineering, Ritsumeikan University
56-1 Tojiin Kita-machi, Kita-ku, Kyoto 603, Japan

[††]Department of Industrial Engineering,
Faculty of Science and Engineering, Kinki University
3-4-1 Kowakae, Higashi-Osaka, 577, Japan

Concurrency and reliability are main issues of the distributed systems. However, conventional distributed operating systems have been concerned only with concurrency. This is due to the fact that operating systems have not yet been used intensively in distributed environments. On the other hand, distributed database systems had improved reliability. The aim of this paper is to apply those techniques to an operating system. Firstly, the operating system is represented by objects. Secondly, those objects are realized by persistent objects. Thus the operating system can be composed by persistent objects, and persistent operating system may be realized.

1 はじめに

分散システムの最大の課題は，並行性と信頼性の向上である．分散システムの二つの重要な応用領域であるデータベースシステムとオペレーティングシステム(ＯＳ)は，多くの共通の問題を持ちながらも独自の技術を開発し，並行性と信頼性の向上を図ってきたと言える．しかし，最近の研究では，これらの分散システムの諸領域の課題を共通のフレームワーク上で解決していく試みされている．この中でも，Cornell 大学の ISIS プロジェクトは注目に値するものがあると言えよう [1].

ＯＳは，分散環境の広範囲な応用に利用する目的で分散化の方向に進み，さらにソフトウェア工学上の利点だけでなく，性能面からもオブジェクト指向の技法を取り入れるようになってきている [2]．しかし，従来のＯＳの研究は，分散環境においては並行性の向上だけに目が向けられてきたといえる．この理由は，ＯＳを分散環境で利用してきた歴史が浅く，かつその信頼性は Non-Stop Computer に代表されるようなハードウェアの多重化技術に依存してきたためである．一方，データベースシステムの分野では着実に信頼性向上の技術を積み重ねてきた．ＯＳの分野でも，上述の ISIS に触発され，信頼性を有するＯＳの研究も行われるようになった [3]．また，データベース用のプログラミング言語である永続プログラミング言語の研究が盛んに行われるようになり，ＯＳも永続オブジェクトをサポートするようになってきている [4][5][6]．本論文で紹介するＯＳは，これらとはのＯＳ目的が異なっている．ＯＳをオブジェクトで構成し，そのオブジェクトに永続性を持たせることにより信頼性のあるＯＳの実現を目指すものである．

2 ＯＳの永続性

本来永続性とは，データベース用のプログラミング言語で使用されてきた用語である．永続性をもつオブジェクトとは，プログラムの終了とは関係なくデータベースの中に存在するものをいう．ところでＯＳは，計算機がクラッシュしたり，シャットダウンされるまでは計算機上に存在している．ＯＳが永続性を持つように構成することにより，クラッシュした場合の影響からＯＳを守り，システムの信頼性を向上させることができる．ここで，ＯＳに永続性を持たせる方法が問題となる．一般に分散システムでは，チェックポイントと複製で状態を保護することによって，故障に対する信頼性を高め，ロールバックでトランザクションの回復を実現するというアプローチでシステムに永続性を持たせることが行われている．ＯＳにおいて，特にリアルタイムを考慮した場合には，一般の分散システムでとられているこのアプローチをそのまま採用することはできない．そのため，ＯＳ固有の技術を開発する必要がある．

ＯＳは，カーネルの機能を最小限にとどめ，システムの機能をカーネル外で実現する方向に進んでいる．分散ＯＳ Theta [2] は，カーネルの機能を必要最小限にとどめることにより極小カーネルを実現し，システムの大部分の機能をシステムサーバにより構成している．本論文では，極小カーネルＯＳのシステムサーバをオブジェクトとして実現し，それらのオブジェクトに永続性を持たせることを考察する．この手法では，一般の分散システムに永続性を持たせる場合と比較して小さなオーバヘッドで構成できる手法について述べ，本手法を使えばリアルタイムにも十分対応できる．

永続プログラミング言語 (Persistent Programming Language, 略して PPL) と本論文書の永続オペレーティングシステム (Persistent Operationg System, 略して POS) の関係について説明する．M. Atkinson ら [7] も述べているように，

PPL と POS の機能には重複するところが多々あり，PPL と POS の実現を別々に行う場合，両者の機能の衝突や干渉により，性能の劣化ならびに一貫性の欠如を引き起こす可能性がある．これを防ぐには，まず PPL をベアマシン上で作り，この PPL を使って POS を実現すればよいとされている．

本論文で提案する POS は，将来 PPL を使って POS を構築することを考えており，PPL の機能に係わるものは一切実現していない．

2.1 Theta の極小カーネル

Theta では，極小カーネルを実現しており，従来のシステムの機能は，カーネル外のシステムサーバとライブラリ関数により実現している．ＯＳのほとんどの機能をシステムサーバで実現することにより，ＯＳ自体の並行性が向上し，その結果としてシステム全体の並行性も高めている．Theta では，実行の単位としてスレッドを用いており，タスクによってスレッドと資源の管理を行っている (図1参照)．また，同一タスク内には複数のスレッドが存在でき，スレッド間で資源の共有を行っている．システムサーバでは，メモリやＩ/Ｏデバイス等のシステムのハードウェア資源の管理，タスクやスレッドの生成，スレッドの実行状態の遷移の管理を行っている (図2参照)．これらのシステムサーバは，互いに独立性が高く，システムサーバ自体の改良や追加を容易に行える．

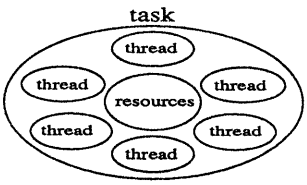

図１　スレッドとタスク

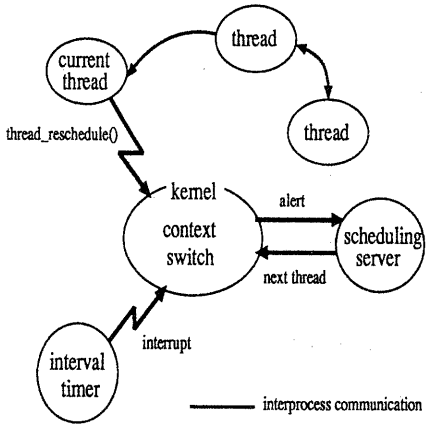

図２　カーネルとシステムサーバ

2.2 Theta におけるオブジェクト

Theta では，同一タスク内のスレッド間で資源を共有している．従って，スレッドを管理し，共有資源を保持しているタスクをオブジェクトに対応付けることによって Theta をオブジェクトの集合で構成することが可能となる．特に，Theta のシステムサーバをシステムオブジェクトと呼ぶ．各々のオブジェクト内では，複数のスレッドが管理され，そのスレッド間でオブジェクト内の資源の共有を行う (図3参照)．

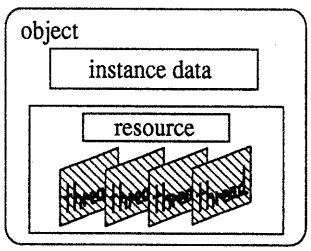

図３　オブジェクト

Thetaにおけるシステムオブジェクトは，システムの機能を実行するオブジェクトであり，ＯＳの一部であると見なすことができる．また，他のオブジェクトとは独立性が高いために，その状態を保持することが他のオブジェクトに比較して容易である．従って，システムオブジェクトを永続オブジェクトとすることで，簡単に実現でき，結果としてＯＳ自体に永続性をもたせることも可能となる．

2.3 永続性のレベル

ＯＳを永続オブジェクトで構成する場合，永続性実現のためのオーバヘッドが問題となる．Thetaでは，オーバヘッドを軽減させるために，弱い永続性 (weak persistence) と強い永続性 (robust persistence) との組合せによる永続性のレベルを設定している．ここで，弱い永続性とはオブジェクトの複製を揮発性メモリに生成することを意味し，強い永続性とはオブジェクトの複製をstableデバイスに生成することを意味する．

実際に処理を実行している永続オブジェクトはアクティブであるとし，複製オブジェクトはパッシブであるとする．そのため，アクティブである永続オブジェクトが失われた場合，パッシブである複製オブジェクトをアクティブにすることで処理が回復できる．弱い永続性を持つオブジェクトは，メモリ上に複製が生成されるために，障害回復処理のオーバヘッドが小さいが信頼性は低い．このため，弱い永続性を持つオブジェクトは，短い周期で状態が変化する場合に使用する．強い永続性とは，永続オブジェクトの状態保持をstableデバイスに対して行う．stableデバイスとは，ディスクなどの不揮発なデバイスである．状態保持のための処理は，ＯＳの実行に比較的大きな影響を与えるが，先に述べた弱い永続性とは違い，stableデバイスへの保持であるため，その状態は失われない．システムが動作しているノードが不揮発なデバイスを持たない場合には，他のノードへ依頼して行う．オーバヘッドが大きいため，比較的長い周期で状態が変化する永続オブジェクトに用いる．stableデバイスは，トランザクションとロールバックで実現するが，ＯＳの場合にはシングルユーザを考えればよい．

永続性のレベルにより，永続性とオーバヘッドとのトレードオフを図る必要がある．

3 永続性の機構

本章では，ＯＳに永続性を持たせるためのオーバヘッドを軽減するために用いる永続性レベルについて述べる．また，永続性の機構として，障害の検出，回復の機構について述べる．

3.1 永続オブジェクトの状態保持

永続オブジェクトは，オブジェクトに永続性を持たせたものであり，一度生成されると消去されるまで存在し続ける．そのためには，状態 (インスタンス) の変化を保持する必要がある．すでに述べた，弱い永続性と強い永続性の組合せによって永続性のレベルを複数用意する．即ち，永続性のレベルを選択することを可能とし，オーバヘッドを軽減する．表1に永続性のレベルと永続性の種類との関係を示す．表1においてローカルノード/リモートノードは，複製が存在するノードを示している．例えば，レベル７ ではローカルとリモートノードで弱い永続性を使用している．即ち，複製がメモリ上に存在することになる．さらにレベル７では，リモートノードで強い永続性を使用している．永続性レベルは，複製が存在するノードと永続性の種類の組合せによって８種類存在する．

永続性による状態保持の処理は，チェックポイントを通過するときに行われる (図4参照)．

表 1　永続性のレベル

永続性のレベル	ローカルノード		リモートノード	
	弱い永続性	強い永続性	弱い永続性	強い永続性
レベル 1	○			
レベル 2		○		
レベル 3	○	○		
レベル 4			○	
レベル 5			○	○
レベル 6	○		○	
レベル 7	○		○	○
レベル 8	○	○	○	○

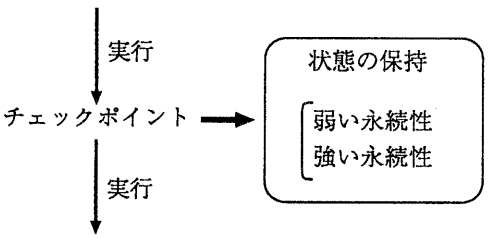

図 4　チェックポイント

3.2　永続オブジェクトの管理

永続性には，弱い永続性と強い永続性があることを述べた．ここでは，その管理について述べる．

(1) 環境オブジェクト

システムオブジェクトの一つである環境オブジェクトは，各ノードに一つ存在している．ノード内に存在する永続オブジェクトを管理し，生成，消滅，オブジェクトの名前参照の解決，永続性の処理なども行っている．永続性の処理とは，永続オブジェクトがクラッシュしたことを検出し，保持状態から永続オブジェクトを回復させることである．

また，環境オブジェクトも永続オブジェクトであるので，そのための永続性の処理も行っている．

(2) 弱い永続性の管理

弱い永続性を持つ永続オブジェクトの生成時に，複製オブジェクトが生成されるノードの環境オブジェクトに複製オブジェクトの生成が依頼され，そのノードで管理される (図 5参照)．生成する複製オブジェクトの数は，永続オブジェクトの生成時に指定することができる．

(3) 強い永続性の管理

強い永続性では，環境オブジェクトにより，チェックポイントを通過するごとに，stable ストアオブジェクトに対して状態保持が依頼される (図 6 参照)．stable ストアオブジェクトは，システムオブジェクトの一つであり，stable デバイスを持っているノードに存在する．

3.3　クラッシュの検出

永続オブジェクトに対する障害 (永続オブジェクトが失われた) を検出する機構には，永続オブジェクトに要求を送る側 (クライアント) と，永続オブジェクトを管理している側 (環境オブジェ

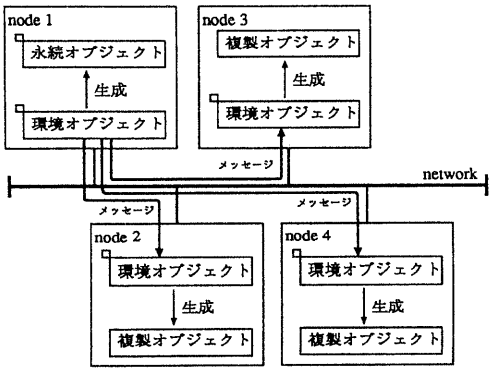

図 5　複製の生成

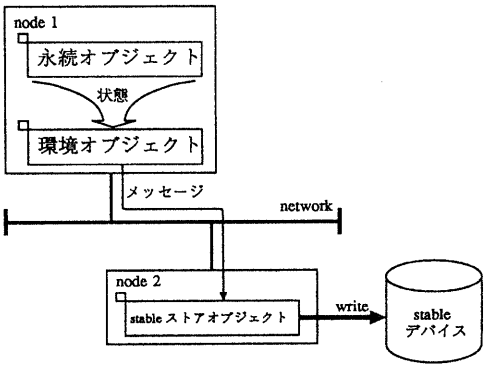

図 6　stable デバイスへの状態保持

クト) の機構の 2 つが存在する．ＯＳに永続性を持たせる場合には，ノードのクラッシュを検出する必要もあり，その機構についても述べる．

(1) クライアントからの検出

クライアントは，オブジェクトに対してメッセージを送信することにより処理を要求する．クライアントがメッセージを送信した後，一定期間内にオブジェクトからの応答がない場合は，エラーとなる．しかし，クライアントが処理を要求したオブジェクトが永続オブジェクトの場合には，失われた状態にあると判断される．さらに，環境オブジェクトに対して，対象となっている永続オブジェクトの回復を依頼し，メッセージを待つ．環境オブジェクトからの回復処理の結果を受け取り，回復されていれば，その永続オブジェクトに対して処理を要求する．

(2) 環境オブジェクトによる検出

環境オブジェクトの管理しているオブジェクトが永続オブジェクトである場合，一定期間ごとにメッセージにより永続オブジェクト (アクティブ) と複製オブジェクト (パッシブ) の存在を確認する．さらに，パッシブ側を管理している環境オブジェクトからも，アクティブの存在を確認する (図7参照)．どちらの場合も存在が確認できず，失われたと判断された場合，回復処理を行う．

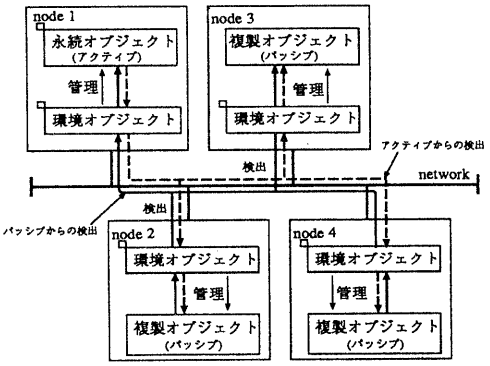

図 7　環境オブジェクトによる検出

(3) ノードのクラッシュの検出

ノードのクラッシュは，他のノードからの検出により行う必要があるが，ノードに存在する環境オブジェクトからの検出により行う．多ノードの場合，各ノードの環境オブジェクトからの検出結果に基づきクラッシュしたかの判定を行い，クラッ

シュしていれば回復処理に移る (図8参照).

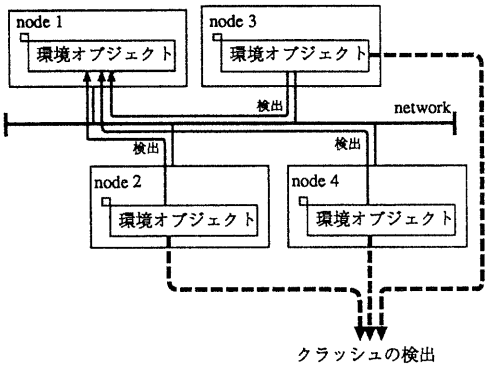

図 8　ノードのクラッシュの検出

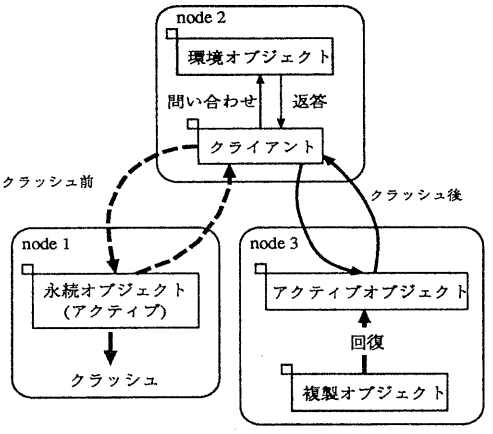

図 9　弱い永続性での回復処理

3.3.1　クラッシュからの回復

環境オブジェクトは，永続オブジェクトのクラッシュが検出されると回復処理を行う．回復処理を行うための使用対象は，弱い永続性/強い永続性，ローカル/リモートの優先順位が存在するため，永続性のレベルによって処理が異なり，リモートよりローカル，強い永続性より弱い永続性からの保持状態を用いた処理を行う．

(1) 弱い永続性での回復処理

弱い永続性では，複製を用いて行っているため，クラッシュした場合，複製オブジェクトの一つをアクティブにすることにより回復を行う．即ち，永続オブジェクトのクラッシュが起こったノードでの事象(イベント)をアクティブにしたオブジェクトにより行う．クラッシュした永続オブジェクトにメッセージを送信する場合，環境オブジェクトの参照によりメッセージを送受信する．そのため，環境オブジェクトにより，クラッシュした永続オブジェクトへの参照を変更して，新たにアクティブにしたオブジェクトへの参照にすることにより解決することができる (図9参照).

(2) 強い永続性での回復処理

強い永続性では，stableデバイスに対して状態保持を行っている．そのため，保持状態の読み出しを行い，その状態を用いてオブジェクトを生成する (図10参照)．生成されたオブジェクトは，クラッシュする前の永続オブジェクトとして振舞う．

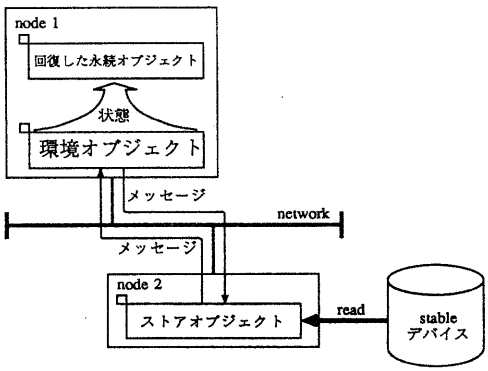

図 10　強い永続性の回復処理

(3) ノードのクラッシュからの回復

ノードのクラッシュが検出された場合，あるノードによりクラッシュしたノードの代行を行う．代行処理を行うノードでは，代行処理を行う以前には独自の処理を行っているが，その状態を stable デバイスへ保持し，クラッシュしたノードの状態に切り替わる．

4 おわりに

本論文では，ＯＳに永続性を持たせる方法について述べた．ＯＳに永続性を持たせることで信頼性は向上するが，信頼性と永続性実現のためのオーバヘッドとはトレードオフの関係にある．従って，永続性にレベルの概念を導入し，ＯＳ設計者が各オブジェクトに対して永続性のレベルを設定可能にする手法について考察した．本手法により，オーバヘッドを軽減する一方で，OS の信頼性を高めることが可能になると考えられる．

参考文献

[1] Kenneth P. Birman, *Maintaining Consistency in Distributed Systems*, Technical Report TR 91-1240, Department of Computer Science, Cornell University, November 1991.

[2] Hiromitsu Shirakawa and Eiji Okubo, *When Object-Oriented Operating System is Time Critical*, 1992 EUROMICRO Workshop on Real Time Systems, pp.54-59, June 1992.

[3] Tony P. Ng, *The Design and Implementation of a Reliable Distributed Operating System-ROSE*, Proceedings of the Ninth Symposium on Reliable Distributed System, pp.2-11, 1990.

[4] Partha Dasgupta, Richard J. LeBlanc, Jr., Mustaque Ahamad and Umakishore Ramachandran, *The Clouds Distributed Operating System*, IEEE Computer Vol. 24, No. 11, pp. 34-44, November 1991.

[5] Marc Shapiro, Philippe Gautron and Laurence Mosseri, *Persistence and Migration for C++ Objects*, In *Proceedings of the 1989 European Conference on Object-Oriented Programming*, Cambridge University Press, pp. 191-204, 1989.

[6] Roy H. Campbell and Peter W. Madany, *Considerations of Persistence and Security in Choices, an Object-Oriented operating System*, In *Security and Persistence*, pp. 289-300, Springer-Verlag 1990.

[7] Malcom Atkinson and Ronald Morrison, *Persistent System Architecture*, In *Persistent Object Systems*, pp. 73-97, Springer-Verlag 1989.