

HORB のレプリケーション拡張におけるレプリカの自動制御

若狭 建 山口 実靖 相田 仁 齊藤 忠夫

東京大学工学部 電子情報工学科

筆者らが提案及び実装を行っている“レプリケーション拡張 HORB”はモバイル/分散環境などの不安定な環境においても十分な可用性を持つ分散オブジェクトシステムのプロトタイプである。現状の分散オブジェクトシステムは、安定したネットワークへの常時接続を前提としており、それ以外の環境での可用性は著しく低下する。提案システムはリモートオブジェクトのレプリカをクライアント側に作成するというアプローチを採用し、リモートオブジェクトに対する操作を行う際、オリジナルオブジェクトの代わりに適宜ローカルにあるレプリカを使用することにより、ディスコネクトドオペレーションなどの高度な機能を達成した。

また、初期のシステムはプログラマが明示的に指示することによりレプリカの管理を手動で行っていたが、更なる可用性の向上を目指しシステムによるレプリカの自動管理を行うように改良を施した。その結果、プログラマが全く意識することなくレプリケーションに対応したアプリケーションが作成できるようになった。

Automatic Replica Control for Replication Enhancing of HORB

Ken Wakasa Saneyasu Yamaguchi Hitoshi Aida Tadao Saito

Department of Information and Communication Engineering,
Faculty of Engineering, University of Tokyo

We have been proposing and implementing “Replication Enhanced HORB” which is a prototype of distributed object system achieving higher availability for unstable environment such as mobile/distributed one. Current distributed object systems depending on permanent connectivity to stable networks, so their availability fall down significantly under unstable environment. The proposed system takes an approach that creating Replica of remote object at local side, and using Replica suitably on each remote method invocation. In this way, advanced functions such as Disconnected Operation is realized.

Our early system which replica was manually manipulated requires explicit Replica Controls by programmers. We have improved and implemented an automatic replica control version in order to extend the availability of the system. As a result, programmers need not to pay special attention to create replication-aware applications.

1 はじめに

分散オブジェクト技術は、ネットワークコンピューティングにおける今後のキーテクノロジーの一つであるが、現状ではネットワーク透過性や位置透過性の実現に重点が置かれているものが多く、システムによる可用性や性能の向上などの支援は課題として残されている。我々は、レプリケーションの枠組を用いた分散オブジェクトシステムの可用性・信頼性及び性能向上の効果を確認するため、プロトタイプとして JavaTM 上の分散オブジェクトシステムである HORB をベースにレプリケーション拡張を施し、評価を行った。本稿では“レプリケーション拡張 HORB”のアプローチ、基本構成及び基本性能を示し、続いてレプリカ管理に関して手動管理と自動管理の比較検討を行う。

2 レプリケーション拡張 HORB

2.1 HORB

HORB[1] (図1) は JavaTM を拡張することによりオブジェクト指向ネットワークコンピューティング

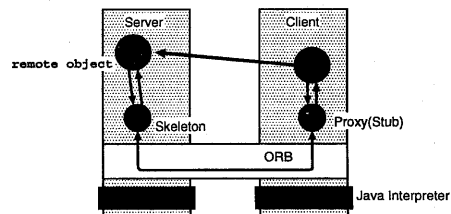


図 1: HORB のアーキテクチャ

を実現している分散オブジェクト言語 (システム/ライブラリ) である。クライアントはリモートオブジェクトの代理であるプロキシオブジェクトを経由して、リモートオブジェクトにアクセスする。プロキシオブジェクトは通信処理を行う ORB (Object Request Broker) 及びサーバ側にあるスケルトンオブジェクトを介して通信を行って、リモートオブジェクトを利用することにより、リモートオブジェクトのメソッドをクライアントに提供する。これらの機構によりプログラマは下部の通信機構を気にすることなくコーディングできる。

2.2 システム提案の動機

モバイルコンピューティングに代表されるような様々なコンピュータの使用形態（図2）が広まるにつれ、不安定な環境でも不自由なく操作が行えるシステムが求められている。このような状況を踏まえ、ディスコネクテッドオペレーション[2]、障害対策、高信頼化、キャッシングによる高速化、といった機能を分散オブジェクトの枠組の一部として提供すべきと考えた。拡張に当たっては分散オブジェクトシステムの利点である使い易さを損なわないよう配慮した。

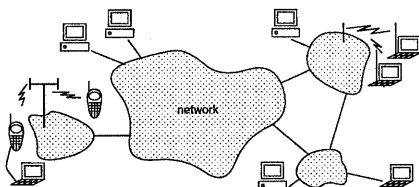


図2: 様々なネットワークでのコンピュータの使用形態

2.3 提案システムのアプローチ

オブジェクトのレプリケーションの手法[3]をシームレスな形で HORB に統合することを目標とする“レプリケーション拡張 HORB”システムを提案する。HORB はマルチプラットフォームである Java 上で稼働するため移植性に優れており、かつネットワークとの親和性が高い。また種々の分散オブジェクトシステムの中で最も容易に使えるものの一つであるので今回採用した。“レプリケーション拡張 HORB”の特徴を示す。

- リモートオブジェクトのレプリカをクライアントが保持し、HORB のリモートオブジェクトに対するディスコネクテッドオペレーションが可能
- ディスコネクテッドオペレーションはユーザからは完全に透過的
- 通信量の削減
- 楽観的なレプリカ管理手法とそれによるレプリカ間の不整合の検出とその解決

拡張は horbc コンパイラのみを改造することにより行ったので、実行時環境に一切手を加えず動作可能である。なお提案システムは Java, HORB という特定の環境を想定しているが、提案システムで用いた手法・技術は分散オブジェクト技術一般に適応できると考える。

2.4 提案システムの構成

提案システムの具体的な構成を図3に示す。Java のオブジェクトシリアライズ（直列化）[4]機能を用いてオブジェクトをバイストリームに変換し、それをサーバからクライアントに転送することによってレプリカを作成する。ネットワークから切断された状態では、

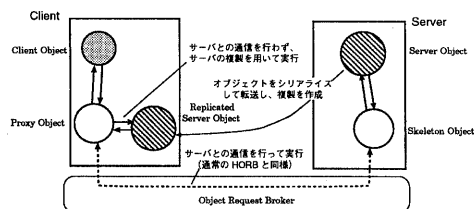


図3: レプリケーション拡張 HORB の構成図

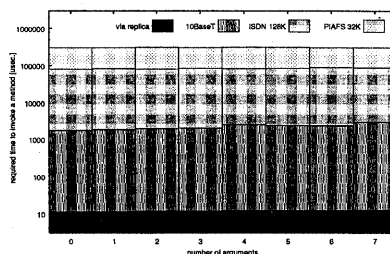


図4: リモートメソッド呼び出しに要する時間

サーバオブジェクトの機能を肩代りするため Proxy オブジェクトとレプリカが協調して動く。また HORB のセマンティクスが保たれているので、クライアントはディスコネクテッドオペレーションのためにメソッドを呼び分けたりする必要がない。

2.5 基本性能

同一のサーバ及びクライアントマシンを用い、異なるネットワークを経由させた際の提案システムの性能を測定した。

サーバはデスクトップ PC (PentiumPRO 200MHz, Linux 2.0)、クライアントはノート PC (Pentium 150MHz, Linux 2.0) で、それぞれ HORB 1.3b1 (+replication), JDK1.1.3 を用い、Ethernet 10BaseT, ISDN 128K, PIAFS¹ 32K の3種類のネットワークで比較を行った。図4は各ネットワークでのリモートメソッド呼び出しに要する時間の測定結果である。横軸はメソッドの引数の数である。当然ながらレプリカに対するメソッド呼び出し速度はどのネットワークにおいても同一である。

各ネットワークにおいて、レプリカに対するメソッド呼び出しはネットワーク経由のリモートメソッド呼び出しに対して 10BaseT で約 190 倍、ISDN 128K で約 7100 倍、PIAFS 32K で 26000 倍高速である。

図5は各ネットワークにおけるレプリカ作成に要する時間の測定結果である。転送バイト数が増加するにしたがって傾きが1に近づいており、転送バイト数小さい場合には効率が悪いことが分かる。これは通信処理及びオブジェクトのシリアライズのコストが効いてくるからである。つまり本システムで小さなオブジェ

¹PHS Internet Access Forum Standard

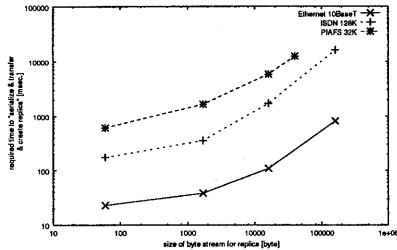


図 5: 各ネットワークでのレプリカ作成に要する時間

クトを頻繁に転送するのは得策ではない。このコストの低減は今後の課題の一つであり、レプリカの差分だけを送ることや、また圧縮プロトコルの使用などが考えられる。

上記の測定結果から、レプリカに対するメソッド呼び出しを積極的に活用することにより、特に細いネットワークほど大幅にメソッド呼び出しが高速化できることが確認できた。しかし、図 5 で示したようにレプリカ生成のコストが比較的大きいことに注意しなければならない。

3 レプリカ管理手法

3.1 レプリカ手動管理メソッド

初期の提案システムでは、レプリカを用いるためにはプログラマがメソッドを呼び出すようにした。レプリカがローカルにある時がオフラインモード、ない時がオンラインモードである。Proxy オブジェクトに追加したレプリカ操作メソッドを以下に示す。

createReplica() レプリカを作成しオフラインモードに移行する。

syncReplica() レプリカへの変更をオリジナルに反映させる。レプリカはクライアント側に残したままにしておく。

commitReplica() レプリカへの変更をオリジナルに反映させ、レプリカを消去しオンラインモードに移行する。commitReplica(true) と同じ。

commitReplica(boolean sync) レプリカへの変更をオリジナルに反映 (して/しないで)、レプリカを消去しオンラインモードに移行する。引数 sync により動作を変更できる。

discardReplica() レプリカへの変更をオリジナルに反映しないで、レプリカを消去しオンラインモードに移行する。commitReplica(false) と同じ。

レプリカ作成の流れを図 6 に示す。Proxy オブジェクトはバイトストリームを受け取り、それをデシリアライズ (deserialize) し、レプリカオブジェクトとして復元する。

なお、本システムは「楽観的なレプリカ管理方式」を用いているので、syncReplica(), commitReplica()

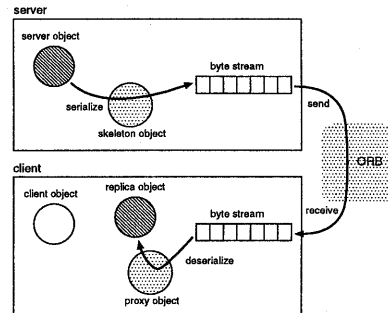


図 6: レプリカ作成の流れ

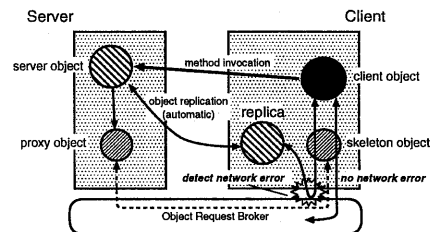


図 7: レプリケーションの自動化

を呼び出した際に、自分が持っているレプリカとオリジナルの双方の変更が衝突する場合がある。それを解決するためコンフリクトの検出及び解決が行われるが、本稿では省略する。

3.2 手動管理と自動管理

当初のレプリケーション拡張 HORB では、レプリカの生成及び、オリジナルへの反映、また破棄といったレプリカ操作は、プログラマが行うことにしていた。これは、従来のシステムとの親和性が重要である場合、計算機資源が乏しくレプリカを置くのが困難な場合、通信量をできる限り抑えたい場合などがあると考えたからである。

その一方、プログラマやユーザにレプリカ操作を意識させるのはかなりの負担になるのは確かであり、レプリケーションを自動的に行うのが望ましい環境 - モバイルコンピューティングなど - もある。レプリカ手動管理バージョンと自動管理バージョンの利点・欠点を表 1 にまとめる。

4 レプリカの自動制御

4.1 基本構成

前節の議論を踏まえ、レプリカをデフォルトで作成するバージョン (レプリカ自動管理バージョン) (図 7) の実装を行った。メソッド呼び出しによりオブジェクトの状態に変更があった場合には、そのオブジェク

レプリカ管理方式	可用性	構成	プログラマの手間	ユーザの視点	通信量
手動管理	低い	単純	多い	ある程度レプリカを意識しなければならない	必要最小限に抑えられる
自動管理	高い	複雑	無し	レプリカを意識する必要無し	無駄なトラフィックは不可避

表 1: レプリカ手動/自動管理の比較

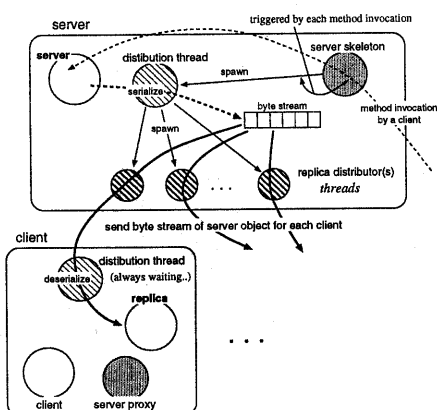


図 8: レプリカ自動配送の流れ

トを参照している全ての（その時点で配送可能な）クライアントに対しレプリカを自動的に配送する（図 8）。

4.2 接続・切断状態の判断

“レプリカ自動管理バージョン”の基本的な戦略は以下の通りである。

- クライアントはメソッド実行時に先立って常にネットワークの接続状態を確認するようにし、確認プロセスによりネットワークから切断されていると判断された場合には通信を行わず、レプリカに対してメソッドを実行する。
- サーバはレプリカを各クライアントに対して配送する際、それに先立ってネットワークの接続状態を確認し送信可能かどうかを見て、接続されていると判断したらレプリカを送信し、切れていると判断したら送信しない。

接続状態の定義

“レプリカ自動管理バージョン”では、クライアントは2つの遷移状態を持つ（図 9）。ネットワークの接続状態の確認及び状態の遷移はシステムによって自動的に行われる。

接続された状態 (connected) ネットワークが使用可能であると判断され、次に使用不可能と判断されるまでの間の状態

切断された状態 (disconnected) ネットワークが使用不可能であると判断され、次に使用可能と判断されるまでの間の状態



図 9: ネットワーク接続状態の遷移

接続状態の自動確認

今回の実装では UDP データグラムを用いて接続確認を行った。クライアント/サーバ双方で UDP データグラムを受け取って UDP データグラム（それぞれ 1 バイト）を返すだけのサーバを常時別スレッドで動かしている（図 10）。

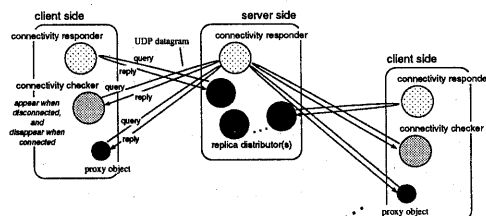


図 10: ネットワーク接続状態の確認

図 8 及び図 10 で示したスレッドについて以下に述べる。

Proxy (client) 側

● distribution thread

Proxy オブジェクトとはほぼ同じ機能を持つ、レプリカ配送を受け取るためのスレッド。機能的に重なる部分が多いので、Proxy オブジェクトのもう一つのインスタンスとして作成し別スレッドで動かす。

- **connectivity responder**

Skeleton 側からの UDP データグラムによる問い合わせに答えて 1 バイトの UDP データグラムを送り返すスレッド。これはレプリカの自動配送の際の確認用に用いられる。機能的に小さいので Proxy オブジェクトのインナー（内部）クラスとして定義した。

- **connectivity checker**

クライアント側システムがネットワークから切断されたと判断した場合に生成されるスレッド。ネットワーク接続が回復するのを確認し、確認でき次第消滅する。機能的に小さいので Proxy オブジェクトのインナークラスとして定義した。

Skeleton (server) 側

- **distribution thread**

Skeleton オブジェクトとはほぼ同じ機能を持つ、レプリカ配送を準備するスレッド。機能的に重なる部分が多いので、Skeleton オブジェクトのもう一つのインスタンスとして作成し別スレッドで動かす。

- **connectivity responder**

Proxy 側からの UDP データグラムによる問い合わせに答えて 1 バイトの UDP データグラムを送り返すスレッド。これはクライアント側からネットワーク越しのメソッド呼び出しを行う前に接続状態を判定する際に用いられる。機能的に小さいので Skeleton オブジェクトのインナークラスとして定義した。

- **replica distributor**

オリジナルオブジェクトに対するメソッド呼び出しに応じて distribution thread によって生成されるクライアントへのレプリカ自動配送用のスレッド。複数が並行して動作するので、レプリカは受け取れるクライアントからなるべく早く受け取れるようになっており、かつ他のクライアントに待たされることもない。バイトストリームを配送するだけの機能しか持たないので Skeleton オブジェクトのインナークラスとして定義した。

上記の説明をまとめ、“レプリカ自動管理バージョン”で生成されるクラスファイルを図 11 に示す。

5 試作アプリケーションによる評価

“レプリケーション拡張 HORB”の評価用アプリケーションとして、複数のクライアント間でデータが共有できる簡単なドロッププログラムを作成した。当然ながらネットワークから切断された状態でのディスコネクトオペレーション [5] が可能なツールとなっている。またレプリカとオリジナル間でコンフリクトが発生した際の処理も行うことができる。

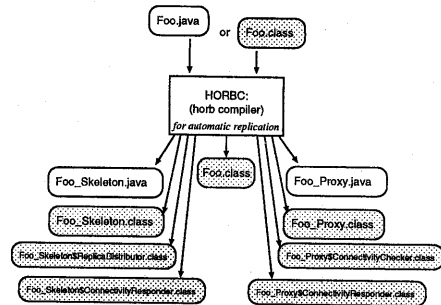


図 11: レプリカ自動管理バージョンで生成されるファイル

なお“レプリカ手動管理バージョン”と、システムが自動的にレプリカを扱う“レプリカ自動管理バージョン”の2種類を作成した。

5.1 ネットワーク版ドロッププログラム

このドロッププログラム（図 12）は、基本的には線画のみによるお絵描きソフトである。HORB によりサーバオブジェクト内にある描画データは複数のクライアント間で共有される（図 13）ようになっている。

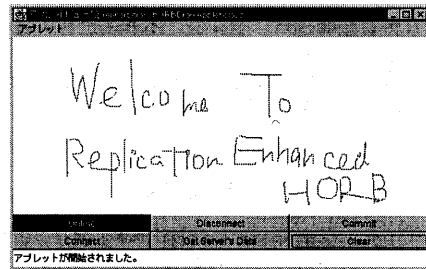


図 12: ネットワーク版ドロッププログラム（レプリカ手動管理バージョン）

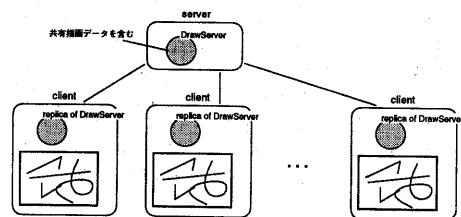


図 13: ドロッププログラムにおけるデータの共有

なおここで詳細は述べてないが、ドロッププログラムでのコンフリクトの扱いは簡単なものとした。

5.2 可用性に関する考察

“レプリケーション拡張 HORB”で作成することにより、ディスコネクテッドオペレーションに対応するとともに、ディスコネクテッドオペレーションによってレプリカ・オリジナル間のコンフリクトが発生した場合でも、システムによりコンフリクトの検出及び解決が自動的に行われるようになっている。

また“レプリカ自動管理バージョン”で作成した方は、不意のネットワークからの切断に耐え、かつネットワークに再接続した際には自動的に新しいデータが反映されるものとなっている。

5.3 プログラミングに関する考察

ディスコネクテッドオペレーション対応にするための、レプリカ自動/手動管理バージョンにおけるプログラミングの手間を比較する。

レプリカ手動管理バージョン

クライアントは、オン/オフラインモード切り替えボタンを作成し、ボタンを押すと `createReplica()` などのレプリカ管理メソッドが呼び出されるようにした。サーバは特に変更を必要としなかった（コンフリクトの扱いを簡単にしたため）。

レプリカ自動管理バージョン クライアントは何も変更しなかった。モード切り替え用のボタンなどもない。サーバも特に変更を必要としなかった（コンフリクトの扱いを簡単にしたため）。

6 レプリカ管理手法の比較

レプリケーション拡張 HORB の“レプリカ手動管理バージョン”、“レプリカ自動管理バージョン”それぞれについてまとめる。

レプリカ手動管理バージョン

レプリカをプログラマが手動で管理する場合は、レプリカがローカルにあるオフラインモードと、ローカルにはないオンラインモードの切り分けが明確であり、任意の時点でどちらかの状態にある。

オフラインモード中にメソッドを呼び出せば、それに伴った通信は起こらないので自動的にディスコネクテッドオペレーションとなる。当然、オフラインモード中はネットワークへの接続が全く無くて作業が実行できる。

Proxy オブジェクト経由でメソッドを呼び出すという点でインタフェースに違いはないので、クライアントプログラムの作成者はレプリカ操作メソッドを呼び出すことを除き、ほとんど通常のプログラムと同じように作成すれば、ディスコネクテッド対応のアプリケーションとなる。

また、オフラインモード中は必ずローカルな呼び出しとなるので、性能向上手段としても効果的に用いることができる。

レプリカ自動管理バージョン

レプリカを自動制御する場合、クライアントはレプリカの操作を全く行う必要がないので、通常の HORB と全く同様に記述すれば、ディスコネクテッドオペレーションに対応したクライアントプログラムができる。

プログラマはオリジナル・レプリカ間の一貫性が損なわれた際の処理 (Merge メソッド) を記述することを除き、ディスコネクテッドオペレーションを意識して作成する必要はない。特にクライアントは通常の HORB と全く同様にコーディングできる。

7 まとめ

モバイル/分散環境における可用性、信頼性、性能の向上を目指し、分散オブジェクトシステムにおいてリモートオブジェクトのレプリカをローカルに保持する手法を提案し、HORB にプロトタイプを実装した。更にレプリカの手動管理方式に加え、自動制御方式も実装し、サンプルアプリケーションにより比較・検討を行った。その結果、レプリカ自動管理バージョンでは、プログラマをほとんど意識させることなく、レプリケーション対応アプリケーションが作成できることを示した。

参考文献

- [1] S. Hirano, "HORB: Distributed Execution of Java Programs", *Worldwide Computing and Its Applications '97 (WWCA97)*, March 1997.
- [2] J. J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System", *ACM Transaction on Computer Systems*, Vol. 10, No. 1, pp. 3-25, February 1992.
- [3] Abdelsalam A. Helal, Abdelsalam A. Heddaya and Bharat B. Bhargava, "Replication Techniques in Distributed Systems", Kluwer Academic Publishers, 1996.
- [4] Sun Microsystems, Inc., "Java™ Object Serialization Specification", February 1997.
- [5] K. Mani Chandy, Joseph Kiniry, Adam Rifken, Daniel Zimmerman, "Webs of Archived Distributed Computations for Asynchronous Collaboration", *Journal of Supercomputing*, 11(3), August 1997. available at <http://www.infospheres.caltech.edu/papers/archive/archive.ps.gz>