システムソフトウェアと
オペレーティング・システム 77－6
マルチメディア通信と分散処理 87－6
（1998. 2. 26）

レプリケーション拡張HORBでのレプリカ間の一貫性管理

山口 実靖　若狭 建　相田 仁　齊藤 忠夫
東京大学工学部電子情報工学科

Java上の分散オブジェクト環境HORBに対して サーバオブジェクトのレプリカをクライアント側に作成する拡張を行った。レプリカを使うことによりディスコネクティッドオペレーションを可能とした。
そして、ディスコネクティッドオペレーションにおける楽観的コンシステンシ管理およびコンフリクトの検出・解決システムを実装した。
コンフリクト検出はレプリカ作成時のサーバのバージョンと再コネクト時のサーバのバージョンの比較により行う。また、ディスコネクト中にレプリカに対して行った処理はログに記録しておき、コンフリクト発生時にはレプリカを破棄しオリジナルに対してログを再実行することによりコンフリクト解決を行う。

Consistency Management between Replicas for Replication Enhanced HORB

Saneyasu Yamaguchi　Ken Wakasa　Hitoshi Aida　Tadao Saito
Department Information And Communication Engineering,
Faculty of Engineering, University of Tokyo

We enhanced HORB, a distributed object system on Java, to enable to create a replica of server objects on client machines. Disconnected operation can be done by using this replica. We also implemented optimistic consistency control system and conflict detection/resolution system. The way to detect conflict is to compare the version number of the server object when replica was created with the one after reconnecting . Conflicts are resolved by remembering the operations done after disconnection and redoing it to the original object after reconnection.

1　はじめに

ネットワーク上に多くの情報が分散している今日、学校、企業などの枠組みを超えた情報の共有化がかつてなかったほど必要とされている。それにともない、ネットワーク上で動くアプリケーションの規模はどんどん拡大し、分散アプリケーションを開発するコストも非常に大きくなってしまった。そこで少ない労力で大規模な分散アプリケーションを開発する方法として現在注目されているのが分散オブジェクトシステムであり、多くのベンダーから 多くの製品が出ている。

しかし、今後モバイル環境などで重要となるであろうディスコネクティッドオペレーションを実現しているシステム [1] はまだ少ない。

そこで本研究ではJavaで書かれた優れた分散オブジェクトシステムであり、現在多くの人によって使われている HORB[2, 3, 4, 5] を取り上げ、レプリケーション[6] 拡張を施しディスコネクティッドオペレーションを実現した。そして、その際問題となるコンフリクト検出およびその解決についてレプリケーションに注目して述べる。

2　レプリケーション拡張HORB

HORBは図1の様なクライアントとサーバをProxyとSkeletonが繋ぐ構造をしている。

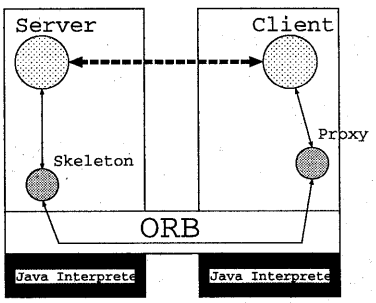

図 1: HORB

そこで、図2の様にサーバオブジェクトのレプリカを一時的にローカルホスト上に作る。このレプリカに対してメソッドを実行すれば サーバホストとの通信を必要

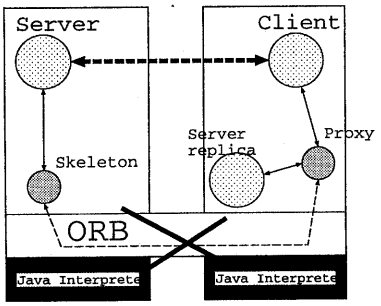

図 2: レプリケーション拡張 HORB

とせずディスコネクティッドオペレーションを実現できる。

ただし、そのオペレーションはサーバオブジェクトのレプリカで可能な操作に限られる。サーバの資源にアクセスする操作などをディスコネクトで実行することは不可能である。

2.1 実行環境

horbc コンパイラのみに変更を加えたので 実行環境は通常の HORB と全く同じであり、通常の HORB で実行することができる。

2.2 楽観的レプリカ管理方式

ロック方式としては" 楽観的な" ロック方式のみを用い "悲観的な" ロック方式は採用しなかった。" 楽観的な" ロック方式を用いたの理由は以下の通りである。

- 悲観的な手法を用いると、ロックが掛かっている間は他のクライアントのオブジェクトへのアクセスが制限される。
- あるクライアントがロックを掛けたままネットワークを離れてしまうと、ロックが解除されない状態が長く続いたり、外されないといった事態を招き、システムの可用性を著しく低下させる。
- ネットワークから切断された状態ではロックを掛けることができない。
- モバイル／分散環境では、悲観的な方式を採用することによる可用性低下の影響の方が、楽観的な手法を採用することによるコストよりも大きいという報告が多い [7]。
- 楽観的なレプリカ管理手法を用いることにより一貫性が崩れてたとしても、実際上は何らかの方法を用いて解決できることが多い。

楽観的ロック方式によりディスコネクティッドオペレーションを行った場合は、ネットワーク再接続時に変更をサーバに対してコミットしなくてはならない。その際にコンフリクトが生じることがある。

2.3 拡張した機能

ディスコネクティッドオペレーションの実現にあたって以下の機能を付け加えた。その際、何を変更したかを括弧内に示す。

- サーバのレプリカをローカルに作成する。(*_Proxy,*_Skeleton)
- サーバオブジェクトへのメッセージを [ORB で転送 / レプリカで処理] を切り替える。(*_Proxy)
- ディスコネクティッドオペレーションからネットワーク再接続の コンフリクト検出を行う。(*_Proxy, *_Skeleton)
- コンフリクト発生時に、矛盾しているオブジェクトをマージする (*_Proxy)。

これらの拡張を行うにあたって Proxy, Skeleton の作成を行う horbc コンパイラに変更を加えた。

2.3.1 サーバオブジェクトのレプリカの作成

サーバオブジェクトをシリアライズしてローカルにコピーする。

これを実現するために 通常の HORB と比べて以下のことをしなくてはならない。

- サーバオブジェクトは java.io.Serializable を実装する。
- サーバのクラスファイルをローカルにもコピーする。

また JDK のシリアライザを用いたため、JDK1.1 以降でないと動作しない。

2.3.2 サーバオブジェクトへのメッセージの振り分け

ネットワークに接続されている時は サーバオブジェクトへのメッセージは ORB がリモートサーバに転送し、ネットワークから切断している時は サーバオブジェクトへのメッセージは レプリカが処理することになる。

サーバオブジェクトへのメッセージの振り分けは Proxy オブジェクトが行う。これにより、クライアントは ネットワーク接続時と全く同じインタフェースでディスコネクティッドオペレーションを行うことができる。

振り分け先の変更は プログラマが手動で行うバージョンと、ネットワーク接続状況により自動的に行うバージョンを用意した。

3 コンフリクト検出

コンフリクト条件 次の二つの条件が共に成り立つ場合にはコンフリクトが生じていることになる。

(1) ディスコネクト中にクライアントが何らかのオペレーションを行った。

(2) クライアントがディスコネクト中にサーバに何らかの変更が施された。

まず、(1) を検出するには、クライアント側でダーティーフラグを用意し、クライアントがディスコネクティッドオペレーションを行いレプリカに変更を施したときはこれを ture とする。

次に、(2) を検出するには クライアントがレプリカを作成してネットワークからディスコネクトするときにサーバのバージョンを記録しておく。クライアントがネットワークに再接続したとき、サーバのバージョンがクライアントの記録しておいたものと同じであればサーバには変更が施されていない。

サーバのバージョン管理 サーバのバージョンは Skeleton が管理し、サーバのメソッドが実行される度に version は上昇する。

ReadOnly メソッド メソッド名が _ReadOnly で終わるメソッドを実行した場合は Server の version は上昇せず (図 3)、ダーティーフラグも立たない (図 4)。変更を施さないメソッドの場合 version を上昇たり、ダーティーフラグを立てる必要がないので、ReadOnly メソッドとすることで不要なコンフリクト検出を避けることができる。

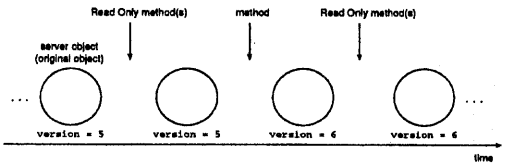

図 3: オリジナルサーバオブジェクトへの Read Only

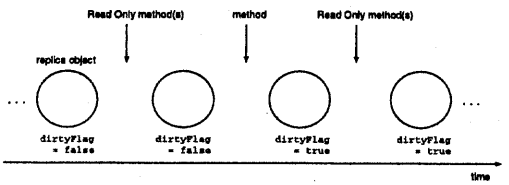

図 4: レプリカへの Read Only

コンフリクト検出例 図5の例では、クライアントはサーバオブジェクトが version 5 のときに レプリカを作成し、ディスコネクトしている。よって、クライアントが知っている最も新しいサーバオブジェクトは version 5 である。また 切断直後は レプリカのダーティーフラグは立っていない。

切断中にクライアントがレプリカに対して何らかの変更を施し、ダーティーフラブが立つ。

一方、オリジナルのサーバオブジェクトにも、変更が加えられて version 9 となる。

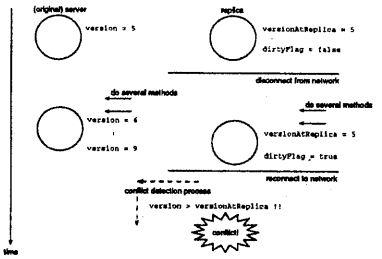

図 5: コンフリクトの検出

この状態で クライアントがネットワークに再接続しコミットを試みると、「ダーティーフラグが立っており」かつ「サーバのバージョンがクライアントの知っているものよりも上がっている」ので コンフリクトとなる。

4 コンフリクト解決

コンフリクトの解決として次の方法を提案する (図 6参照)。

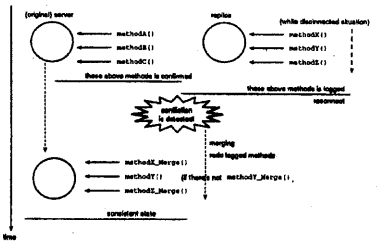

図 6: コンフリクトの解決

- クライアントはサーバのバージョンを記録し、ダーティーフラグを false として、ネットワークからディスコネクトする。
- クライアントは レプリカを用いたディスコネクティッドオペレーションを行うにあたって、行ったメソッドをログに保存しておく。
- クライアントがネットワークに再接続したとき、クライアントがダーティーでかつサーバのバージョンがクライアントの記録していたのものより上がっているとコンフリクトの発生である。
- クライアントはオリジナルのサーバオブジェクトを一時的に受け入れ、そのオリジナルオブジェクトに対してディスコネクト中に行ったメソッドを再実行する。再実行によりディスコネクト中に行った処理をマージする。

これにより、クライアントがディスコネクト中に行った処理は失われない。

しかし、サーバとクライアントの間のずれは解消できるが、メソッドを単純に再実行するのではユーザの実行時と再実行時のオブジェクトの状態が異なり 意図した結果になるとは限らない。そこでマージ用メソッドの定義を提供する。ログの `foo()` を 再実行する場合 `foo_Marge()` を行う。`foo_Marge()` は `foo()` が再実行される時に矛盾を解決するよう定義することが出来る。ただし、`foo_Marge()` が存在しないときは `foo()` をそのまま実行する。

4.1 メソッドログの作成

ディスコネクティッドで行ったメソッドを再実行するために、ディスコネクティッド中に行ったメソッドをすべて記録しておく。

ネットワーク切断時に Client が Proxy オブジェクトに対してメッセージを送ったら (メソッドを呼び出したら) Proxy オブジェクトは レプリカに処理を依頼する前に 送られたメッセージを記録する。記録する内容は以下のようになる。

- 呼び出したメソッド
- そのメソッドに渡した引数の値

4.1.1 メソッドの記録

改造した horbc が Proxy を作成する時に、メソッドに ID を割り当てる。Proxy クラスに変換表を内蔵させ、ID を記録する。

4.1.2 引数の記録

引数は そのメソッドが呼ばれた時の値を保持する。つまり、渡された値へのポインタではなく 呼び出されたときの値をコピーして保持する。

基本型 引数が基本型の場合は その値を保持する。再実行時は メソッドが呼び出されたときの値が使用される。

参照型 引数が参照型の場合は 引数に渡された参照を保持するだけでは 参照先がその後変更される可能性があるので、コピーをしてそれを保持する。

参照先が 別のオブジェクトへの参照を保持している場合は その値もコピーしなくてはならないので コピーには `clone()` ではなくシリアライザを用いる。よって、メソッドに渡すクラスは Serializable を実装する必要がある。(ただし、インスタンスがシリアライズ可能でないときは諦めて `clone()` を用いる。Cloneable でもない時は 参照をそのまま保持する)

5 コンフリクト解決の実際

5.1 ネットワーク版同時編集可能共有エディタ

このエディタ (図 7) は、入力、削除、画面表示というエディタの最低限の機能のみを持つサンプルアプリケーションである。ネットワークに対応しており、同時に複数のクライアントが同じ文章に対して編集操作を行うことができる。

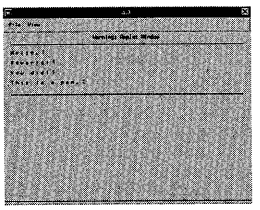

図 7: ネットワーク版エディタ

エディタでの入力及び削除メソッドは、カーソル位置への文字入力及びカーソル位置の文字削除という 2 種類のみの簡単なものである。それぞれのメソッドについて、対応する Merge メソッドを定義した。

5.1.1 ディスコネクティッドオペレーション対応によるデザインへの影響

テキストエディタをディスコネクティッドオペレーション対応にするにあたっての留意点は Merge メソッドの作成のみであることが望まれるが実際にはその他にも幾つかの留意点があった。それを以下に示す。

行ごとにバージョンを保持する システムはサーバオブジェクトのコンフリクトを検知できるが、それ以下の粒度の矛盾は検知及び解決を行うことができない。

そこで、具体的には各行に単調増加するバージョンを付け、入力及び削除メソッドが実行されると該当行のバージョンをインクリメントするようにした。更に各メソッドに引数を 1 つ増やし、どの対象行のいくつのバージョンの時のメソッドなのかを指示するようにした。

これにより、メソッドログを再実行するときに 自分が編集しようとしている行が ネットワーク切断中に変更されたか否かを調べることができる[1]。

各行に固有の ID を設定する ネットワーク切断中に編集した行の相対的位置が (行の挿入、削除などにより) 再実行時には ずれている可能性があるために 各行を固有の ID を割り当て、ID を用いて行を指定する。

Merge メソッドの作成

文字の挿入、削除のマージ Merge メソッド内では、再実行により編集を加える行のバージョンをチェックする。メソッドが対象としている行のバージョンが一致しているなら、行単位で矛盾が起きていないので、そのままメソッドを再実行しマージを行う (図 8)。

行単位で矛盾が発見された場合は、オリジナルの該当行及びレプリカの該当行をユーザに対してダイアログ

[1] 例えばログが「version 4 である行 34 に対する編集」である場合、再実行時に行 34 が version 4 であるかを調べればよい

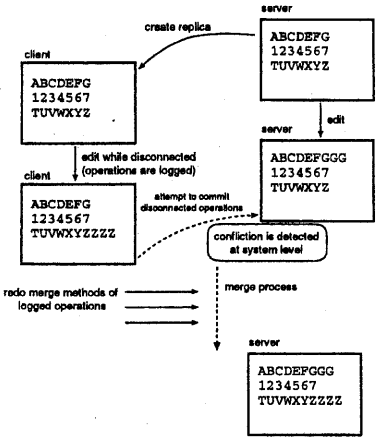

図 8: エディタでのコンフリクトの解決 (Merge メソッドが定義した矛盾が発見されなかった場合の例)

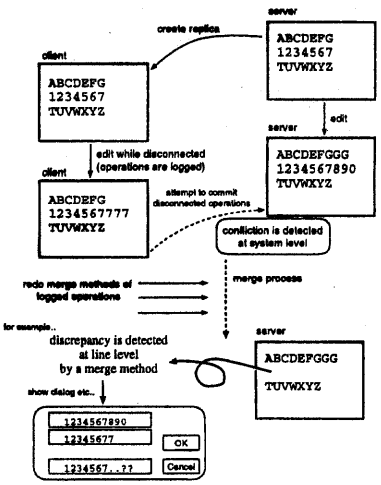

図 9: エディタでのコンフリクトの解決 (Merge メソッドが定義した矛盾が発見された場合の例)

ボックスで示し、無理矢理マージを行った行をテキストボックス内に表示してユーザに対し整合性を取るようその行の編集を要求するようにした (図 9) 。

行の新規作成 ネットワーク切断時に行を新規作成した場合 その行に対する ID はレプリカにより割り振られることになる。これと同じ ID がサーバで割り振られないためには ID は 32bit の乱数を用いた。

対象とする行が存在しないとき 再実行時に、対象とする行が (削除されてしまったなどで) 存在しない場合の解決策としては以下の二つが考えられる。

破棄 削除されてしまった行への変更は破棄する。あるいは、(挿入の場合は) 行の選択を破棄し臨時に文末などに挿入する。

行のリスト構造の永久保存 テキストデータを双方向リスト構造で保存すれば削除されてしまった行もかつてあった場所を調べることが出来る。

図 10において、行 24 が削除されると行 35 と行 66 が参照しあうことになる。ここで行 24 の参照のみを残しておけば行 35 や行 66 アクセスすることができる。

さらに行 35 が削除された場合はさらにその前を辿って行けば削除されていない行にたどり着くことができる。なぜなら 挿入・削除だけでは 順序は保たれ、前へ辿って行けば必ず先頭にはたどり着くからである。

しかし、これを実現するためには削除された全ての行のリスト構造を保存しておかなくてはならない。

今回のサンプルエディタでは前者の破棄を採用した。

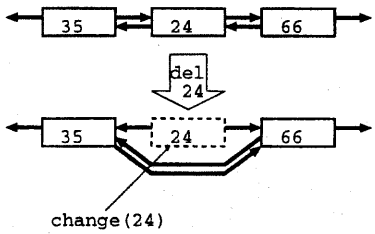

図 10: リスト構造の保存

サンプルコードは図 11のようになる。

6 評価

以上の様にレプリケーション拡張 HORB のコンパイラを用いることにより、通常のデータ共有テキストエディタにわずかの変更を施すだけでディスコネクティッドオペレーション対応のテキストエディタを作成することができた。

6.1 システムで解決困難なこと

例に示したテキストエディタのように共有するデータが相対的位置の意味が強い場合は特別の処理を施さなく

```
public void insert(char ch, int lineID,
                   int offset, int lineVersion){
    data.insert( ch, lineID, offset);
}
public void insert_Merge(char ch, int lineID,
                    int offset, int lineVersion){
    if( id2Line(lineID) == null ){
        // lineID が示す行はもはや存在しない
        return;
    } else if( id2Line(lineID).getVer() != version ){
        // コンフリクト発生
        コンフリクト処理
        return;
    } else {
        // コンフリクトせず
        data.insert( ch, lineID, offset);
        return;
    }
}
```

図 11: コーディングサンプル

てはならない。ディスコネクティッドオペレーションの実行時と、ログの再実行時で絶対的位置がずれている可能性があるからである。

単位ごとに 固有の ID を割り振ることによりこれを解決することが出来る。

7 今後の課題

コンフリクト解決策としてマージ用メソッドを用意することを提案したが、ログ上のメソッドに対応するマージ用メソッドを実行するだけでは解決が困難な場合も多く、更に良い解決策について考察をする。

また、ログ上のメソッドを再実行するにあたってメソッドからの返り値は現在利用していないので、有効的な利用方法について考察する。

今回のシステムでは write-write コンフリクトの検出及び その解決のみ行われ、read-write コンフリクトへの対処はされていない。read-write コンフリクトへの対処もする必要がある。

メソッドのログの再実行の前後に実行されるフックメソッドの指定を提供する。

8 まとめ

HORB に対してレプリケーション拡張を施し、HORB アプリケーションでディスコネクティッドオペレーションを可能とした。

そして、本拡張におけると楽観的なレプリカ間の一貫性管理方式 及び コンフリクトの検出、解決方法について説明した。

サンプルプログラムをあげ、コンフリクト解決の実例を示した。

また、本拡張では解決困難な問題とその解決策に付いて述べた。

参考文献

[1] Anthony D. Joseph, Joshua A. Tauber and M. Frans Kaashoek, "Mobile Computing with the Rover Toolkit", *IEEE Transactions on Computing*, pp. 337-352, March 1997.

[2] S. Hirano, "HORB: Distributed Execution of Java Programs", *Worldwide Computing and Its Applications '97 (WWCA97)*, March 1997. available at `http://ring.etl.go.jp/~hirano/wwca97/index.html`

[3] HIRANO Satoshi "The Magic Carpet for Network Computing" http://ring.etl.go.jp/openlab/horb/, 1997

[4] 中原 真則, 平野 聡, "飛べ、オブジェクト！ HORB プログラミングマジック", bit vol.28 no.10,11, 共立出版, 1997.

[5] "Java^{TM} によるオブジェクト指向技術セミナー 忘れまい阪神大震災" 配布資料

[6] David A. Nichols, Pavel Curtis, Michael Dixon and John Lamping, "High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System", *in Proceedings of ACM Symposium on UIST '95 Pitt. PA.*, November 1995.

[7] Douglas B. Terry, Marvin M. Theimer, Karin Peterson, Alan J. Demers, Mike J. Spreizer and Carl H. Hauser, "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Strorage System", *in Proceedings of the 15th ACM Symposium on Operating System Principles SOSP-15), Copper Mountains Resort, Colorado*, December 1995. available at `http://www.parc.xerox.com/csl/projects/bayou/pubs/sosp-95/BayouConflictsSOSPPreprint.ps`