レプリケーション拡張HORBでのレプリカ間の一貫性管理

6E−4 山口 実靖 若狭 建 相田 仁 齊藤 忠夫

東京大学工学部

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[1] を取り上げ、レプリケーション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。

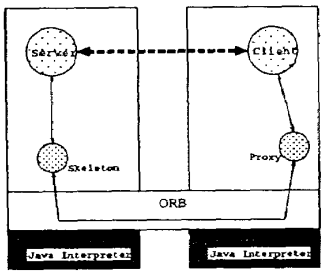

図 1: HORB

そこで、図2の様にサーバ オブジェクトをシリアライズしてローカルホストにコピーし、サーバ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。

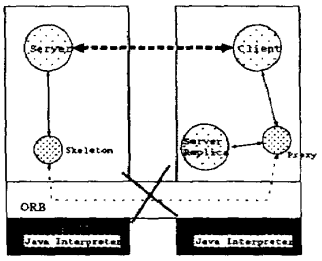

図 2: レプリケーション拡張HORB

3 ディスコネクティッドオペレーションのコンフリクト

ディスコネクティッドオペレーションの結果として次の4通りがある。

1 レプリカ、オリジナルとも変更なし
クライアント、サーバとも以前の状態を維持する。

2 レプリカ変更なし、オリジナル変更
クライアントが変更されたサーバオブジェクトを受け入れる。

3 レプリカ変更、オリジナル変更なし
クライアントが変更を加たレプリカをサーバに対してアップロードし レプリカを採用する。

4 レプリカ、オリジナルとも変更
オリジナルへの変更と、レプリカへの変更が衝突(コンフリクト)をしている。

3.1 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。

3.2 コンフリクト検出方法

(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[1]。

[1]読み込み専用のメソッドの実行は関係しない

(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ティッド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コネクトしたとき、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。

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3参照)。

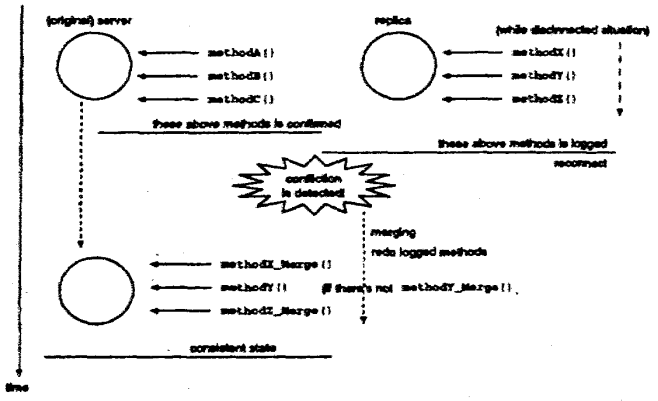

図 3: コンフリクトの解決

- 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、行ったメソッドのログを保存しておく。
- クライアントがネットワークに再コネクトしたとき、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サーバの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、これにより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。

例えば、「銀行の口座から 10 万円下ろす」メソッドに対応するメソッドは再実行時に口座の残高が 10 万円以下の危険性を考えなくてはならない。

```
void minus(int dif){
        money -= dif;
}
void minus_Marge(int dif){
        if( money >= dif ){
                money -= dif;
        } else {
                perror("message...");
        }
}
```

5 今後の課題と問題点

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。

6 まとめ

分散オブジェクトシステム HORB に対してレプリケーション拡張を施し、ディスコネクディッドオペレーションを実現した。そして、ディスコネクティッドオペレーション時のサーバ、クライアント間のコンフリクトの検出およびその解決策を示した。

参考文献

[1] Satochi Hirano, "Prelimirary Performance Evaluation of a Distributed Java:HORB", available at http://ring.etl.go.jp/~hirano/obpdc97/obpdc97.ps

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

[3] 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.