モーバイルコンピューティング 4－9
（1998. 2. 13）

オブジェクト同期化モデルを用いた無線ネットワークでのデータ転送方式

黒田 正博† Luosheng Peng‡ 渡辺 尚† 水野 忠則† 下間 芳樹§
†静岡大学 ‡米国三菱電機情報技術センター
§三菱電機(株)

本論文は、楽観的オブジェクト同期化モデルを、ネットワークを意識しない「いつでも、どこでも」グローバル情報サービスのインフラに適応するための、低転送レート下でのデータ転送方式と考慮すべき点を述べる。モデルとするシステムは、常にデータの更新を図っているプライマリ・サーバ群、プライマリ・サーバに接続した時にデータ同期を行うセカンダリ・サーバ群、情報の取り出しを行うクライアント群からなっている。その中で無線ネットワークでの通信を想定したクライアントとセカンダリ・サーバ間のデータ転送について、本オブジェクト同期化モデルのアプリケーション/データオブジェクト/データストリームそれぞれの層での最適化を述べる。

Data Transfer Architecture of Optimistic Data Consistency Model

Masahiro Kuroda† Luosheng Peng‡ Takashi Watanabe†
Tadanori Mizuno† Yoshiki Shimotsuma§
† Shizuoka University
‡Mitsubishi Electric Information Technology Center America
§ Mitsubishi Electric Corporation

This paper proposes a data transfer architecture and its data transfer optimization consideration suitable for an optimistic data consistency model which evolves the existing optimistic consistency scheme. This model consists of three components: primary servers, secondary servers, and clients. Primary servers form an infrastructure core for systems based on this model. Secondary servers duplicate data from primary servers using the optimistic consistency model. Clients are entities for information retrieval from primary/secondary servers. This data transfer architecture is layered into three layers in order to achieve efficient data transfer for data synchronization depending on the data characteristics in each layer. The separation of parameters for efficient data transfer from this data synchronization will add this model flexible feature for "anytime anywhere" mobile network computing.

1 はじめに

最近の無線ネットワークインフラの整備、ソフトウェアインフラの共通化、さらにネットワーク利用機器の低価格化により、「いつでも、どこでも」グローバル情報サービスを提供する環境が揃い始めている。インターネット/イントラネットあるいは無線ネットワーク環境において、最新の

データをローカルにアクセスするためには、複数サイトに分散したデータの整合性を確保するモデルが必要になる。そのモデルとして楽観的あるいは悲観的データ同期モデルがあるが、ここでは無線環境を意識した楽観的データ同期モデルを採用している。このモデルは以下の４つのキー要件が必要である。

(1) データ整合性と利用性を提供するインフラ
(2) 非接続オペレーションのサポート
(3) 同期化でのデータバージョン管理
(4) データ同期化のユーザコントロール

データ整合性(Data Consistency)とデータ利用性(Data Availability)に関しては、楽観的整合性(Optimistic consistency)と悲観的整合性(Pessimistic consistency)の２つのスキームがあるが、サーバ非接続状態での操作を考えた場合、悲観的整合性は適していない。また、Coda[1]や Ficus[2]は、楽観的整合性を実現しているが、基本的に LAN 環境の信頼性及びネットワーク性能が高い環境を前提としている。さらに、Coda は Unix ファイルシステムをベースにしており、データのバージョン管理やデータ同期タイミングのコントロールは持っていない。

ここで提案するデータ同期化モデルは、上記の４要件を満たすものである。プライマリ・サーバ群からなるコア部があり、Coda などで提案された従来の楽観的データ同期スキームを用いてプライマリ・サーバ間のデータ整合性を保ち、コア部以外のデバイスにはデータ利用性を提供することを狙っている。

プライマリ・サーバ群は信頼性の高い、高速ネットワークで接続されているが、これに対してセカンダリ・サーバ群と呼ばれるデバイスはローカルにデータを保存し、他のプライマリ/セカンダリ・サーバ群あるいはグループとして定義されたサーバ群と自由に同期を取る事ができる。

典型的なプライマリ・サーバのグループとしては、サーバ群をクラスタ構成したものが考えられる。その外にセカンダリ・サーバがあり、コア部にあるクラスタと同期を取る。その他のサーバは、セカンダリ・サーバとの間でレプリカを作り、同期を取る構成である。

また、クライアントはデータを取り出すあるいは情報を見る端末であり、プライマリ・サーバあるいはセカンダリ・サーバからデータアクセスを行うアプリケーションやユーザを意味する。このクライアントは、セカンダリ・サーバとの間でデータ同期の初期化を行う。

データバージョン管理については、すべてのサーバに VersionVector[3]というデータ構造を定義する。この VersionVector は、セカンダリ・サーバがインフラコアにあるプライマリ・サーバとの間でレプリカを作成したり、データ同期時にその違いを吸収(reconcile)するのに使われる。

本論文では、このデータ同期モデルの中でのデータ転送方式について議論する。このモデルのコンポーネントアーキテクチャ、セキュリティ、不安定なネットワークでの同期方式、については今後議論していく事とする。

2 アーキテクチャ

本データ同期モデルは楽観的データ整合性モデルをベースにしており、３つのコンポーネントから成っている。

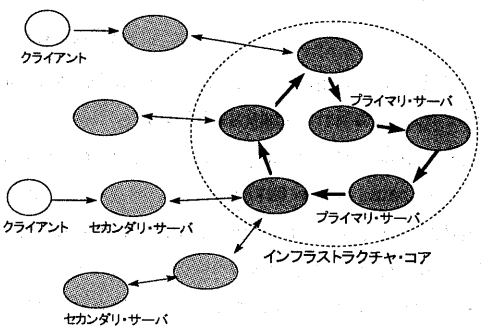

図1 アーキテクチャ

(1) プライマリ・サーバ
LAN など高速ネットワークで接続し、外部に対し

てデータ及びサービスを提供する。これに属するサーバでインフラ・コアを構成し、外からは1つのサーバと見せるために Peer-To-Peer データ同期プロトコルにより常にデータを最新にしている。これを Reconciliation と呼び、例えば、時間周期、データ更新/ネットワーク接続のイベント、コールバック要求によりトリガがかかる。このポリシーは初期化時のコンフィギュレーションで行う。

(2) セカンダリ・サーバ

信頼性及び転送性能の異なるネットワーク構成で、ローカルなデータレプリカを持ち、必要な時にインフラ・コアに接続しデータの更新ができる。このサーバは無線ネットワーク上にあることも想定しており、常にプライマリ・サーバとのデータ同期ができるわけではない。Reconciliation もアプリケーションあるいはユーザによりトリガをかけることになる。

(3) クライアント

インフラ・コアあるいはセカンダリ・サーバにデータアクセスをすることができる。情報消費のためのデバイスにあたる。

このアーキテクチャでは、上記3つの構成要素から成っているが、これは論理的な単位を示している。例えば、高機能のモバイル端末はクライアント機能とセカンダリ・サーバ機能を持っている。

3 データ同期化モデル

データ同期化モデルは、MNCRS(Mobile Network Computer Reference Specification)[4]標準仕様策定会議で策定中の Data Consistency API を基本とする。本機構は、複数のサイトでレプリカを持ち、それらのレプリカ間で整合性を保つ方式を採用している。これは、地理的に離れた場所にあるレプリカの整合性を取ることを前提にしており、楽観的データ同期機構を採用している。

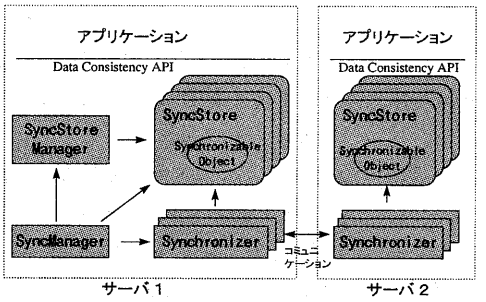

図2 プライマリ/セカンダリ・サーバの構成要素

本モデルを Java[5]技術を用いて実現したものをオブジェクト同期化モデルと呼んでいる。以降すべての記述は Java 用語で記述している。本モデルは、アプリケーションから見れば、データを保存したり取り出したりするコンテナとしての SyncStore、その SyncStore に入れるデータとしての Synchronizable な Java オブジェクトからなっている。SyncStore に Java オブジェクトを投入あるいは取り出す場合、put()/get()メソッドでアクセスする。また Java オブジェクトの更新には reconcile メソッドを用い最新のデータにする。

実際のデバイスでは、SyncStoreManager の Create()メソッドを呼び出す事により複数の SyncStore を生成することができる。1つのデバイスには1つの SyncStoreManager があり、そのメソッド Create()を実行することにより生成したすべての SyncStore を管理する。Create()メソッド以外には、現在ある SyncStore 内の Object の lookup(), listing(), delete() などの操作ができる。これら SyncStore は、それぞれ唯一の SyncStore の ID を持っており、このモデル内で対象とする SyncStore を参照する時に利用する。

Synchronizer は、1対の SyncStore 内のオブジェクト群を同一状態に保つための処理を行うキー要素である。1つのデバイスには複数の SyncStore があるが、それぞれの SyncStore 対は異なったネットワークプロトコルを利用することができる。

例えば、Synchronizer1 はネットワークの信頼性

が高く転送速度が高いため、TCP/IP 上の双方向コミュニケーションプロトコルを用いるが、Synchronizer2 はネットワークが不安定で転送速度が低い無線パケット網の場合、片方向のメッセージングプロトコルを利用するといったネットワーク環境に最適なコミュニケーション方法を採用することができる。この Synchronizer は、SyncStore 間の同期を行う時に SyncManager によりダイナミックに生成され、同期が終了すると消滅する。この SyncManage は相手の SyncManager との間で利用可能ネットワーク情報を交換し、ネットワークの性質に最適な Synchronizer ペアをそれぞれのデバイス上に生成する。

4 データ転送方式

データ同期として、異なる Java Virtual Machine 上の２つの SyncStore の内容を比較し、その内容が同一になるようにデータの送受信を行い、そのデータの変更を SyncStore 内のそれぞれのデータオブジェクトに反映する方式を採用している。

4.1 End-to-End 同期フロー

アプリケーションのデータ更新などにより SyncStore1 が相手の SyncStore2 と同期つまりデータ転送するときの典型的な End-to-End フローでの基本サイクルを考える。なお、SyncStore i (i=1,2) に対応して SyncManager, Synchronizer をそれぞれ SyncManager i, Synchronizer i と呼ぶ。

(1) SyncManager1 は、相手となる SyncManager2 に対して、その SyncStore2 と Synchronizer2 を指定してデータ同期要求を送る。
(2) SyncManager1 と SyncManager2 はそれぞれ対応する Syncronizer1 および Syncronizer2 を生成し、それらを接続する。
(3) Synchronizer1 は SyncStore1 の持っているオブジェクト群のバージョン情報 VersionVector を Synchronizer2 に送る。この VersionVector とは SyncStore の現在の状態を記述したデータ構造である。
(4) Synchronizer1 から VersionVector を受信した Synchronizer2 は、それ自身が現在持っている VersionVector と受信した VersionVector を比較し、その中で SyncStore2 にすでにロギングされているが SyncStore1 にロギングされていない変更情報を取り出す。
(5) Synchronizer2 は上記(4)で取り出した変更情報を SyncStore2 の VersionVector と共に Synchronizer1 に送信する。
(6) Synchronizer1 は、SyncStore1 の put()メソッドを用いて、受信したアップデートを SyncStore1 に反映する。

SyncStore の Put()メソッドは、それぞれのオブジェクトの reconcile()メソッドを呼びだし、更新データをそのオブジェクトにマージする。そのマージが終了すると同時に、その SyncStore の VersionVetor を更新する。なお、Synchronizable オブジェクトの reconcile()メソッドは、1 対の SyncStore での同時データ更新を検知し、それを解決するためのアプリケーション依存のコンフリクト解決手段を持っている。reconcile()メソッドは interface 定義する。

4.2 VersionVector

SyncStore は、アプリケーションがオブジェクトに対して行った変更内容を示すログを保持しており、下記情報が記録されている。
・ログを生成した SyncStore のレプリカ Id
・ログが生成された時刻
ある１つの SyncStore 上において、
(a) レプリカ Id
(b) SyncStore 上に存在する(a)で表される SyncStore で生成されたログの中で、最新の時刻を持つログの時刻
の組((a), (b))を、システム上で使用されているすべての SyncStore のレプリカ Id に対して集めたものを VersionVector と呼ぶ。SyncStore の VersionVector は、ある SyncStore が各レプリカ Id で表される

SyncStore で生成されたログのうち、それぞれの SyncStore のどの時刻までのログを保持しているかを表したバージョン情報である。

2つの SyncStore が同期を行うとき、2つの SyncStore はお互いの VersionVector を交換する。SyncStore は、自分の VersionVector と相手の VersionVector を比較することによって、

(1) 自 SyncStore が持っているログのうち、相手 SyncStore が持っていないもの

(2) 相手 SyncStore が持っているログのうち、自 SyncStore が持っていないもの

を検出することができる。SyncStore は、上記(1)に該当するログを抽出して相手方に送信する。同時に、(2)に該当するログを相手方から受信し、受信したログをオブジェクトに適用し、ログを保存する。また、上記(1)、(2)がともに存在するときには、2つの SyncStore の間に衝突が存在する可能性があることが検出される。この衝突の解決方法はアプリケーション依存であるため、アプリケーションが行う。

以上によって、同期が終了した時点で2つの SyncStore にはまったく同じログが存在することになり、2つの SyncStore の内容が同一であることが保証される。

ログおよび VersionVector の保持する時刻は必ずレプリカ Id と対応づけられ、時刻の比較は同じレプリカ Id に関してしか行われない。したがって、各 SyncStore はそれぞれの SyncStore 内でのみ通用するローカルな時刻を使用することが可能である。また、ログがレプリカ Id と関連付けられているため、同一のオブジェクトに対する同時更新による衝突が、容易に検出できる。

5 データ転送方式の最適化

オブジェクト同期化モデルのデータ転送最適化では、データオブジェクト処理の最適化が中心に研究されているが、本論文では、システムとしての最適化から考えていく。アプリケーション、データオブジェクト、データストリーム転送、それぞれのレイヤで最適化を行い、トータルとして最適な転送を実現することが狙いである。これは、対象であるセカンダリ・サーバが、さまざまな環境で利用されることに依っている。

無線ネットワーク環境では、LAN 環境に比較して低いデータ転送性能とネットワーク通信路の不安定さから来る実効データ転送性能の低下がある。このような環境では、転送する情報そのものを削減することと一連の処理を並行的に行いトータルとして処理時間を短くする方法がある。

(1) アプリケーション・レイヤのデータ簡略化

アプリケーションがその簡略化方法を知っているデータそのものの圧縮としては、例えば従業員名のコード化やホストアドレスがある。Serializable オブジェクトの SyncStore への投入あるいは取り出し時の put()/get()メソッドでデータオブジェクトの encode()/decode()メソッド実行によりデータ自身の簡略化ができる。

(2) データオブジェクト・レイヤのデータ量削減

データオブジェクトレベルの圧縮は、オブジェクトのストレージフォームに依存する。例えば、オブジェクトそのものを転送する方式と同期を行うオブジェクトの差異のみを転送する方式とがある。両方式に共通な最適化としては、Object Serialization 時に Serialize するオブジェクトのメソッドと同一のメソッドがマージする相手オブジェクトにある場合には、それを Serialize しない。

また、差分データオブジェクト転送方式では、相手 SyncStore にある VersionVector を知ることにより、そのデータ変更の差分であるログだけを転送する。さらに、この差分データ転送では、VersionVector に基づいたログ情報を相手 SyncStore に転送するが、そのログ内に同一データに対する再加工があった場合、それを省略することができる。

(3) データストリーム転送レイヤのデータ量削減

実際にデータ転送を行う Synchronizer レベルでデータストリームのビット列の圧縮を行う。データストリームの授受を行う Synchronizer が compress()/decompress()し、使用するネットワークの最適パケットサイズに分割して転送する。

(4) データ転送とデータ処理の並行動作
全体の処理時間を短くする意味で、Synchronizer によるデータ転送において、相手側 SyncStore へのデータストリーム転送をオブジェクト単位とし、相手側 SyncStore でのマージ処理との並行処理を実現する。このオブジェクトデータストリーム単位の転送は、LAN 環境に比較して不安定な無線ネットワーク下で、転送途中でネットワーク切断が発生した場合にも、その再転送が少なくすむという利点がある。

これらの転送量削減方法を、その利用環境に合わせて組み合わせることにより、効率的なデータ転送とすることが期待できる。 さらに、上記(1)(2)(3)の各レイヤの最適コンフィギュレーション値を標準的に得る API (Adaptation API) を定義することで、無線/LAN 環境ごとにユーザがわずらわされることなく最適なデータ転送を実現することが可能である。

6 まとめ

本オブジェクト同期化モデルで取り上げる対象データオブジェクトはまず1オブジェクトとしており、一連のオブジェクト（オブジェクトグループ）を対象としていない。まず、このモデルでのデータ転送の最適化を図る。このモデルでは3レイヤの最適化が可能であり、これをパラメータ化することにより、最適な「いつでも、どこでも」利用が実現できる。 この中でデータオブジェクトの同期では VersionVector による差分転送方式を採用しているが、現在は Serializable オブジェクトのメソッドのバージョンは考慮していない。このメソッドのバージョンを付加した VersionVector 方式の検討も行う同時に、方式評価を行っていく予定である。

また、現在のモデルは一連のオブジェクトつまりトランザクションやバッチ処理は扱えない。これを扱えるようにすると同時にそのデータ転送最適化も検討していく予定である。

7 おわりに

本論文作成にあたって、貴重な時間を割いてレビューしていただいた三菱電機情報技術総合研究所 撫中主事、清原主事、桜井研究員、小野研究員に感謝します。

参考文献

[1] Mahadev Satyanarayanan, et al, Coda : A highly Available File System for a Distributed Workstation Environment, IEEE Transaction on Computers, 39(4), April 1990

[2] Richard G.Guy, et al, Implementation of the Ficus Replicated File System, USENIX Conference Proceedings, USENIX,June 1990

[3] Luosheng Peng, et al, Using Version Vectors in the Star-structured Data Consistency Model, Principles of Distributed Computing (PODC), June 1998 (paper submitted)

[4] Mobile network computer reference specification, http://www.internet.ibm.com/computers/networkstation/os/mncrs.html

[5] James Gosling, et al, The Java Language Specification, Adison-Wesley, Menlo Park, Califomia, August 1996