Coda ファイルシステムにおける 隔離されたクライアントでのキャッシュミス補償機構

稲村 浩
NTT 情報通信研究所
〒 239-0847 神奈川県 横須賀市 光の丘 1-1
Internet: inamura@isl.ntt.co.jp

概要

モバイルコンピューティングにおいて，一時的なネットワークの切断によるサーバからの隔離に対処するため，クライアントキャッシングは広く用いられている．Coda ファイルシステムのクライアントはディスコネクトオペレーションと呼ばれる機構を用いてサーバにコンタクトせずに動作の継続が可能である．しかし，クライアント同士のみでファイルをやりとりする事はできない．本論文では Import/Export と Session server の 2 つの機構の設計と実装について述べる．これらの機構はクリーンなオブジェクトに対する読みだし共有機能を，受け入れ可能な応答時間のもとにサーバから切り離されたクライアントに付加している．

1. はじめに

モバイルコンピューティングにおける計算機の利用状況においては，サーバからの一時的な隔離は自然に起こり得る．モバイルコンピューティング向けの分散ファイルシステムにおいてクライアントキャッシングは広く用いられている [1] [5]．ネットワークからの隔離を透過にするためには，キャッシュの質が重要な要素になる．もし必要なファイルが十分にキャッシュされていなければ，利用者はキャッシュミスによって作業の継続が妨げられる．

サーバから隔離された複数のクライアントを考える．あるクライアントで起こったキャッシュミスは，他のクライアントが保持するキャッシュされたオブジェクトで満たすことができる．サーバに接続すること無しにクライアント間でキャッシュミスを補償することで，読みだし共有の限界を実質的に広げることができる．

このキャッシュミス問題を検討するために Coda ファイルシステムは良い基盤となり得る．Coda はクライアントサーバモデルに基づいて構成されている．この構成は可伸性，管理の容易さ，進んだセキュリティの提供という利点がある [6]．性能と可用性の向上のために Coda はクライアントキャッシングとサーバ複製を採用している．Coda では，サーバから隔離されたクライアントが，そのローカルディスクにキャッシュされたファイルを使うことで動作を継続する，ディスコネクトオペレーションを提供している．Coda クライアントはネットワークに再接続すると，ディスコネクトオペレーションの期間中に行われた変更点をサーバに送る．これをリインテグレーションと呼ぶ．クライアントとサーバの間で，行われた変更点に食い違いがあった場合，Coda は利用者の助けを得て解決する．

この論文の以降の構成は以下の通り．第 2 章では隔離されたクライアントにおけるキャッシュミスの問題を提示する．第 3 章ではクライアントにおけるキャッシュミスの補償機構の設計と実装について述べる．第 4 章では評価について述べ，第 5 章では関連研究について述べた後，第 6 章でまとめる．

2. 隔離されたクライアントにおけるキャッシュミス

サーバからクライアントが隔離された結果のうちで，キャッシュミスは利用者に対して最も影響がある．現在の Coda の設計では，一台のクライアントにあるデータを他のクライアントに伝播させることはできない．したがって，キャッシュミスしたオブジェクトはサーバに接続しない限り得られない．

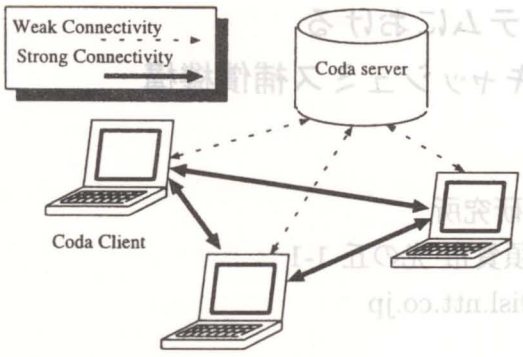

図 1: サーバから隔離されたクライアント

クライアントサーバモデルにおけるセキュリティの改善のために，Coda ではサーバは信頼できるものと仮定し，多くの設置事例では物理的に守られている．したがって，複数のクライアントがサーバから地理的に隔離されることが起こり得る．ディスコネクトオペレーションは，たとえ近くのクライアントが多くのファイルをキャッシュしていたとしても，隔離された複数のクライアントの間でキャッシュミスを補償できない．

この問題を解決するために，2 台以上のクライアントがサーバに接続することなく，直接にキャッシュミスを補償する新しい機構を導入した．すなわち，他のクライアントが保持するキャッシュを協調的に利用してキャッシュの補償を行う．Coda の設計方針は利用者から隔離の影響をでき得る限り隠すことにある．この方針に則り，本機構は利用者の観点からは透過であるべきである．さらに本機構は，モバイルコンピューティングで利用可能なさまざまなデバイスを活用すべきである．例えば，ラップトップコンピュータは他のコンピュータにファイルを渡すためのさまざまなデバイスを備えている．そのデバイスにはフロッピーディスク，フラッシュメモリカードなどのリムーバブルメディアや，モデムや無線ネットワークによってもたらされるネットワーク接続等がある．

キャッシュミスの問題と，本機構の有効性を示す例を挙げる．大域的なネットワークの接続性が不可能あるいは高価である，例えば外国のような場所で複数の利用者が作業する場合を考えよう．ある利用者が，emacs を使うためのライブラリファイルを自分のラップトップにキャッシュしていなかったことに気がついたとする．一方，隣の利用者は emacs のライブラリファイルを沢山持っているが，現状ではファイルを透過に渡す方法はない．サーバは母国にあるため，接続するためのコストは高いかも知れない．図 1 はこの状況を表している．Coda サーバへの接続性は弱いが，Coda クライアント同士の接続性は強い．本機構はこの局所的な強い接続性を活用する．

本研究のゴールは，これらの能力を持つ新たな機構を，信頼性と頑健さに関する Coda の基本的な設計方針を大きく変えることなく提供することである．

3. 設計と実装

問題解決のためには，Coda を拡張しなければならない．変更無しの Coda と通常の UNIX コマンドや NFS などの他の手段を組み合わせる手法では十分解決できない．なぜなら，

- ftp, cp や tar を用いて，キャッシュミスしたファイルを元の位置に書き込もうとすると，アクセス制御によって必ずしもうまくいかない．例えば /coda/misc/lib/emacs をキャッシュミスしたが，その利用者がこのディレクトリに書き込み権限を持っていなかった場合が考えられる．つまり，通常の利用者コマンドでは不十分なアクセス権による制限を回避できない．
- 他のクライアントマシンのディレクトリを NFS によってマウントして利用する方法を考える．この手法では単一名前空間の利点が得られない．さらに，Coda と連係しない限り，ファイルキャッシングの利点も得られない．

可搬計算機で利用可能なさまざまなデバイスをキャッシュミスの補償に利用するために，それらの特性の違いを考慮しなければならない．キャッシュミスの補償には，クライアントキャッシュのデータや状態のサブセットを何らかの伝送路経由で別のクライアントに移動させることが必要である．可搬計算機で利用可能なさまざまなデバイスに関して，伝送路として見た時のトポロジーと接続期間について表 1 にまとめた．

これらのデバイスをキャッシュミスの補償に利用するために重要なのは，接続期間である．リムーバブルデバイスは接続期間を予測するのが難しいため，それを用いた複雑なやりとりは仮定できない．したがって，接続期間に関して 2 つの別個の仮定を置き，それぞれに対してメカニズムを設計した．

トポロジ \ 接続期間	安定あるいは予測可能　短い または 予測不能
ポイントトゥポイント	シリアルリンク　リムーバブルメディア
マルチプルアクセス	無線/有線 LAN

表 1: 可搬計算機で利用可能なデバイストのポロジと接続期間

Import/Export はバッチ形式のコマンドである．接続期間が予測できない，例えばリムーバブルなどのデバイスのために設計した．

Session server は接続期間が予測可能あるいは安定したデバイスに対して設計した，サーバクライアントモデルに基づく機構である．“セッション”とはクライアント同士がが安定な接続を維持し，ファイル共有を行うことのできる有限の期間である．

本研究ではクリーンなオブジェクトを読みだし共有する機能の実現を扱った．このように限定する事で設計が非常に簡単になった．キャッシュミスハンドラでは本質的に，他のクライアントで起こった更新をタイムリに検知することができない．そこで本研究では書き込み共有を検討範囲外とした．Coda ではクリーンなオブジェクトとは，リインテグレートすべき変更点を持たないオブジェクトと定義される．読みだし専用に設定された多くのバイナリファイルや，書き込み可能であっても隔離中に変更されなかったファイルはクリーンなオブジェクトである．本機構では隔離されたクライアント間での信頼を仮定している．受け入れ側のクライアントは，他のクライアントから転送されたキャッシュの中身を信頼しなければならない．万一この信頼が誤りであった場合でも，転送をクリーンなオブジェクトに限定することによって，疑わしいファイルがサーバに書き戻されて，そのサーバに接続する全てのクライアントに伝播する危険を避けることができる．

3.1. Import/Export

クライアントキャッシュの状態とデータは Coda の内部ではいくつかのデータベースで管理されている．図 2 はクライアントキャッシュマネージャの主要部である *Venus* の内部構造を示している．Venus はマルチスレッドの利用者レベルプロセスとして実装されている．

キャッシュマネージャの下部構造はカーネルの中に組み込まれており，vnode やキャッシュのイベントをアップコールする機構として働く．Venus は，vnode/VFS インターフェイスを用いて行われた操作によって引き起こされた，カーネルからのアップコールを処理する．Venus 内部のワーカスレッドはカーネルモジュールからのアップコールメッセージを待ち，その vnode をアクセスしたプロセスに対して，ファイルシステムに関するシステムコール処理を行う．

Venus はいくつかのデータベースで構成されている．それらはそれぞれワーカスレッドを持ち，通常の UNIX 環境におけるデーモンプロセスのように振舞う．キャッシュの内部状態は以下のデータベースの要素として記述される．

FSDB (File System DataBase) ローカルなキャッシュファイルと Coda のファイルシステムのオブジェクトとの対応を記述するデータを保持している．このデータは，時刻，大きさ，アクセス制御リストなどの属性情報も含んでいる．

VDB (Volume DataBase) ボリューム (volume) の状態を記述している．ボリュームは UNIX におけるディスクパーティションに相当するファイルの集合の単位である．Coda において，ボリュームは複製や配置の管理の単位である．ボリュームはそれ自体の属性を持つ．

VSGDB (Volume Sever Group DataBase) ボリュームとサーバの対応を記述している．VSGDB はまた，サーバ複製のためにそれぞれのサーバとの接続状態も管理している．

Coda においてはファイルシステムオブジェクトのキャッシュミスには 2 つの種類がある．ひとつのファイルシステムオブジェクトはファイルステータスとファイルデータの 2 つの部分から構成される．ファイルステータスは UFS における inode 情報に該当し，ファイルデータは実際のファイルの中身である．したがって，2 つの場合が考えられる．すなわち，ファイルステータスとファ

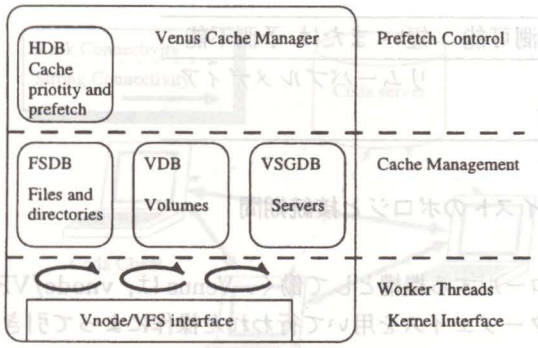

図 2: Venus の内部構造

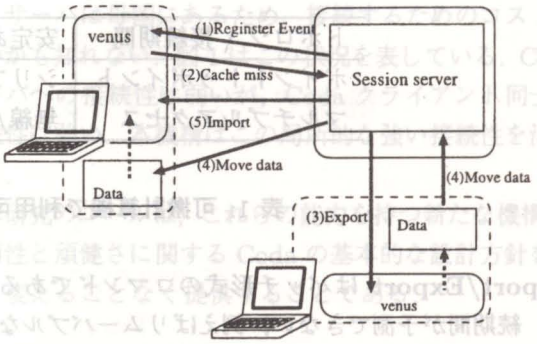

図 3: Session server を用いたキャッシュミス補償

イルデータの両方をキャッシュミスした場合と，ファイルデータだけをキャッシュミスした場合である．

これらの違いは，他のクライアントからのキャッシュを取り入れることに影響する．データミスのみの場合，一貫性の維持のために本機構は取り込むオブジェクトと，ローカルに存在するファイルステータスの間でバージョン情報を検査する．データとステータスの両方のミスの場合には，利用可能性を高めるために，本機構はある程度の一貫性を犠牲にして，検査なしでオブジェクトを取り入れる．

以上の検討の結果，2 つのコマンドを新設した．

Export : クライアントのキャッシュ空間のサブセットを書き出す．

Import : `export` されたキャッシュデータからキャッシュの状態を再構成する．

リクエストのインターフェイスを単純で小さなものにするために，`export` は自己完結するようにキャッシュの状態をダンプする．個別の状況に合わせて要求条件を並べるのは簡単ではない．例えば，利用者はどの VDB や VSGDB の内容がデータに含まれなければならないかを指定しなければならない．指定の単純さのために データサイズが大きくなるデメリットを受け入れて，`export` はキャッシュの状態の回復に必要なものを全て書き出しデータに含めている．

利用者管理コマンドの 'cfs' を拡張してこれらのコマンドを実装した．関係する FSDB, VDB や VSGDB データベースには'export()' と 'import()' メソッドが追加された．'cfs' コマンドは Venus に指示することで，関係するオブジェクトを再帰的に書き出し，取り入れを行う．

3.2. Session server

Import/Export コマンドでのアプローチは利用者の注意と，操作が繰り返し可能であることに依っている．サーバからの隔離の結果，コマンドが ETIMEOUT エラーで終了したとき，利用者はキャッシュミスが起こったことに気がつく．例えば 'abc' という名前のファイルがキャッシュミスしていたとき，`ls abc` というコマンドは `abc: Operation timed out` というメッセージを返す．それから利用者は import/export コマンドを入力してキャッシュミスの補償操作を行う．キャッシュが補充されると，利用者は `ls abc` を繰り返す．`ls` コマンドは再度行っても問題はないが，全てのアプリケーションではうまくいくとは限らない．

隔離されたクライアント同士は協調することで，利用者の介入なしに動作するのが望ましい．操作の繰り返し可能性に依らないためには，Venus はそのオブジェクトにアクセスしたプロセスにキャッシュミスエラーを返してはならない．これらの目標のために，以下のように 2 つの点で Coda に変更を加えた．

1. アクセスしているプロセスにキャッシュミスを見せないように Venus を修正した．Venus の内部では，ファイルにアクセスすることでキャッシュの探索が起こる．もし，ファイルシステムオブジェクトが見付からず，ネットワークが切断されている場合，そのアクセスはキャッシュミスと見なされる．この修正によって，Venus はファイルアクセスを処理する

ラップトップクライアント	IBM PC/AT 互換機, Pentium 133MHz, 40MB RAM, 1GB IDE HDD. 10Mbps PCMCIA 3COM Ethernet card. FreeBSD 2.2.1R
デスクトップサーバ	IBM PC/AT 互換機 Pentium 133MHz, 64MB RAM, 1.4GB IDE HDD. 10Mbps ISA Ethernet card. FreeBSD 2.2.1R

表 2: 評価環境

File name	File size (KB)	Session server (ms)	Well-connected cache-miss (ms)	Well-connected cache-hit (ms)	UFS cache-hit (ms)
coda_cie.tex	16	2500	54	3.3	0.89
coda_cie.ps	180	3100	310	3.6	0.88
clog	590	3200	940	3.5	0.99
kernel	1393	6600	2100	3.4	1.0

表 3: fopen のターンアラウンド時間 vs. ファイルサイズ: 測定した値は 10 回の平均である．ファイル "coda_cie.*" は本論文の原稿ファイルである．"clog" は Coda の認証コマンドである．"kernel" は Coda のモジュールを含む FreeBSD カーネルである．

ワーカスレッドを sleep させ，import メソッドの終了を待たせる．キャッシュの探索の再起動によって，アクセスしていたプロセスはレジュームされる．

2. Session server を追加した．クライアント間の協調のために，Session server はキャッシュミス補償を集中化し，自動化する．Coda に既にある advice サービス [4] は キャッシュミスを含む Venus 内のイベント監視機能を提供している．この機能はキャッシュミス補償を自明に実現できる程には自由度が高いわけではないが，実装の良い基礎になる．このイベント監視機構を用いて Session server を実装した．

図 3 は Session server の働きを示している．そのセッションで単一のクライアントが Session server を動かしている．Session server を用いて，キャッシュミスは以下のステップで補償される．

1. 起動されると，Session server は Venus にキャッシュミスイベント監視を登録する．その登録により，Venus はキャッシュミスイベント発生時に Session server にコンタクトする．
2. キャッシュミスが起こると，Venus は Session server にコンタクトする．サーバはどのクライアントに export を要求すべきかを判別する．
3. その Venus にミスしたオブジェクトを Export するように要求する．
4. 書き出されたデータを，要求元のクライアントに移動する．
5. サーバは要求元のクライアントに Import 要求を行い，クライアントにキャッシュデータを取り込ませる．

4. 評価

サーバから隔離されたことを透過にするという観点から，キャッシュミス補償による明らかな影響は，ミスしたファイルシステムオブジェクトに最初にアクセスした時の応答時間の増大である．キャッシュミス補償時の応答時間を観察するには，open システムコールのターンアラウンド時間が大きな要素となる．Import/Export 機構と実装の多くを共有するため，Session server 機構の評価は両機構に当てはまることから，キャッシュミス補償の評価として Session server の評価で代表させた．キャッシュミス補償にかかる時間とファイルサイズの関係を見るために，いくつかの大きさのファイルに対して，stdio ライブラリを用いてそのファイルを開き，内容を全て読みだし，閉じる，簡単な試験プログラムを用いた．

試験手順は以下の通り．本機構を組み入れた 2 台の Coda クライアントを用意し，その片方で Session server を動作させた．複数のファイルをキャッシュした上で任意の一台のクライアントをサーバから隔離し，キャッシュをフラッシュした後，そのクライアントで試験プログラムを動作させた．実験に用いた機材は表 2 にまとめた．測定には Pentium サイクルカウンタを用いた．実験結果は表 3 に示した．

結果より，Session server を用いた場合の，最小の処理時間は接続性が良好な場合に比べて大きい．この違いは，Session server と Venus が何度も通信を行ったことからメッセージ路が長くなったことに起因する．それでも全体の応答時間は多くの会話的なアプリケーションの元では受け入れられる程度である．

5. 関連研究

本研究と前提や目的を共有するいくつかの研究がある．

- *Xerox* の *Bayou* [7] では可搬計算機で利用可能なさまざまなデバイスを扱おうとしている．本研究とは前提のうちで，アプリケーションが移動性を扱うかどうかという点で異なる．本研究では Coda における移動透過性を重視するアプローチに重きを置いた．Bayou では anti-entropy protocol における再統合の振舞いをアプリケーションに開放している．

- *MIT* の *Rover Toolkit* [3] も，可搬計算機で利用可能な，電子メイルを含むさまざまなデバイスを扱おうとしている．その Queued RPC は低速なネットワークではとくに有効である．また，RDO というオブジェクトベースのアブストラクションを提供している．Rover のアプローチは移動性を意識した アプリケーションを支援している．最近，Rover toolkit を用いたファイルシステム [2] が設計されつつあるが，現状では読みだしのみを提供している．

6. まとめ

Coda ファイルシステムで隔離されたクライアントでは，キャッシュミスが大きな問題である．キャッシュミス補償をクライアント側で行い，可搬計算機で利用可能なさまざまなデバイスを活用するために，Import/Export と Session server の 2 つの機構を提案した．これらの機構はサーバから隔離されたディスコネクトオペレーションで動作するクライアントに対し，クリーンなオブジェクトに対する読みだし共有の機能を，受け入れ可能な応答時間のもとに提供した．

参考文献

[1] L. Huston and P. Honeyman. Partially Connected Operation. In *USENIX Symposium of Mobile and Location-Independent Computing*, 1995.

[2] A. D. Joseph, G. M. Candea, and F. Kaashoek. RFS: A Mobile-Transparent File System for the Rover Toolkit. In *Presented as a Works-In-Progress poster at the Sixteenth Symposium on Operating Systems Principles*, 1997.

[3] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford, and F. Kaashoek. Rover: A Toolkit for Mobile Information Access. In *15th ACM Symposium on Operating Systems Principles*, 1995.

[4] L. Mummert, M. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *15th ACM Symposium on Operating Systems Principles*, 1995.

[5] P. Reiher, J. S. Heidemann, D. Ratner, G. Skinner, and G. J. Popek. Resolving file conflicts in the Ficus file system. In *USENIX Conference Proceedings*, pp. 183–195. USENIX, June 1994.

[6] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steer. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transaction on Computers*, 39(4), April 1990.

[7] D. B. Terry, K. P. Marvin M. Theimer, and A. J. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *16th ACM Symposium on Operating Systems Principles*, 1997.