システムソフトウェアと
オペレーティング・システム 71－10
マルチメディア通信と分散処理 73－10
（１９９５．１１．３０）

# Codaプロジェクトの概要

稲村 浩[1]
NTT 情報通信研究所

カーネギーメロン大学計算機科学科のM. Satya教授のグループで研究されている分散ファイルシステム Coda は Disconnected Operation によりモーバイルコンピューティングを支援していることで知られている。本稿では Coda についてその発展の歴史に沿って説明を行う。Coda の目標は可用性と可伸性に優れ、セキュリティのあるシステムを提供することであった。その後、低速回線のサポート機能などが追加され、現在では Coda は 30 人程度の利用者コミュニティに対して、高可用性とラップトップコンピュータでの利用という 2 つの点で快適な環境を提供している。

# An Introduction to Coda

Hiroshi Inamura[1]
NTT Information and Communication Systems Laboratories

Coda is a distributed file system invented at the research group led by Prof. M. Satya in School of Computer Science, Carnegie Mellon University. It is known as a file system supporting mobile computing with its novel technique "disconnected operation". This paper describes Coda from its origin to current status. The purposes Coda pursuits are highly available, scalabe and secure file service. After several years of research including exploiting low speed network connectivities, Coda now provides highly available distributed file service and convenient use of laptop computers for a user community around 30 people.

## 1 はじめに

カーネギーメロン大学計算機科学科の M. Satya 教授のグループで研究されている分散ファイルシステム Coda は Disconnected Operation によりモーバイルコンピューティングを支援していることで知られている。本稿では Coda についてその発展の歴史に沿って説明を行う。

まず2章で Coda の前身である AFS (Andrew File System) について述べ、それから 3章で Coda の問題意識についてまとめた後、4,5章で実現されたメカニズムを紹介し、6章で最近行われた仕事を説明して Coda の後継になるであろう Odyssey に触れ、7,8章でまとめる。

なお筆者は 1994 年 9 月から 1995 年 9 月まで Coda プロジェクトにて研究を行った。

[1] 〒 238-03 神奈川県 横須賀市 武 1-2356
Internet: inamura@isl.ntt.jp

## 2 AFS から Coda へ

### 2.1 AFS の概要

Coda は AFS の後継であるので、AFS の特徴を引き継いでいる。そこで AFS の特徴的な設計ポリシーを以下に述べる。

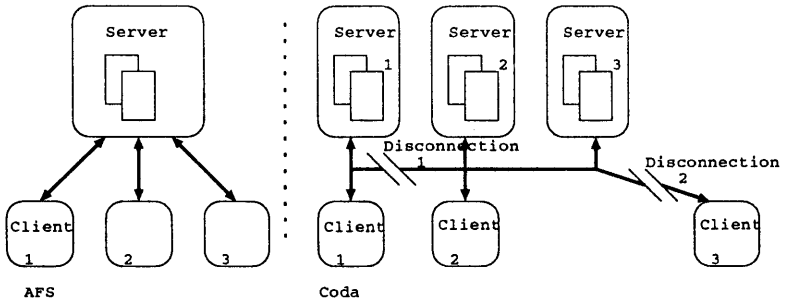


図 1: AFS と Coda

- 単一大域名称空間
  /afs の下に全ての共有ファイル資源の名称空間が割り当てられ、その名称空間は全てのAFS のクライアントから同一に見える。

- セキュリティ
  サーバとクライアントの役割が異なるため、サーバは他のサーバを信用するのに対し、クライアントがサービスを受けるには認証が必要である。

- 可伸性 (Scalability)
  数千台のクライアントをサポートする事を念頭に設計されている。

広く使われている分散ファイルシステムである NFS に対して、AFS では以下のような異なるメカニズムを用いている。

- コールバックの利用
  クライアントがキャッシュしているファイルが変更されると、サーバはクライアントに通知してキャッシュファイルを無効にする。

- ローカルディスクを用いたキャッシング
  比較的大きな単位 (AFS-2 まではファイル全体、AFS-3 では 64KB、Coda ではファイル全体) でファイルをクライアントのローカルディスクにキャッシュとして保持する。

- サーバクライアントモデル
  サーバとクライアントは役割が固定しており、NFS のように同一のホストが同時にサーバとクライアントとして働くことはない。

### 2.2 AFS における可用性の問題

集中的なサーバには可用性 (availability) の問題が避けられない。利用者が AFS を唯一の分散ファイルシステムとして利用し始めると、その可用性に関する問題が明らかになった。

ファイルサーバはその性質上多数の利用者にファイルサービスを提供している。メンテナンスでサーバを停止しなければならない場合、そのサーバにホームディレクトリを置いている利用者は作業を続けられない。AFS でも読みだし専用領域には複製を利用できるが、書き込みが発生するホームディレクトリには有効ではなかった。

## 3 Coda の設計

Coda の最大の特徴は複製を利用できる事である。複製を利用する事により可用性の問題が解決でき、同時にモーバイルコンピューティングに特徴的な動作パターンをサポートすることが可能になった。

図 1 はこの複製の様子を示して AFS と Coda の違いを対比させている。Coda では 2 種類の複製を行う。

1. サーバー複製
   高い可用性のため。図 1中の Disconnection 1 のタイプの障害を覆い隠すことが可能である。

2. クライアントキャッシング

高い可用性と Disconnected Operation の提供。図1中の Disconnection 2 のタイプの障害を覆い隠すことが可能である。

## 4 Coda におけるサーバ複製

サーバの単一故障の問題を避けるため Coda ではサーバを複製している [SKK+90]。ファイルの集合をボリュームと呼び、複製はこのボリュームを単位としている。ボリュームは複数のサーバに複製される。一つのボリュームに対して、それを複製して保持するサーバの集合を VSG (Volume Server Group) と呼ぶ。クライアントはボリュームごとに利用可能なサーバを定期的に検知しており、現在利用可能なサーバの集合を VSG に対して AVSG (Available VSG) と呼んでいる。$\|AVSG\| = \phi$ はサーバクライアント間の回線が切断されたり、サーバが全てダウンしたなどを意味する。

サーバ同士がネットワーク分断などで互いに見えない状態で同一のファイルが変更された場合には内容の不一致が起こり得る。Coda では楽観的複製制御 (Optimisctic replication) に基づいて、ファイルの内容不一致はユーザにその解決を委ねているが、ファイルの種類によっては自動的な解決も可能である。その枠組として ASR (Application Specific Resolution) [KS93] が設けられている。ASR ではファイルの型ごとに更新の衝突を解決するプログラムを利用者が登録しておき、衝突が検知された時にはそのプログラムが自動的に起動される。

## 5 Disconnected Operation

Disconnected Operation[Kis93] の実現により、可用性の向上とモーバイルコンピューティングへの利用が計れるようになった。Disconnected Operation とはネットワークの回線切断後にクライアントのローカルディスクに残っているキャッシュされたファイルで動作を続けるメカニズムである。ファイル単位でキャッシュを保持しているため Coda ではキャッシュの完全性が高い。

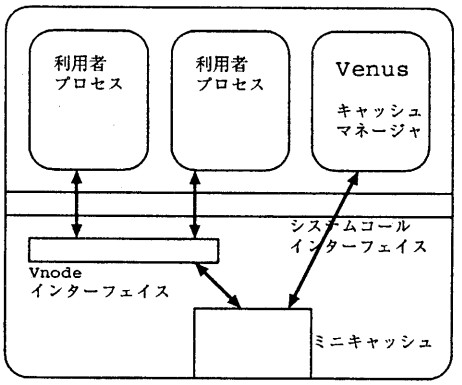


図 2: クライアントホストの構成

### 5.1 Coda クライアントの構成

Coda のクライアントは図 2 に示す構成で動作する。Venus というキャッシュマネージャが利用者プロセスとして動作しており、これが他の利用者プロセスからのシステムコール要求を受けてファイルサービスを提供する。

図中のミニキャッシュは一部の Vnode 操作を提供しており、カーネル空間から Venus へのアドレス空間を越える手続き呼出しの頻度を低下させるのに非常に役立っている。ミニキャッシュを停止させてベンチマークによって性能を測定すると、67% もの低下が見られた [SKS90]。

### 5.2 キャッシュマネージャの内部状態

図 3 のように接続性の変化に応じてボリューム毎に定義された状態がある。

- Hoarding
  少なくとも一つのサーバとの接続性がある状態。利用者にファイルサービスを提供するのと同時にキャッシュを重要度 (0-1000) と参照パターンとの間で平衡状態を保つ。

- Emulating
  Venus はサーバになり変わって利用者にファイルサービスを行う。回線切断状態で行われた変更は全てログに記録する。

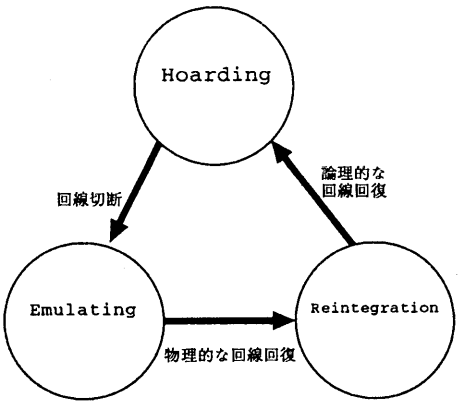


図 3: ボリュームの状態遷移

- Reintegration
  サーバとしての動作状態とキャッシュマネージャとしての動作状態の間の過渡的な状態である。サーバとの接続性を確立した後、まず回線切断中に起こった変更操作をサーバに対して再実行する。衝突が検知され、ファイルに対する更新の衝突のような自動的に解決できない場合には後で利用者に解決を求められるように状態を整える。

キャッシュマネージャ内部の スレッドがキャッシュしているボリューム毎に数分に一度づつ接続性を検知しており、一つもサーバが見えなくなったり、また見えるようになったボリュームは状態遷移が起こされる。

Emulating 状態で採取されたログを最適化することは2つの点で重要である。1) 大きなログはディスクを圧迫する。2) 大きなログは reintegration に時間がかかる。そこで互いに打ち消し合うような操作列はログから消去するなどの最適化を行い、サイズの縮小に努めている。その結果、最適化を行ったことでディスクスペースと reintegration の遅延の両方で利得が得られた [M. 93]。

### 5.3 Disconnected Operation の実際

回線切断には、自発的なものと偶発的なものの2種類がある。高可用性の立場からは障害透過に扱う事が要求される。したがって Coda では認証を除いて回線切断と再接続に当たって必要なコマンドシーケンスは特にないが、キャッシュの質をあげるためにいくつかのメカニズムを備えている。hoard コマンドは、特定のファイルがキャッシュに保持されていることを保証する。どのファイルをどの程度の重要度で保持したいかは設定ファイルで指定する。従って回線切断に先だって持ち帰りたいファイルを指定して hoard でキャッシュの状態を再設定する。ローカルディスクが 200MB 程度の場合、キャッシュの再走査には DECpc 425SL では最大で 15 分程度である。

自発的な場合の典型的な Disconnected Operation セッションは以下の流れになる。

1. `hoard -f` *hoardfiles*
2. `hoard walk` の終了を確認する。
3. 回線切断
4. ネットワークから外して利用する。
5. 回線再接続
6. `clog` にて再認証する

この後 Reintegration が自動的に起動される。Reintegration では可能な限り自動的に一貫性を回復するが、不幸にして衝突が解決できない場合にはそのオブジェクトは特別なディレクトリに変化して、`repair` コマンドによる回復操作を利用者に求める。例えば `test.c` がサーバとクライアントの間で衝突した場合には、`test.c` は特殊なディレクトリに変化して、その下には `local` と `global` という名前でそれぞれのバージョンが参照可能である。利用者はその中身を比較して必要なら変更した上で `commit` することにより、一貫性を回復できる。

## 6 現在のプロジェクトと将来の方向

### 6.1 低速ネットワークの利用

ラップトップ PC を長期に渡って持ち出して利用する場合、更新の衝突の頻度を下げるために定期的

に再接続するのが望ましい。電話などの低速回線を利用してreintegrationを起動したり、低速回線を生かして接続状態を保って作業が可能になれば便利である。

低速な接続性の利用のためにいくつかの変更が行われた[MES95]。

- 転送プロトコルの改良
  RPC プロトコルを 9600[bps] の SLIP のような低速ネットワークでも使えるように改良した。
- キャッシュの有効判定の効率向上
  キャッシュの有効判定の粒度を上げ、ファイル単位ではなくボリューム単位で判定できるようにした。これは「キャッシュされたファイルが変更されるのは比較的稀である」という観測結果に基づいている。
- Trickle Reintegration
  ファイルへの変更を低速ネットワークですぐに送信するのではなく、あるagingの期間遅延させることにより、ログの最適化を利用してデータ転送量を減らす。
- キャッシュ制御に対する利用者によるヒント
  ファイル毎のキャッシュの重要度を会話的に制御できるようにした。低速ネットワーク利用時の `hoard walk` の動作を利用者が制御できるようにした。

これらの改良によって、ネットワーク速度が2桁変化しても利用者に快適な環境を提供できるようになった。

### 6.2 IOT(Isolated Only Transaction)

Disconnected Operationの導入に伴って、以下のようなバージョンの不一致の問題が起こり得る。

例えばあるプログラム prog をディスコネクトした ラップトップPCで make することを考える。その PC がサーバから離れている間に、必要なライブラリ libutil.a が変更されたとする。ラップトップPC では libutil.a の古いバージョンをキャッシュしているので、問題なく実行形式 prog は生成でき、再接続も成功するので、そのユーザは libutil.a が変更された事を知る機会がない。

このような問題を read/write コンフリクトと呼ぶ。IOT[LS95] は write する部分をトランザクションとして扱い、再接続の時にコンフリクトの発生の可能性を検知して、必要な操作の (この例では make コマンドである) 自動的な再実行を行うメカニズムである。

### 6.3 移動性と適応可能性

モーバイルコンピューティングへの適用に当たって、移動に伴って起こる動作環境の変化をアプリケーションや利用者には見せないのが Coda のアプローチであった。しかし、「今いるところから一番近い中華レストランはどこか？」といった問い合わせに答えるデータベースのように、移動したことを積極的に利用するようなクラスのアプリケーションも存在する。また、ラップトップマシンの電池が少なくなった場合に機能を縮小して動作を続けたり、回線速度の変化に合わせて QoS を変化させるビデオプレイヤのようなアプリケーションもある。このように移動性に対して適応的なアプリケーションをサポートするためには、Coda とは違うアプローチが必要になる。Odyssey[Bri95] プロジェクトでは適応可能性をキーワードにして、計算機で利用可能な資源の変動をアプリケーションに伝えるメカニズムを統一的に扱えるリソースマネージャの構築を進めている。

## 7 現状

Coda は CMU で 30 人程度の利用者に提供されている。現在 Mach 2.6 で動作しており、利用者の殆んどはデスクトップワークステーション (DECstation 5000/200) とラップトップ PC (DECpc 486SL) を持ち、両方で Coda を動かしている。利用者のホームディレクトリを保持するボリュームは 3 台のサーバに複製されている。Coda の名称空間は利用者からは `/coda` の下に見え、例えばホームディレクトリは `/coda/usr/inamura` となる。

Coda のソースは Coda の上にあり、開発者は開発に Coda を用いているが、デバッグは少し複雑になる。

## 8 まとめ

Coda の目標は可用性と可伸性に優れ、セキュリティのあるシステムを提供することであった。その後、低速回線のサポート機能などが追加され、現在では Coda は 30 人程度の利用者コミュニティに対して、高可用性とラップトップコンピュータでの利用という 2 つの点で快適な環境を提供している。

筆者は CMU で実際に Coda を利用する機会を得た。したがって筆者のホストには /afs と /coda が両方存在した。ある日 Coda 側で仕事をしていると、AFS のホームディレクトリがアクセスできなくなっていることに気が付いた。原因はネットワークの障害であったのだが、Coda 側の空間が自動的に Disconnected Operation に移行していたのに気が付かなかったというのが実際である。

## 参考文献


[Bri95] Brian Noble and M. Satyanarayanan and Morgan Price. A Programming Interface for Application-Aware Adaptation in Mobile Computing. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing*, 4 1995.

[Kis93] James Jay Kislter. *Disconnected Operation in a Distributed File System.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1993.

[KS93] Puneet Kumar and M. Satyanarayanan. Supporting Application-Specific Resolution in an Optimistically Replicated File System. In *Proceedings of the 4th IEEE Workshop on Workstation Operating Systems*, 10 1993.

[LS95] Qi Lu and M. Satyanarayanan. Improving Data Consistency in Mobile Computing. In *IEEE Fifth Workshop on Hot Topics in Operating Systems*, 1995.

[M. 93] M. Satyanarayanan and James Kistler and Lily Mummert and Maria R. Ebling and Puneet Kumar and Qi Lu. Experience with Disconnected Operation in a Mobile Computing Environment. In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, 8 1993.

[MES95] Lily Mummert, Maria Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *15th ACM Symposium on Operating Systems Principles*, 1995. To appear.

[SKK+90] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steer. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transaction on Computers*, 39(4), April 1990.

[SKS90] David Steer, James Kistler, and M. Satyanarayanan. Efficent User-Level File Cache Management on the Sun Vnode Interface. In *Proceedings of the Summer Usenix Conference*, 6 1990.