修士学位論文

題目

バージョン管理機能をもつファイルシステムの設計と実現

指導教官
井上 克郎 教授

報告者
山本 哲男

平成 11 年 2 月 16 日

大阪大学 大学院基礎工学研究科
情報数理系専攻 ソフトウェア科学分野

平成10年度 修士学位論文

バージョン管理機能をもつファイルシステムの設計と実現

山本 哲男

内容梗概

ソフトウェア開発過程で作成されるプロダクトの識別や制御，状態の把握等を行うことを目的として，ソフトウェア構成管理に関する研究が広く行われている．特に，コーディング段階以降では，開発チームが作成したプロダクトに対する様々な修正を正しく識別し，組織化し，管理することが中心となる．通常，プロダクトの管理はバージョン単位で行なわれる．バージョンに基づいてプロダクトの作成/変更作業の逐次記録や追跡をうまく行なうために，様々な管理手法のモデルが提案され，そのモデルに基づくバージョン管理システムが実装されている．従来のバージョン管理システムの実装には，バージョンをツール等を用いて明示的に登録する実装とファイルシステムレベルで実装したものの2種類が存在する．しかし，ツールの利用方法を学ぶ必要があり，誤った操作を行った場合にはバージョンの記録が行えない，ファイル構造が専用の形式であり利用する際には制約が大きい，という問題点が指摘されている．

本研究では，上述の問題点を解消した新たなバージョン管理ファイルシステムVCFSを提案し，その実装と評価を行うことを目的とする．基本方針として，導入が容易であることから，バージョン管理機構をファイルシステムとして実装する．また，既存のファイルシステムを用いた堆積型(stackable)ファイルシステムとして実装する．堆積型にすることで，ファイル構造が専用の形式に固定されるという欠点が解消される．更に，ファイルシステムが用いるバージョン管理モデルをファイルシステムから独立させることにより，複数の異なる管理ツールを使い分けられるようにする．最後に，VCFSを，BSD UNIX系のオペレーティングシステムであるFreeBSD 3.0-RELEASEを対象にして実装し，バージョンの読み出し時間，書き込み時間，アプリケーションのコンパイル作業を行う場合の処理時間，ある一連のプログラム開発において蓄積されるバージョンデータの量の点で評価した．その結果，十

分に高速な動作が行え，バージョン管理機能を容易に利用できることが確認できた．

主な用語

ソフトウェア構成管理 (Software Configuration Management)
バージョン管理 (Version Control)
ファイルシステム (File System)

目 次

1 はじめに **5**

2 バージョン管理システム **7**

2.1 バージョンモデル 7

2.1.1 バージョンと差分 7

2.1.2 バージョン付け 8

2.2 従来のシステム 9

2.2.1 RCS 10

2.2.2 3D Filesystem 10

3 バージョン管理ファイルシステム VCFS **12**

3.1 VCFSの概要 12

3.2 VCFSの機能 14

3.3 堆積型ファイルシステム 15

3.4 バージョン管理サブシステム 17

3.4.1 RCS 17

3.4.2 VCS 17

3.5 管理コマンド群 20

3.6 実装 20

3.6.1 実装する機能 20

3.6.2 コードサイズ 23

4 評価 **24**

4.1 性能テスト 24

4.2 容量テスト 32

4.3 考察 35

5 あとがき **36**

謝辞 **37**

付録 **38**

1　アプリケーションのコンパイルの評価データ　39

2　要領テストデータ　41

参考文献　44

1　はじめに

ソフトウェア開発過程で作成されるプロダクトの識別や制御，状態の把握等を行うことを目的として，ソフトウェア構成管理に関する研究が広く行われている[3]．構成管理は開発過程の各段階で行なわれる．設計段階では構成要素の決定や確認，コーディング段階では生成物の変更状態の追跡，リリース段階ではリリースされる生成物の特定を，それぞれ効率良く行なうことを目的とする．

特に，コーディング段階以降では，開発チームが作成したプロダクト (ソースコードや付随する文書) に対する様々な修正を正しく識別し，組織化し，管理することが中心となる．プロダクトは複数回の改訂が行われる．その際，各改訂毎に作成されるプロダクトをバージョンと呼ばれる．そのため，通常，プロダクトの管理はバージョン単位で行なわれる．このバージョンに基づいてプロダクトの作成/変更作業の逐次記録や追跡をうまく行なうために，様々な管理手法のモデルが提案され，そのモデルに基づくバージョン管理システムが実装されている．

従来のバージョン管理システムの実装には，バージョンをツール等を用いて明示的に登録する実装とファイルシステムレベルで実装したものの 2 種類が存在する．前者のバージョン管理システムとして RCS や SCCS がある[20, 21]．これは，開発者の作業対象となるファイルとは別に，各バージョンを記録するためのファイルが存在し，これを操作するツール群によって構成される．各バージョンは作業者がツールを用いて明示的に checkin, checkout の動作を指示することによって記録される．しかし，これらのツールを利用するにはツール等の利用方法を学ぶ必要があり，誤った操作を行った場合にはバージョンの記録が行えないといった欠点がある．一方，後者のシステムレベルで実装したものとして 3D Filesystem がある[12]．ファイルシステムとバージョン管理機能が組みあわさっており，これと連携するツール群によって構成される．バージョンはファイル名で区別され，ツールによって明示的に指示することでバージョンが定義，記録される．しかし，ファイル構造が専用の形式であり，バージョン管理モデルが固定であるため利用する際には制約が大きい．また，研究段階の物であり，実用性に欠けるという問題がある．

そこで，本研究では，実装の問題点を解消するための，新たなバージョン管理ファイルシステム VCFS を提案し，その実装と評価を行うことを目的とする．基本方針として，導入が容易であることから，バージョン管理機構をファイルシステムとして実装する．また，既存のファイルシステムを用いた堆積型 (stackable) ファイ

ルシステムとして実装する [8]．堆積型ファイルシステムは既存のファイルシステム内部の変更は行わない．つまり，ファイルシステムの出力は HDD 等に直接行われず，下層となるファイルシステム上のファイルとして行われる．従って，ファイル構造が専用の形式に固定されるという欠点が解消される．更に，ファイルシステムが用いるバージョン管理モデルをファイルシステムから独立させることにより，複数の異なる管理ツールを使い分けられるようにする．

VCFS を ,BSD UNIX 系のオペレーティングシステムである FreeBSD 3.0-RELEASE [9] を対象にして実装を行った．実装したバージョン管理システム VCFS は，ファイルシステム部分とそれに付随する管理用コマンド群，バージョン管理サブシステムによって構成される．堆積ファイルシステムの下層ファイルシステムとして UNIX ファイルシステムを利用した．システムは C 言語で記述されており，全体で 5000 行程度である．更に，VCFS をバージョンの読み出し時間，書き込み時間，アプリケーションのコンパイル作業を行う場合の処理時間，ある一連のプログラム開発において蓄積されるバージョンデータの量の点で評価した．その結果，十分に高速な動作が行え，バージョン管理機能を容易に利用でき，また，利用するバージョン管理機構を選択できるため，開発環境に応じた管理手法を適用することが可能であることが確認された．

以降，2. ではバージョン管理モデルで用いられている一般的概念，及び従来のバージョン管理システムとその問題点について述べ，3. では今回提案するバージョン管理システムの設計と実装について述べ，4. ではそのバージョン管理システムの評価と考察について述べ，最後に 5. でまとめと今後の課題について述べる．

2　バージョン管理システム

ソフトウェア開発過程において作成されるプロダクト (ソースコード，マニュアルや付随する文書等) の新規作成や，既に作成されたプロダクトの修正等，プロダクトに対する作業がどのように行われているかを識別し，組織化した上で管理するために，バージョン管理システムが広く用いられている．バージョン管理システムでは，プロダクトの修正作業を開発者間で調整することによって間違いのない修正を行うことを可能とする [1, 5]．

本節では，バージョン管理システムにおいて用いられているバージョン管理モデルで用いられている一般的概念について説明する．また，従来の研究にて提案されているバージョン管理システムである RCS と 3D Filesystem を取りあげ，バージョン管理システムの構成とその問題点について説明する．

2.1　バージョンモデル

バージョンモデルは，管理対象となるプロダクトの各状態 (バージョン, version) を識別し，組織化した上で，プロダクトの任意のバージョンの生成，また，新規バージョンの生成方法について定義したものである．バージョン間の相互の関係はバージョン空間 (version space) として定義されるため，バージョンモデルはバージョン空間に関する定義について述べたものとなる．

2.1.1　バージョンと差分

バージョンモデルでは，管理対象となるプロダクト，プロダクトの持つ共通の属性，そしてプロダクトのバージョン間における差分 (delta) のそれぞれが定義される．ここでは，バージョンと差分の定義ついて述べる．

バージョン　バージョンとは，開発作業の進行と共に変化するプロダクトのある状態を指す．直感的には，ある変更による結果作成されたプロダクトそれぞれがバージョンとなる．バージョン管理されていないプロダクトは，ある特定のバージョンしか存在しないプロダクトと考えることができ，その変更は上書きによってのみ行われると考えることができる．

バージョン付けを行う場合，「何をもって同一とみなすか」は考慮されるべきである．つまり，ある 2 つのプロダクトが存在した場合，これが「全く異なる 2 つの

プロダクト」か「同一のプロダクトであり，その内容が異なる」が判断されなければならない．このような同一性問題については，プロダクトに対して一意の識別子 (identifier) を与えることによって解決できる．同様に，同一プロダクトに対する各バージョンにも，そのバージョンを意味する識別子を与える．従って，あるバージョンはプロダクトの識別子とバージョンの識別子を持つ．バージョン識別子の定義は自動的に生成されるものと恣意的に与えられる物が存在する．

差分　同一のプロダクトに対するある任意の 2 つのバージョンの相異点を差分 (delta) とする．一般的に差分は (対応する英語である delta と名付けられているように) ごく小さな量でなければならない．バージョン v_1 と v_2 に対する差分は，主に 2 種類の方法によって定義される．

- 相称的差分 (symmetric delta)
  「v_1 にあり，v_2 に存在しない部分」と「v_2 にあり v_1 に存在しない部分」の和集合として定義するもの．

- 方向的差分 (directed delta)
  v_1 に対してある一連の原始的操作 $op_1, op_2, ..., op_m$ を適用した結果 v_2 が作成されたとみなし，差分を $op_1, op_2, ..., op_m$ のような操作系列として定義するもの．

実際には差分は小さくなる必要はなく，また，小さくなりえないことがある．例えば，あるソースコード中で定義された関数群のインターフェイスが常に同一であり続けることは考えにくく，また，主要な部分の変更を行う場合にはその差分は一般に大きくなると言える [4]．

2.1.2　バージョン付け

バージョンとその差分の定義については前節で述べた．ここでは，複数のバージョンに対するバージョン識別子の定義や，バージョン自体の定義方法について述べる．

バージョン識別子　各バージョンに対して，個別に識別子を付けることにより各バージョンを識別することについては先に述べた通りであるが，その識別子の名前付けの方法には大きく分けて 2 種類の方法がある．これらの名前付けは互いに相反するものではなく，両方を同時に用いることも可能である．

- 数え上げ可能な識別子
  これは，バージョン識別子として，ある事前に定められた一連の識別子を用いる方法である．この定義では一般的に数字や数字の組み合せ (1.1 など) が識別子として用いられる．このような定義では，過去の任意の時点におけるバージョンを簡単に識別することが可能となる．

- 述語によって導出される識別子
  これは，ある述語 $c()$ を定義し，その述語を真にするような識別子を用いる方法である．この定義では一般的に文字列を識別子として用いられる．任意の識別子名を自由に定義し用いることが可能となる．

バージョンの定義　すでに，各バージョンはプロダクトに対するある状態を指すものとして定義されることを述べた．ここではバージョンの定義に必要となる「プロダクトの状態」の定義について述べる．

プロダクトの状態を考える場合には，大きく 2 種類の方法がある．

- プロダクトそれ自体で定義するもの
  例えばファイル等の場合，ファイルの中身すべてをそのまま状態として考えることができる．この定義の場合，状態の変更内容とプロダクトの持つ内容の変更は一致したものとなる．

- 変更内容の集合を用いて定義するもの
  プロダクトのある状態を「ある基準となる状態」と「基準となる状態からの変更内容」の集合として捉える方法がある．変更内容はある特定の識別子によって識別され，プロダクトの状態は変更内容を表わす識別子の集合として定義される．

2.2 従来のシステム

ここでは，従来提案されているバージョン管理システムである RCS と 3D Filesystem について，どのようなバージョン管理モデルが用いられ，またどのような実装が行われているかについて説明する．さらに，既存のシステムが抱える問題点について考察を行う．

2.2.1 RCS

RCS は UNIX 上の動作するツール群として開発されたバージョン管理システムであり，現在広く用いられている物である [21] . RCS ではプロダクトをそれぞれ UNIX 上のファイルとして扱い，1 つのファイルに対する記録は別の 1 つのファイルとして扱われる．

RCS におけるバージョンは，管理対象となるファイルの中身それ自身によって定義され，バージョン間の差分は UNIX における diff コマンドの出力 (編集コマンドの列として表わされる) として定義されている . バージョンに対する識別子は数字の組で表現され，数え上げ可能な識別子となっている . 新規バージョンの登録や，任意のバージョンの取り出し等は，RCS の持つツールを利用することによって行う .

しかし，RCS はツールとして実装されているため，RCS を利用する場合には RCS の持つさまざまなツールの利用方法を学ばなければならない . また，新規バージョンの定義はツールを明示的に起動することによって行うため，バージョン間の差分が大きくなりすぎる可能性がある . また，誤った操作を行うと，記録されたバージョンが破壊されてしまう危険性がある .

2.2.2 3D Filesystem

3D Filesystem は UNIX SystemV Release 3 のカーネルを変更することによって，ファイルシステム上に実装されたバージョン管理システムである [12] . 3D Filesystem では RCS と同様，ファイルシステム上のファイルをプロダクトとして扱っている .

3D Filesystem におけるバージョンモデルは RCS と良く似た物となっている . つまり，ファイルの中身自身を用いてプロダクトの状態を定義し，バージョン識別子は数字の組みで表される数え上げ可能な識別子となっている . ファイルシステムにバージョン管理システムを組み込むことにより，別途ツールを用いることなく任意のバージョンを取り出すことが可能となっている . 新規バージョン等の登録については，3D Filesystem と連携して動作するツール群によって行う .

3D Filesystem は RCS の持つ欠点を補った物となっているが，ファイル構造が 3D Filesystem 固有の構造となっているため，保存された情報は 3D Filesystem でしか用いることができない . また，バージョン管理モデルは 3D Filesystem が定義する物しか用いることができないため，利用する際には制約が大きいという問題がある . また，3D Filesystem は現在研究段階のシステムであり，実用性に欠けると

いった点も上げられる．

本研究では，従来のバージョン管理システムの持つ問題を解消するための，新たなバージョン管理システム VCFS の提案を行う．

3 バージョン管理ファイルシステム VCFS

バージョン管理ファイルシステム(VCFS)はファイルシステムにバージョン管理機能を持たせたものである．バージョン管理モデルにはcheckin/checkout モデルを採用した[6]．また，ファイルシステムとしての機能とバージョン管理を行う機能を，それぞれカーネルの内部とユーザプログラムに分けることでより柔軟なバージョン管理を行うことが出来る．

3.1 VCFSの概要

VCFS(Version Cntrol File Ssystem)はファイルシステム部分とそれに付随する管理用コマンド群，バージョン管理デーモン，バージョン管理サブシステムによって構成される．図1は各構成要素のつながりを表したものである．各構成要素は以下のような動作を行う．

- ファイルシステム(VFS)
  ファイルシステムとしての機能をもつ部分であり，カーネル内のプログラムである．このファイルシステムに対してユーザプロセスからファイルの読み出し，書き込み，ディレクトリの作成等といった一般のファイルシステムの持つ機能を行うだけで，バージョン管理機能が動作する．

- バージョン管理デーモン (VCD)
  カーネル内のファイルシステムとのやりとりを行うデーモンプログラムである．カーネルから依頼されるファイルのバージョン管理を引き受ける．ファイルに対する実際のバージョン管理は，バージョン管理サブシステムに依頼する．

- バージョン管理サブシステム
  実際にバージョンを記録する部分である．バージョン管理サブシステムは複数のバージョン管理ツールの集合体である．バージョンを記録を行う場合，いずれかが選択されてそのツールを実行する．

- 管理用コマンド群
  過去のバージョンの参照やバージョン間の差分情報の取り出しなどの通常のファイル操作だけでは出来ない操作を補助するコマンド群である．

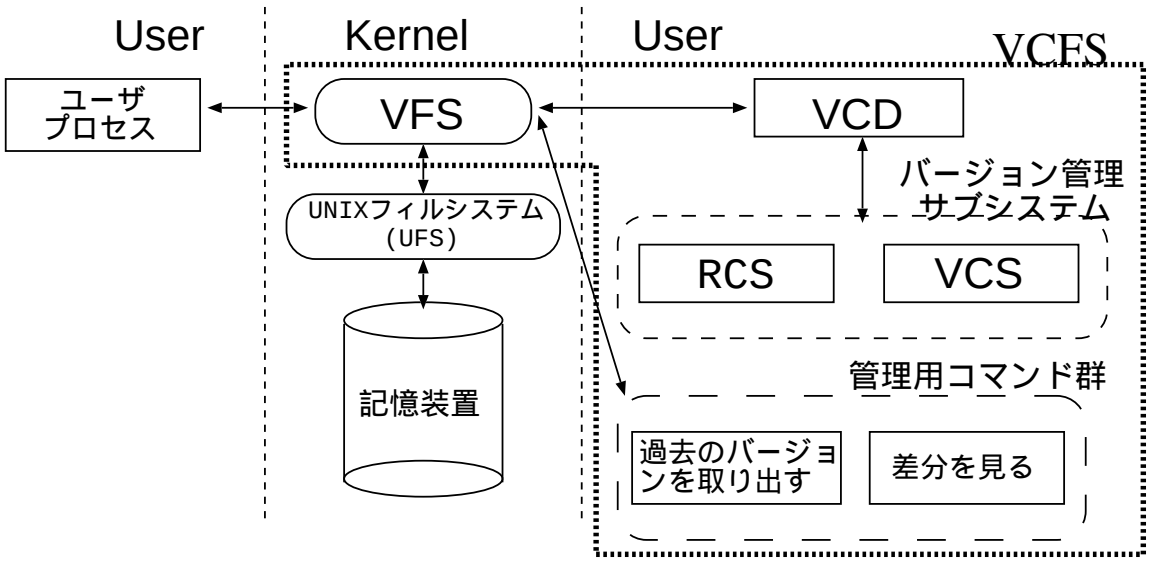


図 1: VCFS の構成

バージョン管理モデルには RCS などで用いられている checkin/checkout モデルを採用した .これは ,バージョン管理を特定のツールに依存するのではなく checkin/checkout モデルに基づいたツールであればそれが使用可能な事を意味する．今回の実装においては，RCS と VCS という二種類の管理方法を提供する．

あるディレクトリ以下をバージョン管理ファイルシステムでマウントすると，マウント元にはバージョンを記録したファイルと最新バージョンが存在し，マウント先には最新バージョンのみが見えるようになる．

実際のファイル操作は，ファイルを操作するユーザプロセスにとっては一般のファイルシステム上のファイルを操作するのと同様の操作で可能である．

常に最新バージョンのファイルをファイルシステム上に checkout しておく．そのため，ファイルの読み書きは高速に動作する．ファイルを読み出し専用で開いた場合は，従来のファイルシステム上と同じ動作をする．それに対して，書き込みフラグを付けてファイルを開いた場合，読み出し，書き込みといった操作は従来と同じであるがファイルに対して閉じる動作 (close システムコール) を行ったとき，そのファイルに対して checkin 動作を行う．この checkin の動作はファイルシステムの中には定義しておらず，カーネルの外のバージョン管理デーモンとバージョン管理サブシステムが行う．

カーネル内に記述しない利点として，checkin/checkout の動作をカーネルを変更せずに切替が出来る事があげられる．また，RCS 等のプログラムを使ってバージョン管理を行っていた場合，RCS で使われている機能をバージョン管理サブシステムのプログラムとして記述することで，今回のファイルシステムにおける実現においても以前のデータがそのまま使えるようになる．checkin/checkout 用のプログラムを，独自の形式で書くことも可能であり，バージョンを記録ファイルに必要な情報を付加したバージョンを付加することが出来る．

3.2 VCFS の機能

VCFS では，通常のファイルに対する読み出し，書き込み動作を直接バージョン管理作業に対応付ける．既存のエディタ等によるファイルの読み書き作業だけでバージョン管理機能が利用できる．通常ファイルのみバージョンを取り，ディレクトリ，シンボリックリンク，特殊デバイスファイル，ソケット，名前付きパイプなどは取らない．バージョンを取るのは，新規にファイルを作成した場合や，既存のファイルを変更した場合であり，それらを新規バージョンの登録作業として扱う．ファイルを読み出す場合，最新のバージョンを自動的に取り出してそれを読める．

VCFS のマウント元のファイルシステムには最新バージョンのファイルとバージョン記録ファイルを保持している．マウント先では最新バージョンのファイルのみ操作でき，バージョン記録ファイルは見えないようになっている．例えば，/versiondb を /proj にマウントしたとする．そして，/proj/foo というファイルを作成した場合，/versiondb には，そのファイル foo と foo のバージョン記録ファイルが作成される．この際，/versiondb には通常ファイルは ,a をバージョン記録ファイルには ,v をファイル名の最後に付加したファイル名として保存する．図 2に示すように/proj/foo は /versiondb/foo,a というファイルを指すことになる．また，/versiondb/foo,v は /proj に下では見ることが出来ない．

通常，バージョン履歴ファイルはファイルシステムからは操作を行わないが，ファイル名変更 (rename システムコール)，削除 (remove システムコール) についてはバージョン履歴ファイルに対してもその操作を実行する．

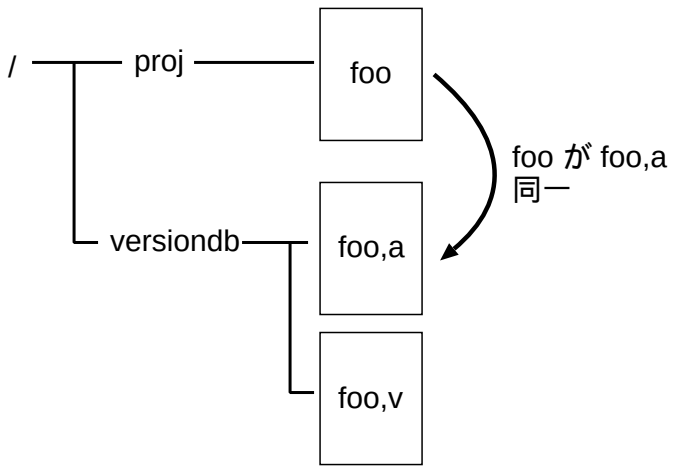


図 2: ファイルシステム上のファイル

3.3 堆積型ファイルシステム

ファイルシステムには直接物理装置を操作する UFS[18] や LFS[2] といったものや，ネットワークを用いた NFS といったものがある．VCFS で用いられるファイルシステムは，既存のファイルシステムの上に構築する堆積型ファイルシステム [8] として設計した．

堆積型ファイルシステムは vnode[10] と呼ばれるデータ構造を用いている．この構造は近年の UNIX オペレーティングシステムでファイルやディレクトリなどを表現するのに使用されている．

堆積型ファイルシステムの特徴としてつぎのようなものが挙げられる．

- 堆積型ファイルシステムは既存のファイルシステム内部の変更は行わない．
- ファイルシステムの出力は HDD 等に直接行われず，下層となるファイルシステム上のファイルとして行われる．
- 記憶装置やネットワークを処理するコードを書く必要ががなく，移植が容易である．

図 3に堆積型ファイルシステムの構造を示す．ユーザプロセスからのファイルに対するシステムコールはすべて vnode 層の呼出し変換される．そして，その呼び出しはファイルの存在するファイルシステムの関数を実行する．もし，ファイルが堆積

型ファイルシステム上にあれば，堆積型ファイルシステムの各操作を呼び出す．堆積型ファイルシステム層の各操作は，下層のファイルシステムの対応する各操作を呼び出す．

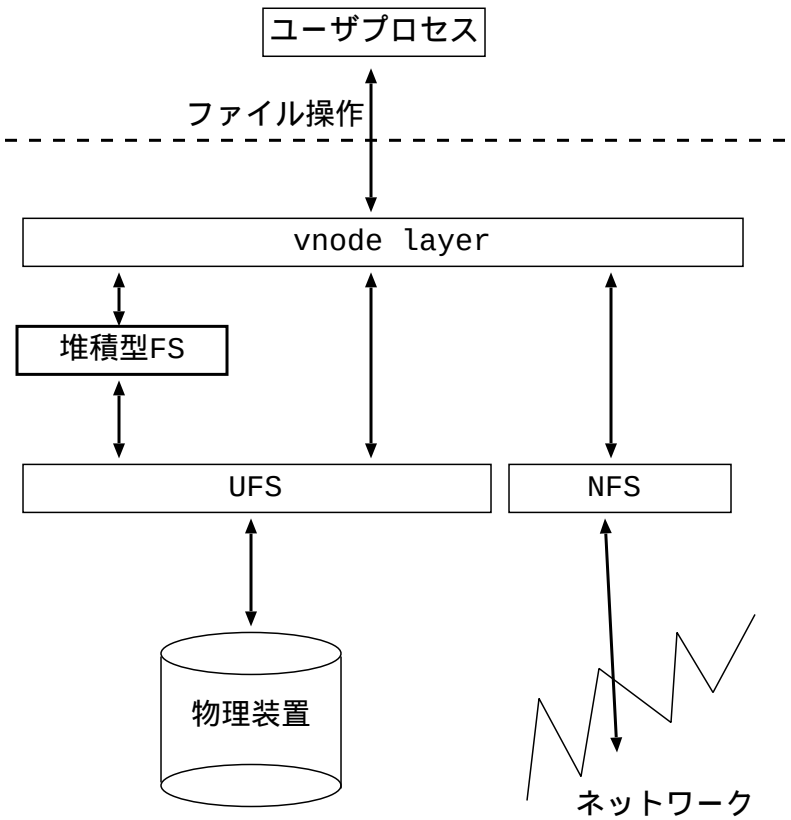


図 3: 堆積型ファイルシステム

以下に堆積型ファイルシステムの例を挙げる．

- NULLFS
  処理を直接下層に渡し，それ自身は何も行わないファイルシステムである．マウントを行うと，マウント元のディレクトリ以下がマウント先のディレクトリ以下ですべての操作可能となる．NULLFS では下層へ渡すときに何も処理を行わないので，どちらのディレクトリも同一に見える．

- UMAPFS
  uid と gid のみをを変更して処理を直接下層に渡すファイルシステムである．

ファイルシステムのマウント時にuidとgidの対応ファイルを指定する．たとえば，下層ファイルシステムにおいてuid 1000のファイルがあるとする．対応ファイルに1000から2000と記述する．そのファイルを指定してマウントすると，uid 1000のファイルは、実際にはuid 2000が所有者であるかのように見える．

- UNIONFS[19]
  複数のディレクトリを重ねあわせ，同一のディレクトリとして見せることができるファイルシステムである．マウントを実行すると，マウント元のディレクトリ以下がマウント先のディレクトリ以下に重ね合わせる．つまり，あるファイルを指定した際にマウント先にファイルが存在すればそのファイルを，マウント先に存在せずにマウント元に存在すればそのファイルを返す処理を行う．

3.4 バージョン管理サブシステム

VCFSでは，バージョン管理サブシステムとして，既存のバージョン管理システムを用いており，ファイルシステムを利用する際に，どのシステムを利用するかが選択できる．今回の実装ではRCSとVCSの2種類のバージョン管理サブシステムが利用可能である．

3.4.1 RCS

RCSは各バージョンの差分のみを保存するために広く用いられているツールである．RCSではバージョンの派生(木構造)を許す．この派生された枝をブランチと呼ぶ．RCSのバージョンの木構造の例を図4に示す．

このサブシステムではバージョン記録ファイルに対する操作はすべてRCSのコマンドを内部で実行することでバージョンを記録する．

3.4.2 VCS

今回作成した単純な機能を持つバージョン管理システム．各バージョンを圧縮しない形でバージョン間の差分もとらずにそのまま保存する．バージョンの派生はなく，線型である．RCSと違いバージョン間の差分情報をとらないことで高速に動作させることを目的としている．

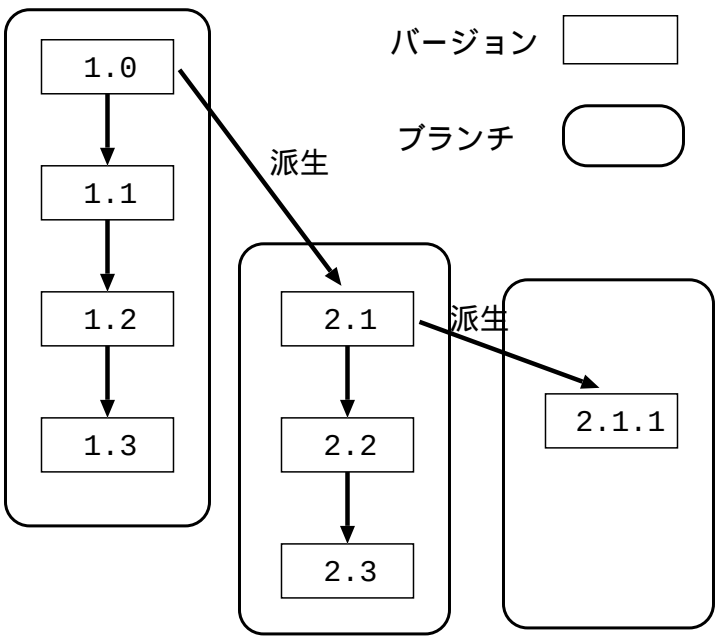


図 4: バージョンの木構造

VCS のバージョン履歴ファイルのファイル形式を図5にファイルのヘッダを表1にバージョンヘッダを表2に示す．checkin 動作は登録するバージョンをバージョンヘッダと共にバージョン履歴ファイルに追加していく．ただし，ヘッダののバージョン番号と最新オフセットは更新しておく．過去のバージョンはヘッダの最新オフセットと各バージョンファイルのバージョンヘッダの前オフセットを辿ることで取得可能となる．

表 1: ヘッダ構造

名前	サイズ	内容
バージョン番号	4バイト	最新バージョン番号を表す．初期バージョンを 1 として以降 2，3，... と続く．
最新オフセット	16バイト	最新バージョンのヘッダのオフセットを表す．

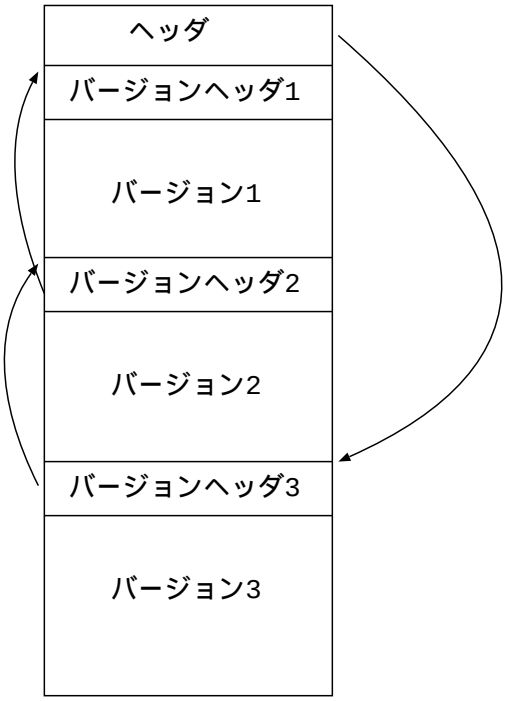


図 5: VCS のファイル形式

表 2: バージョンヘッダ構造

名前	サイズ	内容
属性	vattr 構造体のサイズ	ファイルの属性を表す vattr 構造体 .
前オフセット	16 バイト	前バージョンのヘッダのオフセットを表す .

3.5 管理コマンド群

通常のファイル操作では行えない操作をするためのコマンド群である．
コマンドとしては以下のものがある．

- 任意のバージョンファイルの取り出し
  リビジョン番号や日付などを指定し，過去のバージョンファイルを取り出すコマンドである．
- ブランチ作成
  RCS などバージョンの派生を許すモデルの場合に，派生を行うコマンドである．VCS など許さない場合はこのコマンドを実行しても何も起こらない．
- 差分の閲覧
  バージョン間の差分を取り出すコマンドである．

3.6 実装

BSD UNIX[13, 17] 系のオペレーティングシステムである FreeBSD 3.0-RELEASE [9] を対象にして実装を行った．

3.6.1 実装する機能

堆積型ファイルシステムの下層ファイルシステムとして UNIX ファイルシステム (UFS) を利用する．
以下の関数をファイルシステムとしての機能の実装をカーネルに追加した．

- vc_lookup
  ファイル名からカーネル内部で必要なデータ構造に変換する関数である．ファイル名の変換を行い下層ファイルシステムに処理を渡す．
- vc_access
  ファイルを操作する時に，操作可能化判断する関数である．読み出し専用でマウントされた時に，書き込みを行王とした場合はエラーを返す．それ以外の場合は下層ファイルシステムに処理を渡す．

- vc_setattr

  ファイルに関する属性の設定を行う関数である．設定する値が無効であったり読み出し専用でマウントされた時にはエラーを返す．それ以外の場合は下層ファイルシステムに処理を渡す．

- vc_close

  closeシステムコールに対応する関数である．ファイルが書き込みフラグ付きで開かれていた場合checkin動作を行い，下層ファイルシステムに処理を渡す．

- vc_getattr

  ファイルに関する属性の取得を行う関数である．下層ファイルシステムに処理を渡した後，ファイルシステムのidをマウント先に変更する．

- vc_read

  read システムコールに対応し，ファイルからの読み出しを行うする関数である．そのまま，下層ファイルシステムに処理を渡す．

- vc_write

  write システムコールに対応し，ファイルからの書き込みを行うする関数である．そのまま，下層ファイルシステムに処理を渡す．

- vc_mkdir

  mkdir システムコールに対応し，ディレクトリを新しく作成する関数である．ファイル名の変換を行い下層ファイルシステムに処理を渡す．

- vc_rmdir

  rmdir システムコールに対応し，空のディレクトリを削除する関数である．そのまま，下層ファイルシステムに処理を渡す．

- vc_create

  create システムコールに対応し，新しくファイルを作成する関数である．ファイル名の変換を行い下層ファイルシステムに処理を渡す．

- vc_remove

  unlinkシステムコールに対応し，ファイルをディレクトリエントリから削除す

る関数である．下層ファイルシステムに処理を渡した後，バージョン履歴ファイルも削除する．

- vc_rename
  rename システムコールに対応し，ファイル名の変更を行う関数である．ファイル名の変換を行い下層ファイルシステムに処理を渡す．その後，バージョン履歴ファイルのファイル名も変更する．

- vc_symlink
  symlink システムコールに対応し，シンボリックリンクファイルの作成を行う関数である．ファイル名の変換を行い下層ファイルシステムに処理を渡す．

- vc_readlink
  readlink システムコールに対応し，シンボリックリンクファイルに格納されているデータを読み出す関数である．ファイル名の変換を行い下層ファイルシステムに処理を渡す．

- vc_readdir
  getdents システムコールに対応し，ディレクトリからデータを読み出す関数である．下層ファイルシステムに処理を渡した後，読み出したデータのうちファイル名は変換を行い返す．

ファイルシステムとしての機能の他に以下の機能を実装した．

- システムコール
  バージョン管理デーモンとファイルシステムとのインタフェースとなる．バージョン管理デーモンはこのシステムコールを用い，新規バージョンとして登録するファイルがあるかを調べる．

- checkin 依頼機能
  ファイルを書き込みフラグを付けて open していた場合，そのファイルの close 時に呼び出される．ファイル名を調べ，そのファイルを新規バージョンとして登録するようにバージョン管理デーモンに依頼する．

- ファイル名変換機能
  マウント元のファイル名とマウント先のファイル名の相互変換を行う．

- vnode からパス名変換機能
  vnode からファイルのフルパス名の変換．

- 仮想デバイス
  通常のファイル操作だけでは出来ない操作を行うためのものである．管理用コマンド群はこの仮想デバイスを利用する．

3.6.2 コードサイズ

実装はすべて C 言語で記述し，全体で 5000 行程度になった．それらの内訳はつぎのようになる．

- 堆積ファイルシステム (カーネルへの追加) 4500 行
- バージョン管理デーモン (VCD) (新規作成) 340 行
- その他 (バージョン管理サブシステムとのインタフェース等) 300 行

4 評価

本研究で実装したバージョン管理ファイルシステム (VCFS) の評価を行う．ファイルシステムとしての実用性を考えるうえで有効かどうかを調べる．評価の方法としては，ファイルシステムとして通常用いられる UNIX ファイルシステム (UFS) と堆積型ファイルシステムの一つである NULLFS との比較することで行った．

ファイルシステムの評価として 2 種類のテストを行った．

- 様々な大きさのファイルの読み出し，書き込みのようなファイルシステムの性能に関するもの．
- ファイルシステム上に蓄えられるファイルの大きさに関するもの．

以下で述べるテストはすべて Pentium 166MHz・48MB RAM の計算機で評価した．

4.1 性能テスト

1KB から 8MB までの間の大きさのファイルの読み込みテストを行った．それぞれのファイルシステム上に各大きさのファイルを用意し，read システムコールを用いファイルをメモリ上に全て読み込むプログラムの実行時間の測定を行った．

測定結果を表 3 と図 6 に示す．

表 3: ファイルの大きさの違いによる読み出し時間 (Sec)

size(byte)	1024	2048	4096	8192	16384	32768	65536	131072
VCFS	0.05	0.02	0.05	0.04	0.06	0.05	0.08	0.2
NULLFS	0.07	0.04	0.03	0.04	0.02	0.06	0.09	0.2
UFS	0.05	0.05	0.03	0.05	0.04	0.02	0.08	0.1

size(byte)	262144	524288	1048576	2097152	4194304	8388608	16777216	
VCFS	0.22	0.37	0.77	1.29	2.52	5.45	12.91	
NULLFS	0.34	0.6	0.68	1.38	3.31	5.22	11.23	
UFS	0.13	0.35	0.82	1.44	2.31	4.63	9.41	

これらの結果から，いずれの大きさのファイルに対しても各ファイルシステムでほとんど差が見られないことが分かる．ファイルの大きさにより各ファイルシステムで読み出し時間の差はほとんど変化はなく，ファイルの大きさによって性能が著しく劣ることはない．また，VCFS でおいていづれのバージョン管理サブシステ

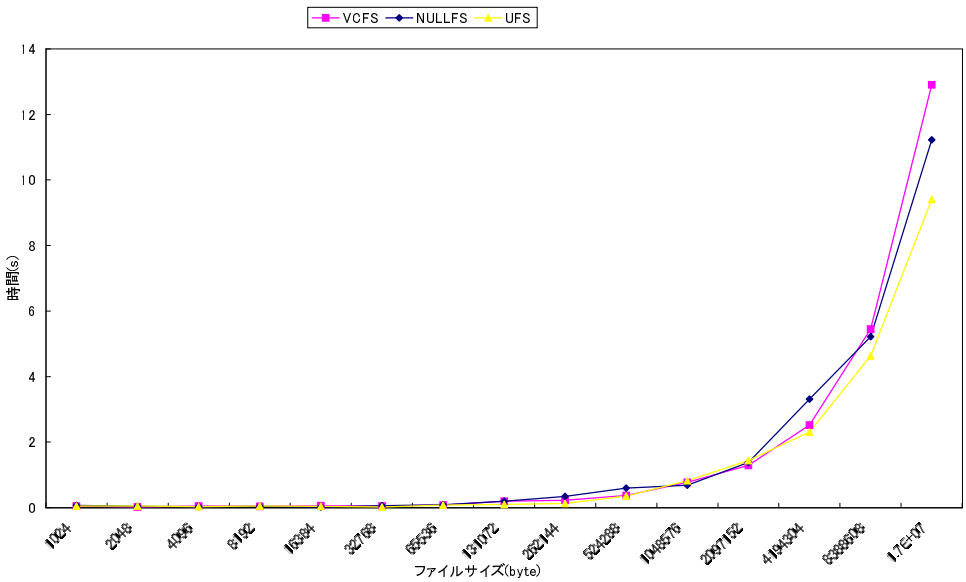


図 6: ファイルの大きさの違いによる読み出し時間

ムを用いても VCFS を使う限り同じ測定結果は値になる．最新バージョンのファイルは常にファイルシステム上に存在しており，ファイルの読み出しに関してはそのバージョンのファイルを読み込むだけなので，読み出しのみの場合はバージョン管理サブシステムは関与しないためである．

次に，ファイルの大きさが 1MB の異なるファイルを単一プロセスで繰り返し連続して読み出す時の処理時間を計測した．ここで処理時間とは連続した読み出しを行うプロセスの実行時間である．つまり，プロセスの生成から消滅までの時間となる．読み出すファイルは既にファイルシステム上に読み出す回数分だけ用意しておく．1 回だけの読み出しから 10 回連続までの 10 個の値が各ファイルシステムについて存在する．

測定結果を表 4と図 7に示す．VCFS(RCS) と VCFS(VCS) はそれぞれバージョン管理サブシステムとして RCS と VCS を用いたという事を意味する．以降，VCFS(RCS)，VCFS(VCS) は同様の意味で使う．

どの回数においても VCFS(RCS) と NULLFS はほとんど違いが見られない．こ

表 4: 1MB の連続読み出し時間 (Sec)

	1	2	3	4	5	6	7	8	9	10
VCFS(RCS)	0.84	1.41	2.53	3.78	4.9	5.29	6.2	6.6	6.96	8.9
NULLFS	0.92	1.4	2.47	3.74	4.41	5.06	5.93	6.73	6.75	8.49
UFS	0.77	1.44	2.36	3.03	4.54	4.62	5.78	5.93	6.21	7.3

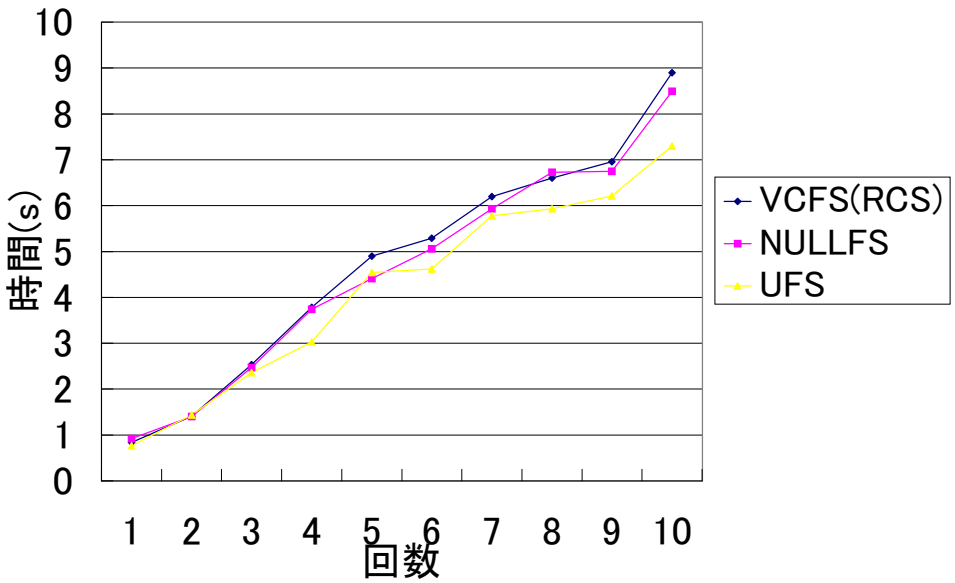


図 7: 1MB の連続読み出し時間

れによりファイルの読み出しに関してはファイルシステム内のバージョン管理機能はほとんど時間がかからないことが分かる．VCFS(RCS)やNULLFSはUFSと比べると約10%の時間を要するがこれは堆積型ファイルシステムとして実装したためであり，堆積型ファイルシステム部分の機能が10%程度であるといえる．一般のファイルシステムと比べても十分に高速である．

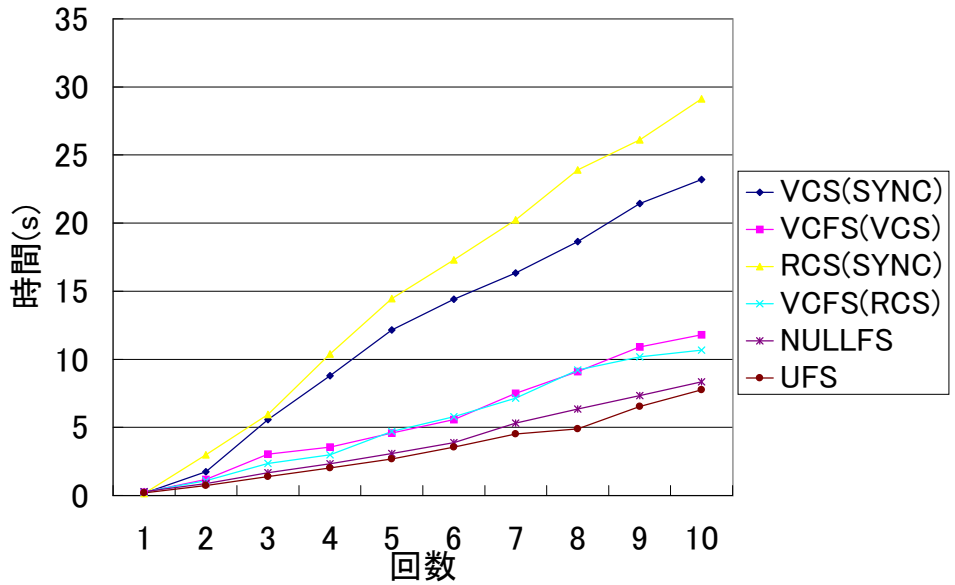
同様のテストを，書き込みに関しても行った．ファイルの大きさを1MBとし，異なるファイル名で単一プロセスで繰り返し連続して書き込む時の処理時間の計測を行った．ここで処理時間とは連続した書き込みを行うプロセスの実行時間である．書き込むファイルはファイルシステム上に存在していないものとする．1回だけの書き込みから10回連続までの10個の値が各ファイルシステムについて存在する．

測定結果を表5と図8に示す．VCS(SYNC)はバージョン管理サブシステムとしてVCSを用い，完全にバージョンを記録するまでの時間を計測したものである．RCS(SYNC)についても同様である．

表 5: 1MBの連続書き込み時間(Sec)

	1	2	3	4	5	6	7	8	9	10
VCS(SYNC)	0.16	1.75	5.57	8.79	12.16	14.4	16.33	18.63	21.44	23.2
RCS(SYNC)	0.15	2.98	5.95	10.39	14.45	17.29	20.24	23.9	26.11	29.13
VCFS(VCS)	0.24	1.18	3.04	3.55	4.59	5.57	7.5	9.1	10.91	11.8
VCFS(RCS)	0.26	1.09	2.36	2.99	4.71	5.78	7.15	9.21	10.18	10.66
NULLFS	0.28	0.86	1.66	2.33	3.07	3.89	5.32	6.35	7.34	8.35
UFS	0.18	0.73	1.39	2.03	2.69	3.55	4.51	4.9	6.54	7.75

UFSと比べ，NULLFSは10%程度の時間が遅い．これは読み出しの際のUFSとNULLFSとの違いと同じ割合である．何も行わない堆積型ファイルシステムでは読み書き共に10%程度の時間を余計に必要なり，読み書き共に十分高速である．VCFSとUFS比べると約30%のVCFSのほうが遅い．読み込み時には10%であったが，書き込みの際にはバージョン管理を行う分の時間を要するためである．しかし，バージョンの記録が完全に終了する時間は倍以上の時間を必要とする．今回の実装ではバージョンの記録はカーネル内では行わず，ユーザプロセスとして実行させるため，30%程度の時間の遅れで書き込みを行うプロセスは終了し，バックグランドでバージョン管理サブシステムが動作しバージョンの記録を行う．バージョンを記録する



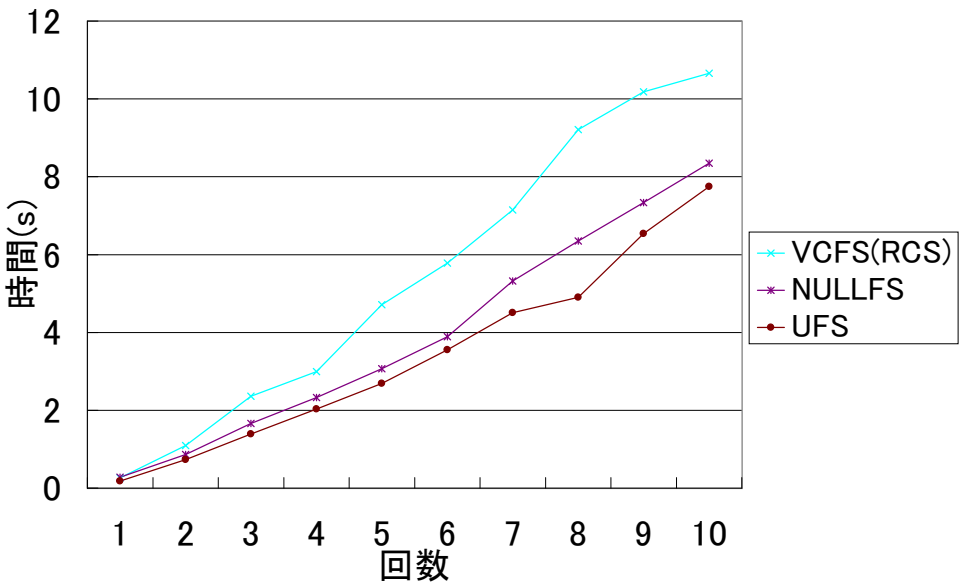


図 8: 1MB の連続書き込み時間

機能をカーネル内部から外したことで高速性をもつファイルシステムが実現出来た．

ファイルの大きさが 32KB にして書き込みテストを行った．測定結果を表6と図9に示す．

基本的には上記の 1MB ファイルの書き込みと同じ傾向が見られるが，ファイルの大きさが小さいときには，バージョン管理サブシステムとして RCS より VCS を用いたほうが高速に動作する．

表 6: 32KB の連続書き込み時間 (Sec)

	1	2	3	4	5	6	7	8	9	10
VCS(SYNC)	0.02	0.13	0.23	0.32	0.46	0.52	0.61	0.73	0.79	0.88
VCS	0.02	0.06	0.15	0.16	0.22	0.29	0.36	0.38	0.48	0.57
RCS(SYNC)	0.02	0.25	0.51	0.75	1	1.39	1.5	1.71	2.04	2.78
RCS	0.02	0.06	0.11	0.16	0.19	0.25	0.37	0.44	0.54	0.69
NULLFS	0.02	0.07	0.09	0.14	0.16	0.26	0.23	0.26	0.27	0.32
UFS	0.03	0.03	0.03	0.03	0.02	0.02	0.02	0.03	0.02	0.01

ファイルの更新時の時間について測定を行った．ファイルの大きさが 1MB のファイルを用意し，各ファイルは RCS で差分が大きくなるように値をそれぞれ 0 から 5 の値で埋めたものにしておく．更新の値が 0 というのは新規にファイルを作成したという事である．測定結果を表7と図 10に示す．

表 7: バージョンの更新による時間 (Sec)

	0	1	2	3	4
VCS(SYNC)	0.75	0.85	0.86	0.87	0.86
VCS	0.18	0.11	0.11	0.11	0.11
RCS(SYNC)	1.91	5.88	7.55	8.6	10.21
RCS	0.2	0.12	0.13	0.11	0.11
NULLFS	0.33	0.21	0.21	0.21	0.21
UFS	0.24	0.12	0.12	0.12	0.12

バージョン管理サブシステムとして RCS を用いた場合を除いて更新したときに要する時間は新規にファイルを作成した時間と同じである．バージョン管理サブシス

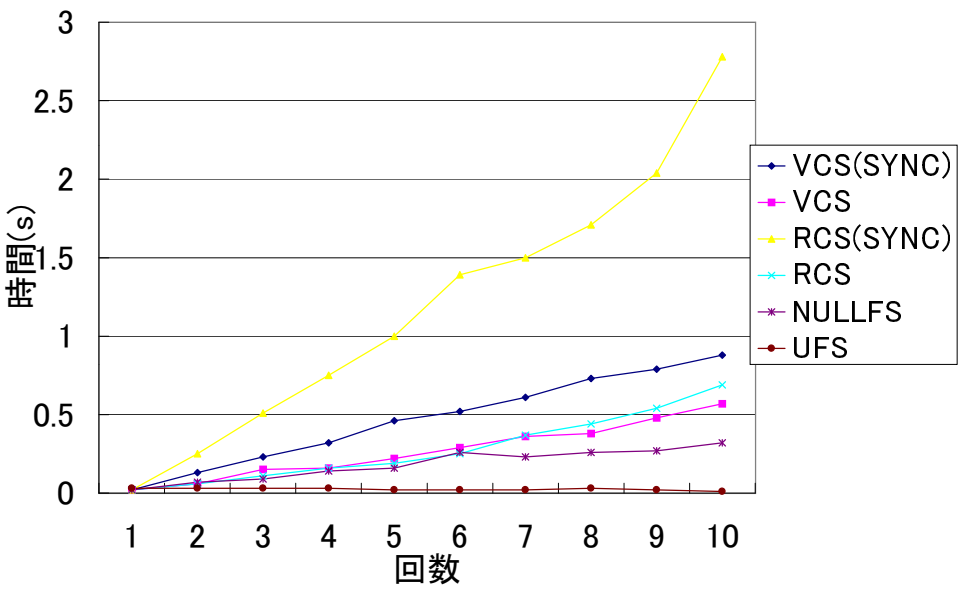


図 9: 32KB の連続書き込み時間 (Sec)

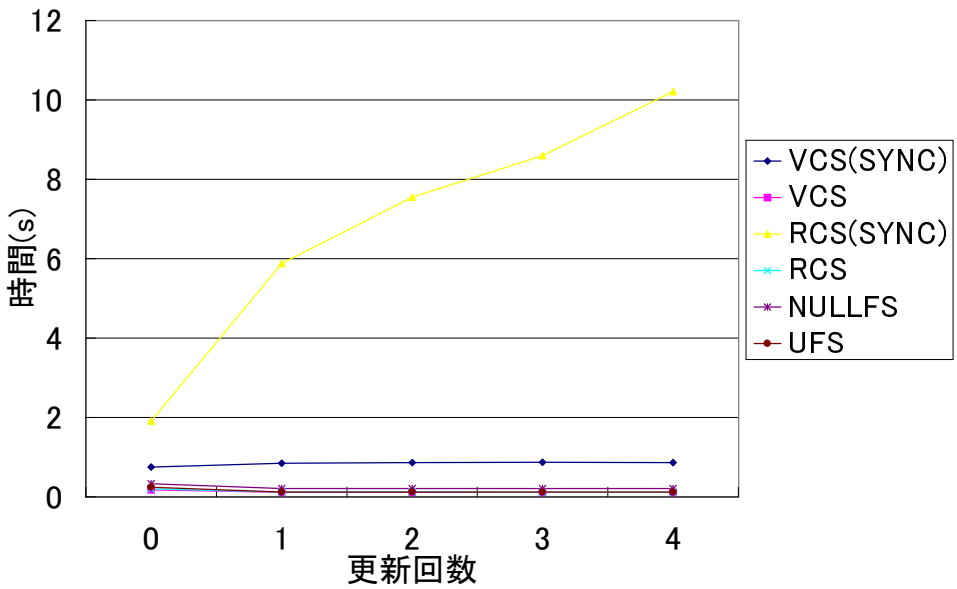


図 10: バージョンの更新による時間

テムVCSは以前のバージョンからの差分は取らずにバージョンの記録を行うため回数によらずファイルの大きさが同じならば，更新する時間はほぼ一定である．RCSの場合，前バージョンとの差分をとろうとするため回数が増えると，更新する時間も増加する．

以上の結果からから，高速性が必要ならばバージョン管理サブシステムにVCSを用いるといい事が分かる．しかし，VCSにもRCSと比べるとバージョンの記録ファイルの大きさが大きくなるという欠点があるため，使い分ける事が重要である．ファイルの容量については次の節でのテストで確認出来る．

実際のアプリケーションのコンパイルに要する処理時間を測定した．測定対象となるアプリケーションとしてはFreeBSDに付属のtarとdumpとした．アプリケーションのソースファイルは各ファイルシステム上に用意しておき，makeの開始から終了までを処理時間とした．測定結果を表8と図 11に示す．

makeが行う作業は，異なるソースファイルのコンパイルの繰り返しである．コンパイルはソースファイルの読み出し，そのソースファイルから生成されたオブジェクトファイルの書き込みであり，ファイルシステムに対して読み書きを行う．VCFSはUFS比べると20%程度の時間を要する．これは，実用上問題ない程度の時間である．

表 8: アプリケーション作成時間(Sec)

	dump	tar
VCFS(VCS)	10.96	28.63
VCFS(RCS)	12.79	33.46
NULLFS	11.3	32.01
UFS	10.9	28.27

4.2 容量テスト

学生実験で得られたプログラムの編集履歴データをVCFSに適用した．編集履歴データから5分間隔でのソースコードのバージョンが復元出来る．ただし，5分以内にファイルが更新されていないときはバージョンを取らない．これらのバージョンファイル時刻順にバージョン管理ファイルシステム上にコピーしていく．

各ファイルシステム上での最終的なソースコードの総容量を表8と図 11に示す．

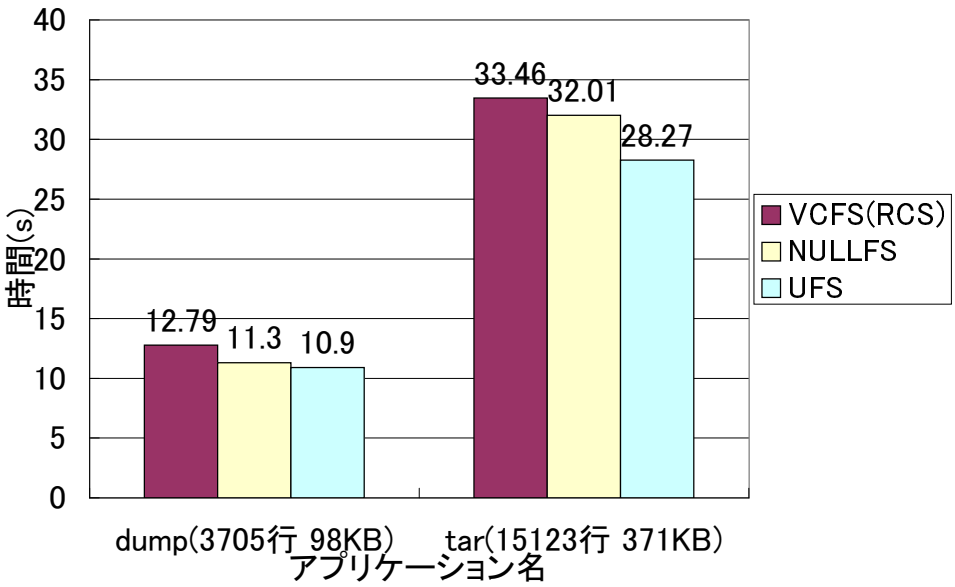


図 11: アプリケーション作成時間 (Sec)

UFS 上の容量が元の大きさであり，この大きさと比べるとバージョン管理サブシステムでの容量は大きくなるが，RCS を用いた場合は数倍程度であり，VCS を用いた場合は数十倍まで増加するが実用上問題ない．

採用するバージョン管理サブシステムによってディスク使用容量が大きく変化し，ディスク容量を重要視するならば，VCS より RCS をバージョン管理サブシステムとして使用すればいい．

表 9: ディスク使用容量 (KB)

	UFS	VCFS(RCS)	VCFS(VCS)
data1(9339 行)	225	657	1784
data2(4067 行)	117	290	899
data3(2543 行)	73	280	1020

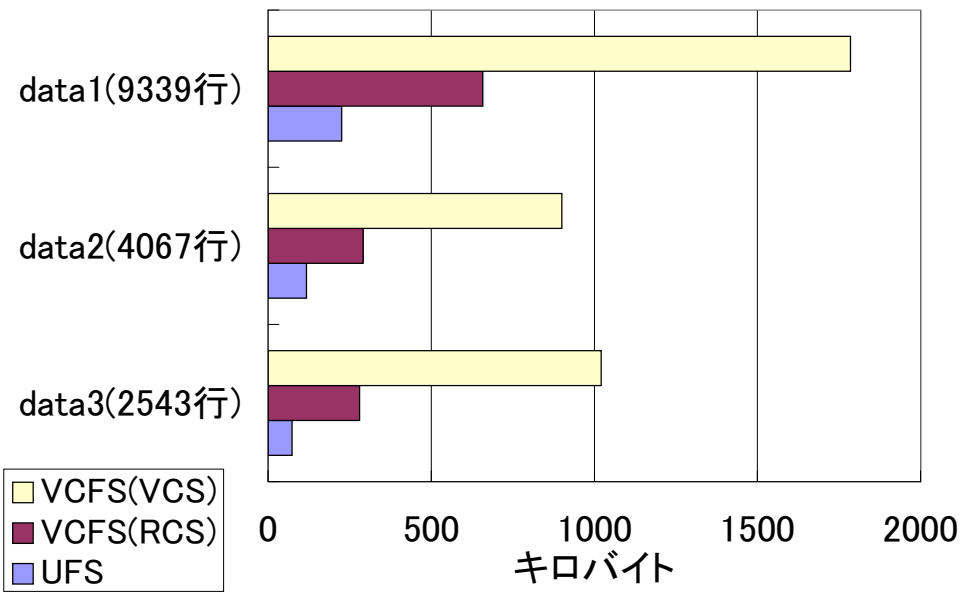


図 12: ディスク使用容量

4.3 考察

性能面およびディスク容量の観点から評価を行った．今回の評価から以下のようなことが分かる．

- バージョン管理ファイルシステムは一般のファイルシステムに比べ20%程度遅いが実用上問題ない．
- ディスク使用容量は数倍程度であり実用上問題ない．
- 高速性を求めるならばVCSを，ディスク使用容量を減らしたいならばRCSをバージョン管理サブシステムに採用すれば良い．

5　あとがき

従来のバージョン管理システムとその問題点について述べた．そして，その問題を解決するバージョン管理機能を持つファイルシステムの設計と実装を行い，システムの評価を行った．本ファイルシステムは，

- 十分に高速な動作が行え，バージョン管理機能を容易に利用できる．
- 利用するバージョン管理機構を選択できるため，開発環境に応じた管理手法を適用することが可能である．

といった利点がある．

分散型，複数人による開発により分散ファイルシステム [7, 11] などの研究が行われている．そこで，今後の課題として分散環境に対応したバージョン管理機構への対応が挙げられる．また，実環境への適用によるユーザビリティの評価を行う．本ファイルシステムを発展させることによって，プロダクトに関する管理を扱いやすくすることが可能となっているが，これをソフトウェアプロセス中心型開発環境 [15, 16, 14, 22] に対して適用することにより，プロセスとプロダクト双方の質を高めるためのソフトウェア開発環境の構築などが挙げられる．

謝辞

本論文を作成するにあたり，常に適切な御指導を賜わりました大阪大学大学院基礎工学研究科情報数理系専攻 井上 克郎 教授に心より深く感謝致します．

本論文の作成において，適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 楠本 真二 講師に深く感謝致します．

本論文の作成において，適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 松下 誠 助手に深く感謝致します．

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院基礎工学研究科情報数理系専攻井上研究室の皆様に深く感謝致します．

付録

1. アプリケーションのコンパイルの評価データ
2. 容量テストデータ

1　アプリケーションのコンパイルの評価データ

tar のファイル

ファイル名	サイズ	行数
COPYING	17982	339
ChangeLog	62125	1738
Makefile	667	17
Makefile.gnu	6480	185
README	1839	40
buffer.c	35197	1584
create.c	34371	1478
diffarch.c	16700	760
extract.c	25250	946
getdate.y	23224	977
getoldopt.c	2310	96
getopt.c	20131	712
getopt.h	4327	125
getopt1.c	3479	161
getpagesize.h	610	38
gnu.c	14028	677
list.c	20292	886
mangle.c	6635	270
msd_dir.h	1060	44
names.c	3302	149
open3.h	2639	67
pathmax.h	1691	53
port.c	25906	1256
port.h	5312	215
rmt.h	3321	98
rtapelib.c	13530	570
tar.1	13134	445
tar.c	38002	1570
tar.h	9491	296
update.c	13900	585
version.c	50	1
合計	42695	16378

dump のファイル

ファイル名	サイズ	行数
Makefile	1097	37
dump.8	11151	400
dump.h	7358	216
dumprmt.c	8779	415
itime.c	6975	274
main.c	16558	635
optr.c	12236	533
pathnames.h	2099	42
tape.c	21825	875
traverse.c	15714	612
unctime.c	3217	103
合計	107009	4142

2　要領テストデータ

data1

ファイル名	最終ファイルサイズ	バージョン数
block.c	5225	17
block.h	5161	5
checker.c	12857	1
code.c	738	8
code.h	738	4
data.c	2170	2
dyalloc.c	869	11
dyalloc.h	869	3
dynamic.h	1027	1
er.c	154	1
error.c	114	19
error.h	117	5
exp.c	15087	15
exp.h	15018	5
igen.c	2637	5
kai.c	3131	3
main.c	1395	29
master.c	6507	2
mmain.c	896	10
mmain1.c	2485	2
originmain.c	2017	2
rep-st.c	6810	1
rep1.c	3259	1
s.c	6081	1
sc.c	5652	1
sc1.c	6677	1
scan.c	7946	1
scanner.c	5918	24
scanner.h	5794	7
shiki.c	5263	5
spc.h	2809	22
sscan.c	7944	9
sss.c	7938	1
ssscan.c	7935	4
ssss.c	39	2
ssst.c	8865	1
sst.c	3131	1
st.c	8865	40
st.h	6076	4
symbol.c	3495	11
symbol.h	3495	5
symbol2.c	1397	1
type.c	4115	13
type.h	4112	4
typeexp.h	1390	1
合計	204218	311

data2

ファイル名	最終ファイルサイズ	バージョン数
code.c	10295	2
code2.c	9710	2
code3.c	9644	2
count.c	495	2
decl.c	8388	13
expcode.c	9286	2
expcode2.c	4352	2
expres.c	8350	18
prog.c	4853	19
prog2.c	5321	1
program.c	5207	1
scan.c	8587	19
scanner.c	4452	1
spcdef.h	3890	8
spclib.c	2842	9
spcmain.c	510	10
sscdef.h	1216	6
ssclib.c	629	5
sscmain.c	307	7
st.c	7498	18
合計	105832	147

data3

ファイル名	最終ファイルサイズ	バージョン数
block.c	9951	20
const.c	823	2
dallo.c	1066	19
error.c	162	25
exp.c	20321	19
gene.c	1095	23
label.c	141	7
machi.c	1136	5
main.c	1017	25
queue.c	686	10
sc1.c	923	3
sc8+.c	5113	1
sc8.c	3770	1
scan.c	7981	24
spc.h	2736	11
st.c	7185	19
symbol.c	3432	15
type.c	2341	19
合計	69879	247

参考文献

[1] Wayne A. Babich. *Software Configuration Management.* Addison-Wesley, Reading, Massachusetts, 1986.

[2] S. Carson and S. Setia. Optimal write batch size in log-structured file systems. *Computing Systems*, Vol. 7, No. 2, pp. 263–281, Spring 1994.

[3] Reidar Conradi and Bernard Westfechtel. Version models for software configuration management. *ACM Computing Surveys*, Vol. 30, No. 2, pp. 232–280, June 1998.

[4] Hartmut Ehrig, Werner Fey, Horst Hansen, Michael Löwe, and Dean Jacobs. Algebraic software development concepts for module and configuration families. In *Proceedings of the 9th Conference on Foundation of Software Technology and Theoretical Computer Science*, Vol. LNCS 405, pp. 181–192, Bangalore, India, December 1989. Springer-Verlag.

[5] Jacky Estublier and Rubby Casallas. The Adele configuration manager. In Walter Tichy, editor, *Configuration Management*, pp. 99–133. John Wiley and Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, 1994.

[6] Peter H. Feiler. Configuration management models in commercial environments. Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, March 1991.

[7] David K. Gifford, Roger M. Needham, and Michail D. Schroeder. The Cedar file system. *Communications of the ACM*, Vol. 31, No. 3, pp. 288–298, March 1988.

[8] J.S. Heidemann and G.J. Popek. File-system development with stackable layers. *ACM Transactions on Computer Systems*, Vol. 12, No. 1, pp. 58–89, February 1994.

[9] Jordan K. Hubbard. RELEASE NOTES FreeBSD Release 3.0-RELEASE. This document is available on the World-Wide Web at the URL "http://www.freebsd.org/releases/3.0R/notes.html".

[10] A. John. Dynamic vnodes – design and implementation. In *Proceedings of the USENIX 1995 Technical Conference*, pp. 11–23, New Orleans, LA, USA, January 16–20 1995.

[11] J.J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, Vol. 10, No. 1, pp. 3–25, February 1992.

[12] David G. Korn and Eduardo Krell. A new dimension for the Unix file system. *Software–Practice and Experience*, Vol. 20, No. S1, pp. 19–34, June 1990.

[13] S.J. Leffler, M.K. McKusick, M.J. karels, and J.S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, 1989.

[14] Makoto Matsushita and Makoto Oshita and Hajimu Iida and Katsuro Inoue. Conceptual issues of object-centered process model. In *Asia-Pacific Software Engineering Conference*, pp. 519–520. IEEE Computer Society Press, December 1997.

[15] 松下誠, 飯田元, 井上克郎. オブジェクトモデルを用いたソフトウェアプロセスの表現方法. 電気学会論文誌 C, Vol. 118-C, No. 12, pp. 1786–1791, December 1998.

[16] Makoto Matsushita, Hajimu Iida, and Katsuro Inoue. Web-based Process Management System with Object-Centered Process Modeling. In *The International Symposium on Internet Technology*, pp. 92–97, April 1998.

[17] M.K. McKusick, K. Bostic, M.J. karels, and J.S. Quarterman. *The Design and Implementation of the 4.4BSD UNIX Operating System*. Addison-Wesley, 1996.

[18] M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, Vol. 2, No. 3, pp. 181–197, August 1984.

[19] J.-S. Pendry and M.K. McKusick. Union mounts in 4.4BSD-Lite. In *Proceedings of the USENIX 1995 Technical Conference*, pp. 25–33, New Orleans, LA, USA, January 16–20 1995.

[20] Marc J. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4, pp. 364–370, December 1975.

[21] Walter F. Tichy. RCS – a system for version control. *Software–Practice and Experience*, Vol. 15, No. 7, pp. 637–654, July 1985.

[22] 山本哲男, 松下誠, 井上克郎. Monoprocessを用いた開発作業連絡支援システムの提案. 情報処理学会研究報告 98-SE-120, Vol. 98, No. 64, pp. 11–18, July 1998.