An Adaptive Version-Controlled File System

Makoto Matsushita[†]
matusita@ics.es.osaka-u.ac.jp

Tetsuo Yamamoto[†]
t-yamamt@ics.es.osaka-u.ac.jp

Katsuro Inoue[†‡]
inoue@ics.es.osaka-u.ac.jp

[†]Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
[‡]Graduate School of Information Science,
Nara Institute of Science and Technology,
8916-5, Takayama, Ikoma, Nara 630-0101, Japan

ABSTRACT
There are many version management systems to identify, record, and track the changes of software products. However, most of these systems require engineers to learn how to use these tools; if there are some miss-operation, version management could be not established. Some systems employ their proprietary versioning file systems, and engineers do not need to consider how to record the changes. However, these changes are stored in special formats, so portability of the systems cannot be accomplished.

In this paper, we propose VCFS, a novel version management file system, which automatically record changes of files. We have implemented VCFS as a stackable file system, which is easily ported to other systems. We have also evaluated VCFS with file reading, file writing, compile operations, and file size of repository. As a result, VCFS provides almost the same performance as usual file systems, and also it provides the version control facility.

Keywords
Software Development Environment, Configuration Management, File System

Introduction

Researches on software configuration management have emerged to solve the difficulties to identify, organize, manage software products[2]. Managing products is required in each phase of software development. Elements of software component should be clearly identified and defined in a design phase.

In a coding phase, products and documents are identified, organized, and managed by software configuration management system. There are various research areas in software configuration management, including version control system that manages software products. Generally, a software product needs some modifications during coding phase. Each product created by a single change operation is called a "version", and version control systems treat sequences of the versions.

Many version control frameworks and systems have been already proposed. Using these systems, reliable changes of products are promised and cooperations with engineers are achieved[1, 3]. However, many version management systems implemented as a set of tools require engineers to specify when a new product is saved (check-in) as a new version and what version should be taken out (check-out). As a result, engineers should learn how to use such version management systems, and a miss-operation causes a repository collapse. Some version management systems introduce automated check-in/check-out facility; however, these systems also employ their own formats for data repository and their original version control frameworks.

In this paper, we propose a new version management system named VCFS to solve such problems. This paper is organized as follows. In Section, we describe about other version management systems which are already proposed. In Section, we explain VCFS and its implementation. In Section, we show some evaluation results of VCFS. Finally, we conclude this work and present further works in Section.

Related Work

In this section, we discuss about related works about version management systems and their problems.

RCS

RCS (Revision Control System) is a set of tools for version management, and already used in many organizations[11]. RCS treats a product as a file in a file system, and does management activity to files with RCS tools.

In RCS, saving a new version is done by making a difference between a new version and a previous version. The version difference is shown as a result of UNIX diff command. Each version is identified as a version number (sequence of numerical value, i.e. "1.2.3"). Extracting one version is also done by a RCS tool.

Since RCS is implemented as a set of tools, engineers must learn how to use these RCS tools. In addition, if an engineer misuses the tools, the registered versions may be lost.

3D Filesystem
3D Filesystem is implemented as a modified file system of UNIX System V release 3[7]. The file system employs the same version management model in RCS; a state of products are defined as a file itself, and a version is defined as a sequence of numerical value. Merging version management features to native file system makes a tool-free operation for extracting any version; however, registering a new version requires some supporting tool associated with the file system.

3D Filesystem makes up for some problems of RCS; however, the file structure is their original one, so the registered version information is only extracted via 3D Filesystem. In addition, 3D Filesystem has only its own version management model, engineers cannot use other version management models.

VMS
The file system in VMS operation system records file changes automatically. VMS saves all of the contents of newly saved file, and these saved contents are identified by a special suffix of filename. The VMS file system provides a version management facility; however, other information such as file owner names is not saved.

VCFS
In this section, we will show our new version management system VCFS.

Design Policy
At first, we specify required features of VCFS to establish better version management mechanisms as follows:

- Easy operation: engineers are not requested to learn how to use VCFS at first; typical operations should be automatically achieved by VCFS.
- Open structure: VCFS should not have proprietary structure of data repository.
- Plug-and-play system: VCFS should be independent from actual version management framework itself; the system can be easily configured also when a new version management framework is emerged.

As a result, VCFS is designed as a stackable file system [5]. A stackable file system does not change the internal structure of raw file system (UFS (Unix File System), MSDOSFS (MS-DOS File System), etc), or does not save the file contents directly to hard disk drive; it saves as a plain file of a lower file system. Operations to the target files are automatically recorded via the lower file system. VCFS allows to use external version management system; two or more version management systems can be exists simultaneously. Detailed design is shown in Section.

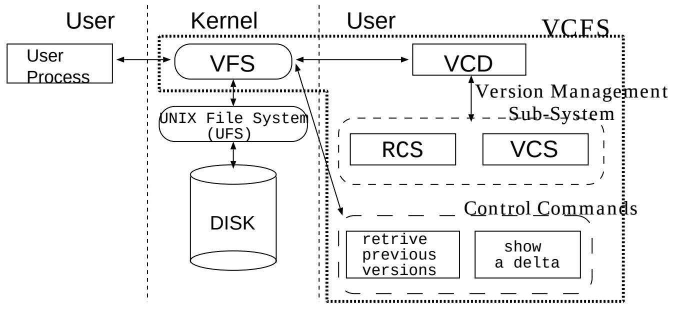

Figure 1: System overview of VCFS

Overview
In VCFS, operations to a file (read and write) are automatically mapped into activities of version management; engineers do not consider what should be done to manage the product versions. There are no difference between the operations to usual file system and VCFS file system from a viewpoint of users' processes.

VCFS manages the versions of regular files (symbolic link, special file, socket, and named-pipe are out of our scope). A new version is created iff a file is created or an existing file is changed. Checking out the latest version is done with simply reading the file. VCFS also supports a file locking mechanism. Before checkin is completed, other process can only checking out the file.

VCFS employs "checkin/checkout model"[4] which was proposed by RCS, and VCFS itself does not have its own version management mechanism (sub-system); engineers can import a favorite version management subsystem which is adaptable to the model, i.e., VCFS can use RCS commands.

System Design
VCFS is composed of file system (VFS), version control daemon (VCD), version management sub-system, and VFS control commands. Fig. 1 shows the structure of the VFS components.

The advantage of the separation of version management facility from the kernel is that switching check-in/check-out operations can be easily done. In addition, installing VCFS to a software development organization can be easily performed if RCS is already employed at the organization. VCFS also allows to work with yet another check-in/check-out style version management system.

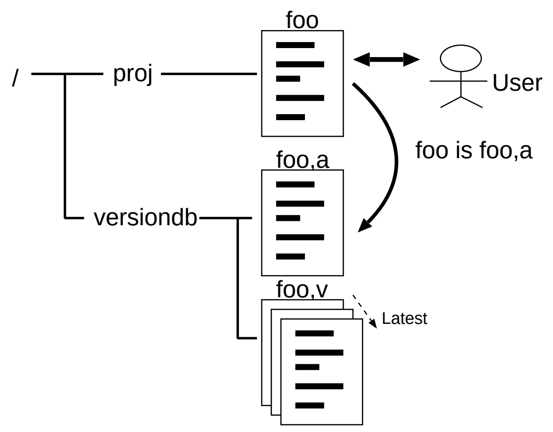

Figure 2: Mapping files between VCFS and actual file system

File System Since VCFS is implemented as a stackable file system, all files handled by VCFS are stored to a raw file system. Each file via VCFS is consists of two actual files; one for the latest version, and another for version repository (Fig. 2).

Usually VCFS shows the latest version of the file via file system. For example, we assume that /versiondb is mounted to /proj by VCFS and create a file /proj/foo. VCFS creates "/versiondb/foo,a" for the latest version of file foo (/proj/foo itself) and "/versiondb/foo,v", for the version repository. Note that /versiondb/foo,v is invisible via VCFS; it is only for version management and not for the ordinal operation.

VCFS always keeps the latest version of every file (/versiondb/foo,a in the previous example) for fast file read/write to the latest version which is modified in the most cases of software development. When a UNIX process opens a file in read-only mode, VCFS behaves as the same as NULL file system[10]. When a process opens a file in write-only or read-write mode, VCFS hooks close() system call and performs a check-in operation to a file. The check-in operation runs by the version control daemon and the daemon calls the version management sub-system which is outside of the kernel.

Version Control Daemon VCD is a daemon process which acts as a bridge between the kernel and the version management sub-system. VCD dispatches the request from the kernel to the version management sub-system.

Version Management Sub-System The version management sub-system is the actual version management part of VCFS. In general, version management sub-system consists of a set of tools. VCFS employs external version management systems as the sub-systems, and we can change the sub-systems to use. Current prototype of VCFS has two kinds of version management sub-systems, RCS and VCS.

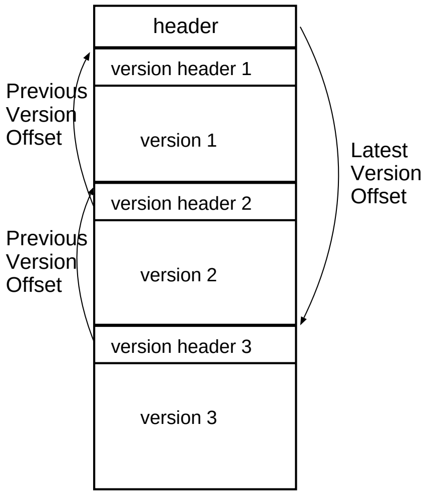

Figure 3: Definition of VCS file structure

Table 1: Definition of VCS header

Name	Size	Content
version number	4byte	latest version number
latest offset	16byte	offset of a latest version

RCS keeps only a delta of each version and the latest version. RCS allows version derivation (a tree structure of version sequence), and a derived version is called as a branch. RCS sub-system uses RCS tools to record versions.

VCS is a simple version management system as the one of VMS file system. VCS saves all versions as-is, and does not calculate the delta between versions. No version derivation is allowed, however, registering a new version is faster than the RCS sub-system.

Fig. 3, Table 1, and 2 show a file structure, header definition, and version header definition respectively. The VCS check-in tool appends the version header and version itself to a version management file (and recalculate the version number and offset of the latest version). A previous versions can be retrieved by tracing header offsets. VCS also saves file attributes (ownership, file modes); checking-out a version restores not only file contents but also file attributes.

VFS Control Commands VFS control commands help to control VFS behavior. Following commands are al-

Table 2: Definition of VCS version header

Name	Size	Content
attribute	size of vattr structure	vattr structure of a file.
previous offset	16byte	offset of a previous version.

ready available.

- Retrieve previous versions
  Users can retrieve any one previous version (not the latest version) by specifying a version number and/or its created date.
- Make a branch
  Branching is done with control commands if the version management sub-system has branching feature.
- Show a delta between versions
  Users can check the difference between versions.

Prototype of VCFS

Current prototype of VCFS runs on FreeBSD 3.0-RELEASE [6], a BSD UNIX[8, 9] variants. VCFS is written in C and about 5000 lines total; VFS in kernel for 4500 lines, VCD for 340 lines, and 300 lines for others.

Evaluation

This section discusses the prototype of VCFS from viewpoints of the system performance and storage size. We take up UFS (actually FFS (Fast File System), a usual UNIX File System) and NULLFS (loopback file system, implemented as a stackable file system) to compare with our VCFS. Note that all tests are made at 166MHz Pentium PC having 48MB RAM and FreeBSD 3.0-RELEASE.

Performance Evaluation

Read Files At first, we measured an elapsed time for a UNIX process to read distinct new files of 1M bytes size repeatedly. "An elapsed time" means time between process initiation and process termination; be aware that an elapsed time includes an overhead of typical UNIX processes (process initialization, etc).

Fig. 4 shows the results of the reading test of VCFS, UFS, and NULLFS. "VCFS(RCS)" means VCFS using RCS as the version management sub-system. The vertical axis shows an elapsed time, and the horizontal axis shows the number of file reading.

The time for VCFS(RCS) is almost the same as one of NULLFS. With regard to file reading, it is understood that it takes almost no extra time as for the version control functions in the file system. VCFS(RCS) and NULLFS are slower than UFS, since there is an implementation overhead of a stackable file system.

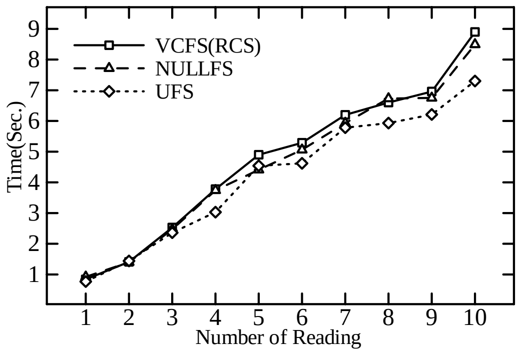

Figure 4: Read 1M bytes files

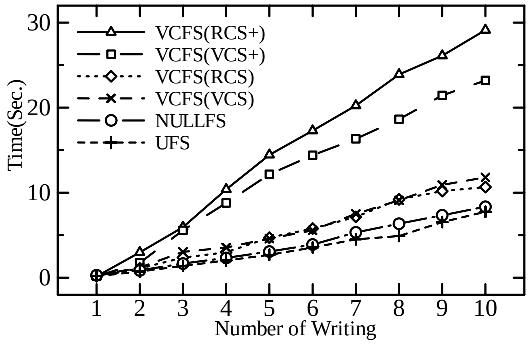

Figure 5: Write 1M bytes files

Write Files We also measured an elapsed time to write files, similar to file reading test described before. Fig. 5 shows the results of writing test. "VCFS(VCS)" means VCFS using VCS as a version management sub-system. "VCFS(VCS+)" and "VCFS(RCS+)" includes a time of synchronization between VCD and sub-system. In general, VCD should not wait the termination of sub-system as usual; however we measured these for comparison purpose.

NULLFS is about 10% slower than UFS, and it is the same result of reading test. Writing a file in VCFS requires a new version registration which is not required by the reading, so VCFS consumes 30% or more time compared with UFS. Actual time for a new version registration is twice as of UFS, comparing VCFS(RCS+) / VCFS(VCS+) with UFS. Actually, VCD and sub-system work asynchronously so VCFS is only 30% slower than UFS, and we assume that it is reasonable.

Fig. 6 shows the same writing test, using a 32K bytes file instead of a 1M bytes file. The same tendency is shown even if a file size is smaller than the previous test.

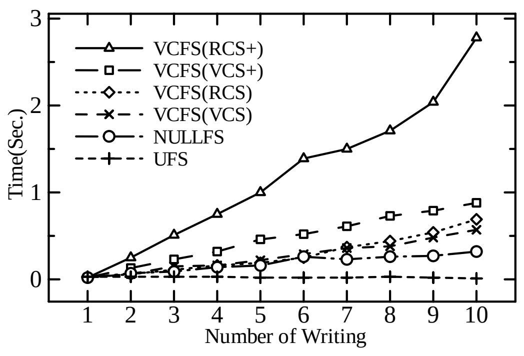

Figure 6: Write 32K bytes bytes files

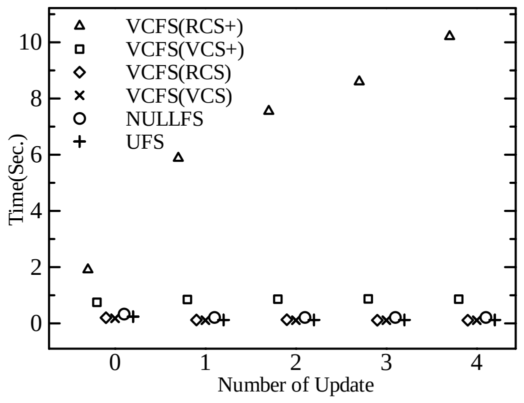

Figure 7: Update 1M bytes files

Update Files This test mixes read/write operations to a single file of 1M bytes size. When a process write to a file, the updated file contents are completely different of the old one (no same lines), to assure this test is the worst case for RCS. Fig. 7 shows the result of this test. Note that "0th update" means the first attempt to write a file (create a file).

Total time of updating a file is the same of creating a file, except the case of VCFS(RCS); there is a slight difference between 0th update and 1st or later update. Since VCFS(VCS) does not calculate a delta between versions, it takes a fixed time (even if the delta is huge or not) to update a file if the file size is the same. In contrast to VCFS(VCS), the more file is updated, the more time is needed in VCFS(RCS).

Compile an Application Finally, we measured a typical software development procedure, by compiling an application. We pick up two applications, "tar" and "dump" bundled with FreeBSD. Table 3 shows the result of an elapsed time for executing "make" programs of those applications.

Table 3: Compile an Application (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

Table 4: A sample data

	All LOC	Files	Versions	Compile times
data1	9339	45	311	222
data2	4067	20	147	92
data3	2543	18	247	110

The procedure of compiling an application does read source codes and write object codes repeatedly. As a results, VCFS is about 20% slower than UFS; we think that this overhead is not serious problems for practical software development environments.

File Size Evaluation

We applied a sample data taken from a programming seminar of Osaka University (Table 4), and measures the total file size saved into a filesystem. The test creates/updates files, and then compile them. Table 3 shows the result.

The result of UFS shows the size of final product version. The result of VCFS shows that more file size is required to save the whole data, VCFS(RCS) requires several times as much as UFS, and VCFS(VCS) requires ten times as much as UFS. The file size may vary; it depends on what sub-system is used with VCFS.

Discussion

Totally, VCFS is about 20% slower than UFS; however, we consider there is acceptable for practical use. Using VCFS with RCS sub-system, disk usage is reasonably small.

Following is the summary of VCFS and other version management system.

- RCS and VCFS(RCS)

Table 5: Total file size (K bytes)

	UFS	VCFS(RCS)	VCFS(VCS)
data1	225	1388	3149
data2	117	546	1377
data3	73	604	1501

RCS consists of a set of tools, and VCFS(RCS) is a file system wrapper for RCS. In general, both systems have almost the same functionality; however, VCFS does not force to engineers to learn how to use RCS tools.

- 3D Filesystem and VCFS(RCS)
  Both systems are implemented as a file system and provide a version management facility. VCFS(RCS) employs existing UFS file system and existing RCS data structure; however, 3D Filesystem is their own data structure, and requires check-in procedure explicitly.

Conclusion

In this paper, we show the problems of existing version management system. We also propose a new version management system VCFS; VCFS acts like a file system wrapper for existing version management system. We also evaluate VCFS in performance and file-size point of view. VCFS provides easy operation of version management, open system structure, and plug-and-play capability of selecting external version management system.

As a further work, version management of directory or special files are planned. Also, supporting distributed software development environment and more usability evaluation are required to VCFS. We are now working on Web-based VCFS tools which overrides VFS control commands, to support graphical and understandable representation of version history.

REFERENCES

1. Babich, W. A.: *Software Configuration Management*, Addison-Wesley, Reading, Massachusetts (1986).
2. Conradi, R. and Westfechtel, B.: Version Models for Software Configuration Management, *ACM Computing Surveys*, Vol. 30, No. 2, pp. 232–280 (1998).
3. Estublier, J. and Casallas, R.: The Adele Configuration Manager, *Configuration Management* (Tichy, W.(ed.)), John Wiley and Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, pp. 99–133 (1994).
4. Feiler, P. H.: Configuration Management Models in Commercial Environments, Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213 (1991).
5. Heidemann, J. and Popek, G.: File-System Development with Stackable Layers, *ACM Transactions on Computer Systems*, Vol. 12, No. 1, pp. 58–89 (1994).
6. Hubbard, J. K.: 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".
7. Korn, D. G. and Krell, E.: A New Dimension for the Unix File System, *Software–Practice and Experience*, Vol. 20, No. S1, pp. 19–34 (1990).
8. Leffler, S., McKusick, M., karels, M. and Quarterman, J.: *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley (1989).
9. McKusick, M., Bostic, K., karels, M. and Quarterman, J.: *The Design and Implementation of the 4.4BSD UNIX Operating System*, Addison-Wesley (1996).
10. Pendry, J.-S. and McKusick, M.: Union Mounts in 4.4BSD-Lite, *Proceedings of the USENIX 1995 Technical Conference*, New Orleans, LA, USA, pp. 25–33 (1995).
11. Tichy, W. F.: RCS – A System for Version Control, *Software–Practice and Experience*, Vol. 15, No. 7, pp. 637–654 (1985).