GitOD: an On Demand Distributed File System Approach to Version Control

Jonatan Schroeder
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada
jonatan@cs.ubc.ca

Dissertation Advisor: Prof. Son T. Vuong, PhD
Dissertation Comittee: Prof. Alan S. Wagner, PhD; Prof. Michael J. Feeley, PhD

DOCTORAL DISSERTATION COLLOQUIUM

EXTENDED ABSTRACT

***Abstract*—Version Control Systems (VCS) are tools used to manage changes in documents, programs and other kinds of information stored as computer files, usually as part of a collaborative project or task. In this project, an extension is proposed to the concept of distributed VCS through the development of a specialized distributed file system. In order to reduce the use of bandwidth and disk space usually required by a local copy of a distributed VCS repository, a new approach is proposed in which only essential files are initially retrieved, while other files are downloaded as they are needed. This strategy can be useful in devices where bandwidth, storage and power resources are limited and/or expensive, such as tablets or smartphones.**

***Keywords*—Collaboration Technologies; Distributed Systems; Mobile and Wireless Collaboration Systems; Version Control.**

I. Introduction

Version Control Systems (VCS) are tools used to manage changes in documents, programs and other kinds of information stored as computer files, usually as part of a collaborative project or task. These systems are an important collaboration tool in many projects involving multiple contributors. A typical VCS platform allows multiple contributors to work in the same project simultaneously, while maintaining the history of all involved files. Contributors also have the ability to synchronize and merge their local copy of the project with other contributors, based on changes performed asynchronously by each one of them. A typical VCS is able to maintain snapshots of the project and its related files, provide tools to view and restore previous versions of the project, and maintain separate branches of project development [1].

This work was supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

Available VCS platforms may be classified as either centralized or distributed. In a centralized (or client-server) VCS approach, a central repository (server) is used to maintain the revision history, and all contributors (clients) connect to this repository to perform revision control tasks such as synchronization and history examination. By contrast, in a distributed (or decentralized) VCS approach, every contributor keeps a local copy of the entire repository, and synchronization is performed between peer repositories. In this approach, common operations such as creating a project snapshot (commonly known as a commit), viewing history, and reverting changes are performed locally. No central repository is necessary, although one may be used if deemed necessary [1], [2].

In this project, an extension is proposed to the concept of distributed VCS through the development of a specialized distributed file system. In order to reduce the use of bandwidth and disk space usually required by a local copy of a distributed VCS repository, a new approach is proposed in which only essential files are initially retrieved, while other files are downloaded as they are needed. This strategy can be useful in devices where bandwidth, storage and power resources are limited and/or expensive, such as tablets or smartphones. The proposed approach also intends to address a known size limitation of distributed VCS platforms. Popular distributed VCS platforms, such as Git, Mercurial and Bazaar, discourage the use of very large repositories, usually recommending that users should divide them into smaller submodules [3]–[6]. Using an approach in which files are only downloaded when needed, this limitation can be mitigated, as the entire repository no longer needs to be transferred and/or stored in each peer.

978-1-4673-1382-7/12/$31.00 ©2012 IEEE

Distributed file systems are modules that provide a user with access to a group of files as if they were normal files, but that are actually stored in a network. Several strategies have been used to store these files, focusing on aspects such as security, fault tolerance, dependability and performance. Files may be stored in a single server or multiple servers, or even in peer clients [7]. Examples of distributed file systems for general purpose include Coda and MooseFS [8], [9].

Some distributed file systems, such as Ceph and HDFS (Hadoop Distributed File System), have implemented snapshot support, which allows a user to retrieve previous versions of a file or directory [10], [11]. This feature, however, is not sufficient for the proposed problem. A version control system provides means for separate development of a set of files by different contributors, and coordinated merging once all contributors provide their set of changes. Also, in a distributed version control system there may be several maintained versions of a project, and each version must be changed independently.

There are some efforts towards building a file system that provides access to a version control repository through virtual access to the repository files. For example, the Cascade File System is a proprietary system that allows a user to access the files in a Subversion or Preforce repository without the need to checkout files [12]. Similarly, cvsFS maps a CVS repository to a file system [13]. These systems, however, implement this access to centralized VCS platforms, which present a different set of challenges and aspects than distributed version control.

Kapitza *et al* presents a replicated VCS based on Git in [14]. This system, called DiGit, uses fault-tolerant object replication for increasing the availability of a central repository. This approach, though, is focused on the server dependability, and does not address local use of bandwidth and storage resources.

II. Proposed Solution and Methodology

The implementation proposed in this project will be implemented as an extension to Git, one of the most popular open source VCS platforms [3]. Through this extension, called GitOD (Git On Demand), a Git repository will be saved locally, but accessed by the user through a virtual distributed file system, based on a FUSE implementation. Only files that have already been requested by some operation will be saved locally, and additional files would be requested from other peers as they become necessary. These requests will be transparent to the user, as they will be accessed through the virtual file system.

Since all the interaction between the user and the working directory of the local repository will use a virtual file system, the proposed implementation will be able to identify when specific files are required, and will be able to produce them as necessary. In this approach, when a repository is cloned, only essential files such as the root directory listing, cloned commit information and snapshot tags are obtained. Once the user starts to use the system and request new files, these files are obtained from other repositories and made available locally for future requests.

As a significant part of the effort of extending the concept of distributed VCS, a new distributed file system will be developed, focused specifically on the characteristics of a Git object repository. This file system will take advantages of the immutability of these objects to provide improvements in caching methodologies (e.g., a specific file in one specific project snapshot will never be changed, since a difference in the file will correspond to a different snapshot). Also, since most users will use only the most recent versions or snapshots of the project, and don't often need very old snapshots, the file system can be adapted to give higher priority and better availability to latest snapshots, providing a better experience to most users.

This approach is intended to be a trade-off between speed and resource utilization. While some operations may become temporarily slower because files are not available in the local repository, bandwidth and local storage resources are potentially saved. However, once a file is downloaded, it is available for future use as well. Based on this approach, a user that requests a specific subset of a large repository may have an initial delay in usual operations, but once files are available locally the operations become local, and thus can be accessed faster.

III. Expected Contributions

One of the main research contributions in this project is the study and development of specific strategies for distributed file systems applied to a version control environment. Specific characteristics of VCS platforms can provide means for better caching strategies and prediction of files that may be required by the user in the near future.

Objects (such as files, directories and snapshots) in a Git repository database are immutable, and changes to the files in a project correspond to new objects associated to a new snapshot. This aspect, allied with some security features already available on Git, allows local repositories to trust that their content is the most recent version of an object, since there is only one version of it.

Since objects are connected through a known structure of snapshots and files, it is possible to predict objects that might be needed in the near future based on objects that have been recently retrieved. This prediction could allow the file system to proactively download files that have a high probability of being requested by the user.

One specific point that distinguishes this file system from other generic file systems is the fact that it should be able to communicate and obtain objects not only from other repositories in which GitOD is installed, but also with "normal" Git repositories. This allows a user to have the benefits of a distributed file system approach even if peer collaborators and servers in the same project do not use GitOD.

Although this approach takes advantage of some of the characteristics of centralized VCS platforms, such as remote operations, the communication with a peer repository is not restricted to a single central server. Each local repository can maintain a local pool of peers, and can connect to them using several strategies, still to be studied. There is no single point of failure, as in centralized systems.

The proposed approach is not intended to replace the original Git implementation in all of the repositories. There are situations in which a complete object database is desirable. Examples of such include main authoritative servers, used as basis for official versions of the project; new projects that correspond to forks (clones) of existing projects, and that are to be developed without intention of further merging with the original project; and small projects with very few contributors in a single local network.

IV. Current Status of Research

A final version of the research proposal is in preparation, after which the development and evaluation of the proposed research will begin.

An initial step towards the implementation of GitOD is implementing the file system that provides the strategy described in this project. After this implementation is complete, some changes will be implemented in the Git core operations, focusing on avoiding access to files that are not available locally and not strictly needed. Some of these changes might involve remote requests to peers that have all required files, by requesting that the operations be performed at these peers, but this implementation is still under study. This implementation is expected to be complete by December 2012.

Based on a correct implementation of GitOD for basic scenarios, further study will be applied towards additional features, such as an Android application to be used as proof of concept, that would allow GitOD to be used in smartphones and tablets. Additionally, changes in garbage collection behaviour are under study, in which files not used for some time could be removed to save storage space, provided that these files could be retrieved from other peers if needed.

After tests involving correctness, completeness and performance evaluation, results would be gather that would allow for refinement in the algorithms and implementation. With this results, a final version of the PhD thesis is expected to be complete by June 2013.

Biography

JONATAN SCHROEDER is a PhD student at the Department of Computer Science at the University of British Columbia (UBC) in Vancouver, Canada. He obtained both his BSc and MSc in Computer Science at the Universidade Federal do Paraná (UFPR), in Curitiba, Brazil. His research involves the application of distributed file systems in version control systems, as well as projects related to distance learning tools. Other research interests include fault tolerance in network routing and connectivity measurement tools. In addition to his research, he is also an academic assistant involved in improving the learning methodology of network programming courses, funded by the Carl Wieman Science Education Initiative (CWSEI).

References

[1] B. O'Sullivan, "Making Sense of Revision-control Systems," *ACM Queue*, vol. 7, no. 7, Aug. 2009.

[2] D. A. Wheeler, "Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Management (SCM) / Revision-Control Systems," 2004. [Online]. Available: http://www.dwheeler.com/essays/scm.html

[3] S. Chacon, "Git - Fast Version Control System," [accessed August 25, 2011]. [Online]. Available: http://git-scm.com/

[4] L. Torvalds, "Google Tech Talk: Git," 2007, [accessed August 25, 2011]. [Online]. Available: http://www.youtube.com/watch?v=4XpnKHJAok8

[5] ——, "Why Git is so fast," 2009, [accessed August 25, 2011]. [Online]. Available: http://osdir.com/ml/git/2009-05/msg00051.html

[6] S. Chacon *et al.*, "Git Community Book." [Online]. Available: http://book.git-scm.com/

[7] G. Silberschatz, *Operating System concepts*. Addison-Wesley Publishing Company, 1994.

[8] P. J. Braam, "The Coda Distributed File System," *Linux Journal*, no. 50, Jun. 1998.

[9] M. Satyanarayan, J. J. Kistler, and E. H. Siegel, "Coda: A resilient distributed file system," in *IEEE Workshop on Workstation Operating systems*, Cambridge, MA, Nov. 1987.

[10] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[11] K. Shvachko, K. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, May 2010, pp. 1–10.

[12] Conifer Systems LLC, "Cascade: Overview," 2010, [accessed Feb. 6, 2012]. [Online]. Available: http://www.conifersystems.com/cascade/overview/

[13] "cvsFS - mount a CVS-Tree," 2009, [accessed Feb. 6, 2012]. [Online]. Available: http://cvsfs.sourceforge.net/

[14] R. Kapitza, P. Baumann, and H. P. Reiser, "Using object replication for building a dependable version control system," in *Proceedings of the 8th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, ser. DAIS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 86–99.