"GIT TORRENT"- DVCS FOR LARGE REPOSITORIES AND CONTEXT SPECIFIC REVISIONING

Vipul Amler
Department of Computer Science
Pune Institute of Computer Technology
Pune, India.
vipulnsward@gmail.com

Prajakta Tambe
Department of Computer Science
Pune Institute of Computer Technology
,Pune, India.

Neeraj Thakur
Department of Computer Science
Pune Institute of Computer Technology
Pune, India.

Nikhil Kumbhar
Department of Computer Science
Pune Institute of Computer Technology
Pune, India.

Abstract

Git is a decentralized , fast Version Control System. Git Torrent Protocol tries to overcome the scalability issues of Git, by providing a Torrent Overlay, removing the haul from a Centralized System. GTP and Git do not harness the context of large and binary files like Multi-Media ones. Providing context specific re-visioning may induce large scalability to the existing architecture. It would also harness the torrent in a more granular and efficient way.

Keywords: Distributed Computing, Context-mapping, File-Revisioning, Information Retrieval Systems, Torrent, Git

I. INTRODUCTION

GTP(Git Torrent Protocol)[1] tries to address the scalability issues of Git by providing a peer-to-peer, Distributed Version Control System(DVCS). It creates an overlay for the Git system to torrent the repository, and its relevant data. This has limitations in Real Time repository[3][4] sharing due to the sheer amount of data transfer involved and the frequency of changes in repositories. Harnessing the context of file, and granulating the revisioning within the file-structure, may induce more scalability and meaningful repository sharing.

This paper tries to discuss

1. Context Specific Diff and refining revisioning to granularize it to File-Structure Level.
2. Enhance GTP using Context Specific Revisioning.
3. Generic revisioning architecture for large self-contained files.

II. GIT-TORRENT

A. Bit-Torrent

Bit Torrent Protocol[2] is a method to share data in a peer-to-peer fashion enhancing the share operation by removing bandwidth and storage limitations from a single server.

A **'swarm'** is a network of connected machines '**torrenting**', data. A **'seed'** is a any machine in the swarm that has the whole data and provides **'pieces'** of this data to **'leeches'** which are machines that need parts of data from the complete file.

A '**tracker**' tracks status or all peers in the swarm.

For sharing a file, a user needs to create a torrent descriptor. This acts as an entry point in sharing the file and initial seed(s) . After getting a complete file, a 'leech' adds to the list of 'seeds' and starts giving other 'leeches' pieces of data.

As the number of seeders grow the operation of sharing scales to a large extent. This is particularly efficient for large files. Efficient ways to share in local neighboring network enhances the share operation even more.

B. Git

Git[5][6] is a fast Version Control System[5]. It has many unique features making every Git repository

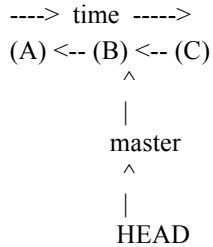

Fig. 1 A repository with three commits where (A), (B), and (C) are the first, second, and third commits, respectively.

independent, with all git data stored in entirety locally along with the repository. This makes it completely decentralized. Various operations are performed in revisioning which make it faster to manage repositories and relevant data.

A Version Control System, tracks revisions in the life time of development of a project. Commit Objects are additions or changes to the existing data. These store references to parent as well.

A head traces a commit object and associated version of change. Tag are special references from amongst the existing ones to signify special events viz. Releases, candidates for release,etc. These act as namespaces for the commit objects.

HEAD contains the reference to current working repository.

Git[5] allows various operations on repository. Branching and merging happen in a very fast manner.

Some of the operations[5] on the repository include -

1. Cloning- Creating a copy of an exiting repository.
2. Branching- Creating a new line of work, for development, on the existing repository, which is independent from the existing one.
3. Merging- Merging Branches of the repositories previously created.
4. Sharing[6] repositories over git, rsync, ssh protocols and many more.

C. Git-Torrent Protocol

Git Torrent Protocol[1] harnesses the power of fast repository actions of Git and the distributed nature of it to the likes of Bit-Torrent[2], which works on sharing this repository data in a peer-to-peer way.

It proposes two protocols to handle the Tracker and Peer transactions and the other to handle the actual data transfer amongst the peers.

All communications are done, using Ben-coding mechanisms. To start the sharing of repository a git-torrent descriptor is created associated with it. Later communications are handled by the two protocols as below.

1. Tracker HTTP Protocol
2. Peer Wire Protocol

C.1 Associated Data Objects

Mapping of repositories to torrent is done by the use of various object structures. The primary object of concern is the Commit Object which gives the state of the project at a particular time.

In Git-Torrent, following Data Objects are proposed for use-

1. Pieces- These are parts of files, mapped to the commit object for sharing the repositories. This is the primary source for sharing the actual data involved.
2. Packs- When the size of pieces a small or to create efficient transfer of large data, small pieces may be bound together to create a 'pack'
3. Index- These act as indices within the file.

All commit objects in the repositories are identified\ in a unique way by creating SHA-1 Hashes based on the content the have.

C.2 The Protocols

1. Tracker HTTP Protocol

This[1] defines mechanisms to setup initially the sharing system for a repository. It performs the communications amongst the tracker and the seeder/peer.

The tracker sends messages to receive updates of the data held by a peer. It requests peer to send updates about the same in regular intervals. The peer sends updated information about its status and data it has.

The most important operation while doing this is generating and updating the list of all seeders in the swarm for distribution amongst all the leeches to use.

2. Peer Wire Protocol

This[1] specifies the working of actual data transfer amongst the peers.

A peer may exist in various states viz .

'interested' in data ; 'uninterested' in data

'choked' from sharing; 'un-choked' ,etc

These define the nature of communication taking place in the data transfer between the peers.

Messages are sent to update the state and status of data received and the acknowledgment of the latter for next pieces to be received.

Three message are involved over here. The most important is that of the specification of request of the actual data required to complete the repository at the destination. This message specifies the index of the data, the pack it belongs, and SHA-1[5] id, the actual data contents and other relevant information.

Other messages include canceling and masking of clients.

D. Extending Storage Architecture

The breaking down of commit objects[3] and its mapping with torrent is an important aspect while the sharing of repository takes place.

The Commit objects define the state of a repository by providing deltas, which are changes from the previous version of the file. Diff[7] is used as a mechanism to generate the changes between the two specified versions. Diff[7] doesn't work in a context specific way. As such the diffing[9] of large ,binary files, as in the case of multimedia files, is stored in entirety or is costly. This means the diffing[9] provides no advantage of deltas, and redundant data is reproduced as it appears to be in-aligned and can't be recognized by the diff.

This poses serious problems when considering files which happen to be mapped as purely binary and the data diffs[9] are remapping whole file, and minor changes appear to be changing the whole file, creating an illusion to store the file in entirety. Some utilities, have tried to address this, but the solution do not work in a context specific way.

An alternative approach[9] tries to work around the tracking of changes in files considering the inner structures of the file. Consider the scenario of an animation file built by blender. This is a .blend file, consisting of various data structures and inner references, within the file. Fig 3. points out various inner data structures involved to manage the different aspects of an animation file.

Generally if, the normal diff was used the revision would not consider the context of these. Taking an advantage of this structure, we can provide a mechanism to identify the context. The diffing [Fig 2.] now would consist of following procedure-

1. Identify the context
2. Create a reference of the structures involved
3. Identify the absence or presence of data in these
4. Diff over this data, individually

The diff would thus be granularized, from that of a file to the structures involved within it. Linked in resources would be needed to provide special considerations for separate revisioning or the lack thereof. A special meta-info file may track these, and data associated with them.

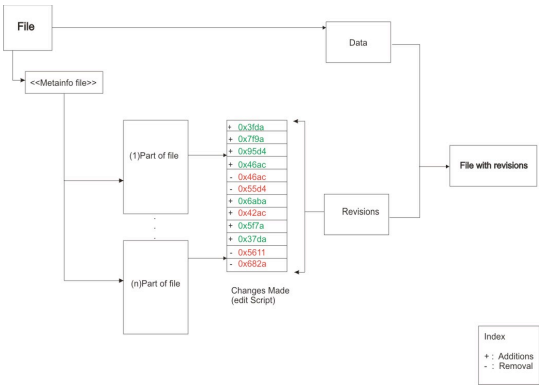

Fig 2. Diff-Operation in Context specific way

E. Mapping to GTP

Now the data diffing extracted from the previous case would then be banked upon and mapped to GTP data objects of pieces and relevant information for re-construction for the same, would be embedded.

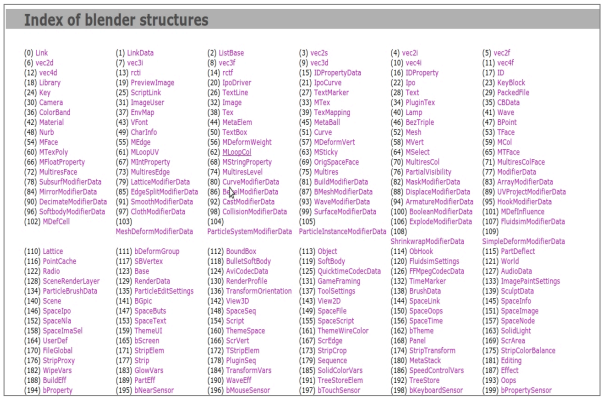

Fig 3. Blender Structures.

There a many folds of advantages[3][4] involved in doing this.

1. The File mapping makes the file reduction from sizes of more than 100 GB to MB.
2. This mapping is essential to share or torrent only the data asked for by the leech.
3. Reduction in the amount of information shared takes place, as only specify changes in the part of file may be addressed and asked by the leech instead of the whole changed version of the file.
4. Frequent changes in same file over different context may not affect the sharing of a different part of the same file.

F. Choosing the seeder

An important bottle neck in real time is tracking down, which peer actually has the data asked for by a leech which addresses a specific revision in time.

Every peer has their own version and changes over the repository. Syncing of these and providing an efficient mechanism for identifying the seeders which too happen to be dynamic is essential.

A central data store cannot act a source of the tree, of all changes in the repository. The mechanism of sharing fails if not many peers have the intended commit or tree requested by the leech.

Yet the file mapping mechanism reduces the intensity of this problem.

G. Associated areas

Deb-Delta and Deb-Torrent are major applications which try to harness the torrent functionality and context specific diff. Deb-Torrent aims at torrenting the deb packages from the Debian Operating System. Deb-Delta tries to achieve context specific diffing of the Debian packages. Visual-Diff[8] would be able to harness the context specific diff, and provide a fast way of seeing changes ongoing in different versions in multimedia files in graphical fashion.

CONCLUSION

Using a context specific mapping of files in the repository and torrent functions of GTP, increases the scalability of Git, many folds. It harnesses the context of a file to granularize the change-set generation. This in-turn , helps to lower the data torrented or requested and increases syncing of repositories in a peer-to-peer fashion in a faster way.

REFERENCES

[1] Jonas Fonseca, "GitTorrent: a P2P-based Storage Backend for Git", University of Copenhagen, November 2006.

[2] Lilja Fjeldsted, Jonas Fonseca & Basim Reza, "Specification and Implementation of the BitTorrent Protocol", University of Copenhagen, July 2005.

[3] L. Cherkasova and J. Lee, “FastReplica: Efficient Large File Distribution within Content Delivery Networks (USITS 2003)”, In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, March 2003.

[4] P. Rodriguez and E. W. Biersack, “Dynamic Parallel-Access to Replicated Content in the Internet”, IEEE/ACM Transactions on Networking, 2002.

[5] Scott Chacon, Git Internals ,http://peepcode.com/products/git-internals-pdf, September 2011.

[6] Scott Chacon , Pro-Git , http://progit.org/ , September 2011

[7] Eugene W. Myers, An O(ND) Difference Algorithm and Its Variations,Department of Computer Science , University of Arizona, 1986

[8] Hector Yee, Sumanta Patanaik and Donald P. Greenberg , Program of Computer Graphics , Cornell University, ACM TOG 2004

[9] Colin Percival, Naive differences of executable code, http://www.daemonology.net/bsdiff, 2003