Pinnacle: A Simple Peer-to-Peer Text Editor that Minimizes User Input

Alexander Chernyakhovsky, Alan Wagner, Luis Lafer-Sousa

1 Introduction

Pinnacle is a simple peer-to-peer text editor that enables a dynamic group of users to collaborate on a document in both online and offline settings with minimal intervention. The design is intended to automatically track document history using git and merge changes by using the principles of operational transformation (OT) and a modified version of the reconciliation algorithm used by git. These algorithms will flag any conflicts, which will then be presented to the users for manual resolution. Additionally, Pinnacle supports fault tolerant commits by specifying a named git commit and using Pretty Good Privacy (PGP) signatures to verify mutual agreement.

1.1 Design Overview

The history of each Pinnacle document is stored in a git repository. Rather than having a single commit tree represent the entire file, however, Pinnacle ensures that there is a separate commit tree for each paragraph. Furthermore, a "table of contents" file exists in the repository to track the relative ordering of paragraphs. In addition to this, each Pinnacle client maintains an OT log for each paragraph file and for the table of contents. These logs contain a vector clock with all of the add and delete operations performed locally since the last successful system commit point.

Pinnacle uses OT, the core technique behind the collaboration features in Google Docs, to merge changes. At regular intervals, a client will scan each collaborator's OT logs, transform and perform all operations in a consistent order, and commit the resulting document to the git repository. Merges executed using OT are guaranteed to be free of conflicts, much like Google Docs, but OT may fail under extreme circumstances. In such a situation, Pinnacle uses git to roll the document back to a previous consistent version. Then, a variant of git's reconciliation algorithm is employed to merge the most recent changes and flag conflicts for user resolution.

In order to allow for connectivity between nearby computers when an Internet connection is not available, Pinnacle implements a link layer. Given the IP addresses of all collaborators, it interfaces with the available hardware to establish a connection.

Pinnacle implements an application layer to interface between the users and the remainder of the system. It is responsible for providing a graphical user interface for document editing and consistent conflict resolution. Furthermore, it is in charge of defining how the document is divided into paragraphs, maintaining all OT log files, and determining when an OT merge is necessary. Additionally, when Pinnacle is operating offline, the application layer regularly asks the link layer to attempt to connect any collaborators that may be nearby.

1.2 Tradeoffs and Design Decisions

Minimization of conflict frequency is Pinnacle's central design consideration. This is based upon the assumption that users prefer a text editor that asks for as little user intervention as possible, such as Google Docs. Fortunately, OT does not ever produce any conflicts. User intervention is only necessary in the infrequent case when OT fails. Under these circumstances, Pinnacle further reduces the conflict rate by employing a variant of the git merge algorithm that reconciles files on a word-by-word basis rather than a line-by-line basis.

Simplicity of implementation is another of Pinnacle's major design goals. This is based upon the fact that Ben and his friends must be able to implement Pinnacle and still have enough time to write their paper. It is for this reason that Pinnacle is built upon git, an open source distributed version control system. By building upon git, Ben abstracts away the low-level complexities of implementing fault tolerant commits and recording version history. Furthermore, if implementation simplicity were to supersede minimization of conflict frequency as Pinnacle's main design consideration, a light version of Pinnacle could be built in which the standard line-by-line git merge is used instead of OT and word-by-word merge.

A tertiary design consideration is acceptable time and space complexity. In the cases where OT is successful, performance is comparable to Google Docs, which has been deemed acceptable by its many users. In the rare case that reconciliation uses git merge, there is slightly more lag, but this case is uncommon enough to be neglected. In terms of space, storing the entire document history is inexpensive due to git's exceptional compression algorithms. An alternative design would have not stored the document history at all, but this would not allow for rolling back on an OT failure and easily storing commits, which is a large price to pay for eliminating the minimal memory overhead imposed by git. Another design alternative is to delete version history once it is old enough, but this would add too much complexity for the marginal amount of space it would save.

Pinnacle's design abides by these design considerations and Ben's requirements. For example, documents are split into paragraph files so that, when using git merge to reconcile, a collaborator editing a paragraph does not conflict with another collaborator moving that paragraph. Additionally, a lock on the entire document is used whenever a collaborator attempts to resolve a conflict to ensure that it is only resolved once. Since the Pinnacle design ensures that conflicts will occur infrequently, overhead from this coarse lock is insignificant in the long run.

2 Design

The following sections describe the data structures, algorithms, protocols, and components of the Pinnacle system. Pinnacle is composed of the git repository storage system, the link layer, and the application layer.

2.1 Data Structures

Each Pinnacle document is represented by a git repository, which is used to track the evolution of each document throughout its lifetime. Pinnacle shards individual paragraphs into separate files and maintains a table of contents file named "toc.txt" to track the relative ordering of the paragraphs. Each paragraph file is assigned a globally unique identifier (GUID) to ensure that there are no collisions, even in offline editing mode. This GUID is both the filename of the paragraph and the value of its entry in the "toc.txt". The paragraph-based sharding allows Pinnacle to support paragraph moves without complex changes to the merging schemes, described later.

In order to support OT, each user will store a linear log for each paragraph file and for the table of contents. These logs are stored on disk outside of the git repository and are named by concatenating a GUID with the name of the file that it is logging for. Each log contains the username of the author along with an identifier and version vector for each add and delete operation performed by the corresponding author on the corresponding file. Data is appended to the log,

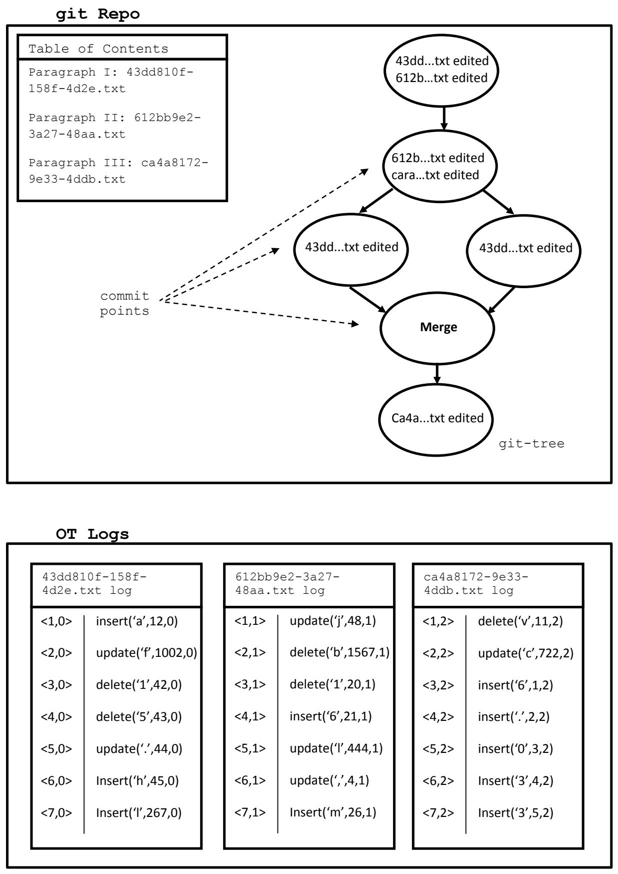

Table of Contents
Paragraph I: 43dd810f-158f-4d2e.txt
Paragraph II: 612bb9e2-3a27-48aa.txt
Paragraph III: ca4a8172-9e33-4ddb.txt

43dd810f-158f-4d2e.txt log	
<1,0>	insert('a',12,0)
<2,0>	update('f',1002,0)
<3,0>	delete('1',42,0)
<4,0>	delete('5',43,0)
<5,0>	update('.',44,0)
<6,0>	Insert('h',45,0)
<7,0>	Insert('l',267,0)

612bb9e2-3a27-48aa.txt log	
<1,1>	update('j',48,1)
<2,1>	delete('b',1567,1)
<3,1>	delete('1',20,1)
<4,1>	insert('6',21,1)
<5,1>	update('l',444,1)
<6,1>	update(',',4,1)
<7,1>	Insert('m',26,1)

ca4a8172-9e33-4ddb.txt log	
<1,2>	delete('v',11,2)
<2,2>	update('c',722,2)
<3,2>	insert('6',1,2)
<4,2>	insert('.',2,2)
<5,2>	insert('0',3,2)
<6,2>	Insert('3',4,2)
<7,2>	Insert('3',5,2)

Figure 1: Snapshot of Data Structures of Pinnacle

2.2 The git Repository Storage System

git is used to track the evolution of each Pinnacle document throughout its lifetime and to share the current working copy between all collaborators. In the case that a new user joins the collaboration, Pinnacle simply has to perform a "git clone", instantly getting a copy of the entire history. If a user wants to roll back to a previous version, this can be done easily since git stores the full history.

Updating to the latest version from other collaborators is also easy, with Pinnacle first performing "git fetch". Incorporating changes from other users can be performed with a "git merge". Finally, Pinnacle supports system-wide commits by utilizing "git tag", along with a cryptographic signature to verify that all users agree to that commit point.

2.3 Algorithms and Protocols

Pinnacle leverages two techniques for merging changes by multiple users. Pinnacle prefers to use operational transformation, a scheme for delivering conflict-free merges. However, if the OT process fails, Pinnacle will revert to using the git based system to perform merging.

2.3.1 Operational Transformation

Before Pinnacle can merge with OT, it must first elect a "server" from among the peers. Pinnacle chooses the server with a random voting algorithm. In the event of a tie, there is a run-off election. If more than five run off elections are held, the user with the highest ID becomes the server.

Next, all of the OT logs are sent to the server, which then redistributes them to the other peers. The server performs the OT merging, and distributes the result to all of the peers by creating a new git commit with the latest version.

The server is also responsible for maintaining a consistent document state while in live-editing mode, occasionally distributing the document in the form of git commits.

The operational transformation scheme is based on 3 operations: insert, update, and delete. Each of these operations modifies a single character at a particular location. When the document histories diverge, one user's operations are transformed so that they can be cleanly applied onto the others'. The transformations are as simple as changing the index with respect to the previous edit, depending on if it happens before or after. Since the paragraphs are separate, Pinnacle does not need an additional "move paragraph" operation.

OT operates by "replaying" one user's edits with respect to the others' in such a manner that will always produce the same document. The operations are transformed by computing the correct index to modify the character by incorporating the previous change history. For example, if there was an "insert at position 3" by user A, the operation "insert at position 6' by user B will be transformed to "insert at position 7". The correctness of OT can be proven by ensuring each operation follows the consistency model. OT methods have been proven correct, but their correctness is outside the scope of this document.

Once OT has produced a merged document, the history up until that unnecessary. If all collaborators reconcile up to that particular version, that history will never be accessed. However, if there a collaborator that has not reconciled since then, the old history will be accessed. Since Pinnacle supports dynamic membership, the full OT log must be preserved.

"Conflicts" (an operation at the same position) are resolved by the user ID, with the numerically lower user's edits going to the left of the cursor. Note that this scheme will result in systematically correct but possibly semantically incorrect resolutions that require manual intervention.

2.3.2 git Based Merging

git supports retrieving of collaborators' changes through the "git fetch" command, which imports their changes into the local repository, but does not perform any merging. Once all of the collaborators' changes have been imported, Pinnacle initiates a merge using git's algorithms. Such a merge is performed entirely offline, although the results of the merge can be shared with the collaborators once it is completed.

git's internal merge algorithms implement a three-way line-by-line merge. 3-way in the sense of there is the original copy, user A's new copy, and user B's new copy, allowing git to produce a more accurate, conflict-free merge. Assuming all edits were entirely independent, git will produce a new copy with all changes incorporated. If some of the changes are dependent, then this will produce a conflict that will require manual intervention. This scheme is able to handle most merges, but unfortunately will produce a conflict if a single word on the line is changed.

These conflicts can be avoided by augmenting git. Fortunately, git provides hooks that can be used to augment the standard reconciliation algorithm. Pinnacle takes advantage of this feature by providing a three-way word-by-word merge driver, which will produce a conflict only when several users modify the same word in different ways. This system will successfully detect word updates, deletions, and insertions, by comparing the new versions from both users to most recent common ancestor, and produce a new copy with the unified changes. The word-by-word algorithm can be implemented by taking git's existing merge driver and having it delimit on spaces rather than newlines.

2.4 The Link Layer

Communication between machines in both the internet-enabled and point-to-point case is handled via `ssh`. The authentication mechanisms (creation of user accounts, as well as knowledge of each other's IP addresses) is outside the scope of this paper. However, Pinnacle uses `ssh` due to its ability to further tunnel connections, as well as facilitate in directly transferring files as `scp` or `sshfs`, a FUSE-based filesystem that exports a directory over `ssh`.

The `ssh` connection is established between each online machine, forming a complete graph if possible. However, the complete graph is not necessary as changes will cascade through. The cascade property will hold as long as the connections are symmetric (i.e., if $A \to B$, then $B \to A$) and transitive (i.e., if $A \to B$ and $B \to C$ then $A \to C$). If for some reason the actual network connection does not satisfy the transitive property, then it can be emulated through `ssh` port-forwarding.

Pinnacle runs an `ssh` server, and then attempts to perform an `ssh` connection to all of the collaborators. Once the connection is established, it is used to transfer files between the peers, as well as send the OT updates.

2.5 The Application Layer

The user interacts with the Pinnacle system through the application layer. It presents a graphical user interface in which collaborators are able to edit the document and resolve conflicts when they arise. Additionally, the application layer keeps track of the location of new line characters in order to tell the git backend what paragraph files it should be keeping track of. Furthermore, the application layer is in charge of creating, deleting, and writing to the appropriate logs whenever a user adds or deletes a paragraph or a character.

At regular time intervals, the application layer decides that it is time to initiate a merge. Pinnacle will poll the peers to determine if they are online, and if they are, will perform a server election and break ties arbitrarily, as described earlier. Merging is necessary if there is any peer that has a divergent copy. Merging operations are performed by the server.

The server will be responsible for gathering the Operational Transform logs from all participants, merging them, and distributing the result. The server will also perform this operation during live-editing sessions. The operations will be both logged and transmitted to the server in this mode, unlike the offline-only mode of logging-only.

In the event that Operational Transformations fails, and a fallback to the git-based merge is needed, any conflicts that arise will be offered to all users to resolve. Any user that volunteers first will be given a 1 minute lock on the conflict. If the user does not resolve the conflict in this time, another user may take up the lock. The lock affects the entire document, to avoid an inconsistent state. Pinnacle ensures that the lock is exclusive by having the elected server act as arbitrator, which will serve the lock on a first-come-first-serve basis.

Once the document is complete, a user can initiate a system-wide commit point. The proposed version is created as a “git tag”, which is a name associated with a git commit. Every user must cryptographically sign the tag with their private key (automatically generated by Pinnacle) before the tag is considered valid.

If two users propose the same name for different versions, the underlying git system will detect the error and Pinnacle will cause the “later” tag (the one referring to a newer version) to be renamed. If this occurs, the tag is no longer cryptographically valid and must be re-signed.

Finally, the application layer helps support disconnected operation. When Pinnacle is operating offline, the application layer is responsible for querying the link layer regularly to attempt to connect to any nearby collaborators without using the Internet.

3 Analysis

The following sections present several use cases and performance metrics to determine how well Pinnacle performs using both the OT and git merge algorithms. Furthermore, the scalability limits of Pinnacle are investigated to add scope to the performance analysis.

3.1 Peer-to-Peer

At the git layer, Pinnacle supports peer-to-peer editing by simply building on git's native peer-to-peer framework. Each client maintains a local repository and can use the git operations to ensure synchronization with collaborators. While using OT, Pinnacle supports a peer-to-peer framework by choosing one connected client to temporarily perform the duties of a server.

3.2 Disconnected Operation

Pinnacle supports disconnected operation by storing all logs and updates in the local disk and git index. When the disconnected collaborator reconnects to the system, a merge is initiated to synchronize it with all other users.

3.3 Automatic and Manual Conflict Resolution

The OT merge algorithm is able to merge all changes without producing any conflicts by utilizing appropriate transformations of all operations. In the case that OT fails, git will run the word-by-word reconciliation algorithm, which will automatically merge all changes that are not in conflict.

When the git merge algorithm has run, it is possible that irreconcilable differences exist. Upon this event, the git backend will command the application layer to prompt all connected users to resolve the conflicts manually.

3.4 Ensuring a Conflict is Resolved Only Once

Once two or more users resolve a conflict, the resolved version is stored in the git tree of each user's repository. Once a system commit is accepted, this version is written into the git tree as the most current version of the file. git always references the most current version of a file in a git merge.

3.5 Ensuring Identical Offline Changes Do Not Conflict

When two users make the same change to a document while offline, such as fixing a spelling mistake in the same word, Pinnacle ensures no conflict resolution will be required. OT recognizes that identical insert, update, or delete operations occurred and therefore no transformation needs to take place. git's word-by-word merge driver is able to automatically recognize the same final word exists, and therefore raises no conflict.

3.6 Ensuring Unanimous Agreement on Commits

Any user can choose to initiate a commit, at which point Pinnacle uniquely names the commit with git tag and sends the commit request to all other users once they connect. Each collaborator has the choice to reject the commit or to accept it by adding their PGP signature to ensure agreement is cryptographically secure. Only when all collaborators have signed that commit point will Pinnacle accept it as a valid commit point.

3.7 Maintaining Internet-Independent Connectivity

Pinnacle supports any IP-based communication technology, given that IP discovery is done automatically. Upon connection, Pinnacle simply initiates an ssh tunnel to allow data access between collaborators. Reconciliation algorithms can be performed through this link to synchronize the collaborators.

3.8 Avoiding Unnecessary Historical Conflict Resolution

When Pinnacle uses OT to reconcile several collaborators, it will necessarily not resolve any unnecessary historical conflicts because no conflicts arise while using OT. In the case that git is used to merge, no unnecessary historical conflicts are resolved because all git operations used by Pinnacle reference the most current nodes in the repository

3.9 Propagation of Conflict Resolution

In a scenario where two users, Alice and Bob, synchronize, disconnect, and then independently interact with a third user, Charlie, conflicts do not need to be resolved twice. Without loss of generality, if Alice and Charlie resolve a conflict first, then Charlie's git tree will reflect this. When Charlie and Bob attempt to merge, the same conflict will not need to be resolved because git automatically recognizes that Charlie's version of the document is newer than Bob's and proceeds to overwrite Bob's older document.

3.10 Supporting Paragraph Shifts

As collaborators edit the document, Pinnacle automatically manages paragraph files and updates the table of contents. As shown by Figure 2, this allows Pinnacle to easily determine when a conflict was caused by a simple paragraph translocation, thus allowing it to suppress the conflict flagged by git. Note that Pinnacle is still able to handle the case when two users move the same paragraph to different locations, in which case a git merge should still produce a conflict.

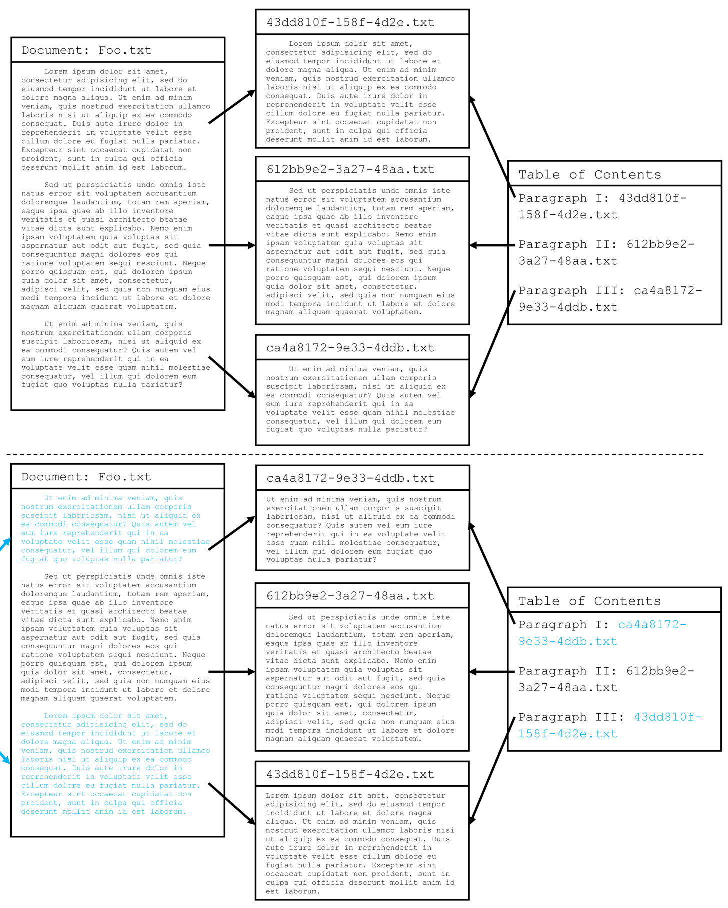
Document: Foo.txt
43dd810f-158f-4d2e.txt
612bb9e2-3a27-48aa.txt
ca4a8172-9e33-4ddb.txt
Table of Contents
Paragraph I: 43dd810f-158f-4d2e.txt
Paragraph II: 612bb9e2-3a27-48aa.txt
Paragraph III: ca4a8172-9e33-4ddb.txt
Document: Foo.txt
ca4a8172-9e33-4ddb.txt
612bb9e2-3a27-48aa.txt
43dd810f-158f-4d2e.txt
Table of Contents
Paragraph I: ca4a8172-9e33-4ddb.txt
Paragraph II: 612bb9e2-3a27-48aa.txt
Paragraph III: 43dd810f-158f-4d2e.txt

Figure 2: Supporting Paragraph Shifts: A user swaps two paragraphs in one document. The paragraph files, identified by a GUID, are unaffected by the swap. The table of contents file rewrites its pointers, allowing Pinnacle to correctly display the changed document.

3.11 Response to Failures during Commits

If a user's system crashes immediately after initiating a commit, Pinnacle waits for that client to reconnect to the shared session. At this point, the application recognizes that the user has a PGP signature on a specific document version that was named with a git tag. It then proceeds to propagate the commit as usual.

If any other user's machine crashes after a commit has been initiated, that user's PGP signature will be missing, which implies that Pinnacle will be unable to commit. As soon as their machine is back online, they will receive the commit as usual.

If any user continues to work after a commit has been initiated, git's maintains the committed version unchanged in the document history. Users are then able to PGP sign the committed version, even after having made changes after the commit.

3.12 Enforcing Unique Commit Names

Commits are named with the git tag command. Pinnacle keeps track of all commit names and returns an error if a user attempts to map a commit to an already used name.

3.13 Handling Concurrent Commits

When multiple users attempt to commit concurrently, Pinnacle uses the git hash identifier to remove duplicates and then proceeds to distributing all commits as usual. Collaborators can then decide what commits they want to sign.

3.14 Handling Dynamic Group Membership

git natively handles dynamic group membership. A new collaborator must simply set up a repository and use the IP addresses of the other users to locate them and obtain a copy of the document history. Pinnacle records the IP address of the new collaborator for future use.

3.15 System Usability

Pinnacle's central design consideration is to minimize conflict frequency. When OT is implemented alongside the table of contents file, no conflict ever needs manual resolution, allowing all users to concurrently edit a document or merge changes made in disconnected mode without having to pause and resolve conflicts. The trade-off of this usability is semantic merges, which are still correct in the sense of conflict resolution though possibly nonsensical when read. For example, if Alice and Bob are working disconnected with a sentence "I am enrolled in 6.033" and Alice changes the sentence to "I am enrolled in 6.046" while Bob changes the same exact version to "I am enrolled in 6.045," then the automatically resolved document would read "I am enrolled in 6.04645."

When OT fails, occasional conflicts have to be resolved manually. The word-by-word merge drive and table of contents file, however, allow for all conflicts to be resolved automatically except when multiple users make different changes to the same word. In practice, this does not occur often because collaborators usually work on different sections of a document, and even if they did work on the same section, the probability that they change the same word is relatively low.

3.16 Potential Problem Scenarios

OT, when correctly implemented, only fails when the computer performing the merges does not have enough RAM to store all of the logs. This problem may arise if there are a large number of offline users, all of which are making many changes and increasing the sizes of their logs. When they all reconnect and attempt to reconcile their changes, OT may fail.

In the case where OT fails and the word-by-word merge driver is used, many conflicts will arise if users have made changes to the same words. Given the large amount of users and changes that caused the OT failure in the first place, it is highly likely that many changes caused conflicts on the word-by-word level. In this situation, collaborators would have to manually resolve all conflicts.

While OT is able to run the live mode in real-time, if it fails then live editing is performed through the git layer. This will introduce noticeable but reasonable lag, resulting from infrequent but substantial commits and fetches to git. A possible solution is to engineer a more efficient protocol for initiating git fetches, presumably one that fetches more frequently. Such a strategy would certainly reduce the size of fetches, but it would eventually lead to a bulky repository and sluggish performance from git.

Another use case that may cause problems for Pinnacle occurs during the OT merge algorithm. Suppose that a computer is chosen to be the temporary server and, just before it finishes, it crashes. Then, another computer must be chosen to be the temporary server. If this happens enough times, OT merge will have considerable latency.

3.17 Scalability Limits

Pinnacle's OT is limited by the RAM of the user elected to mimic the server. When all collaborators meet, the temporary server must compute the possible state graph, which grows with the number of users as well as the number of changes. Assuming a 1GB lower bound on free RAM on most modern personal computers, an average of 4,000 changes per/user, and an average change size of 32 bytes, collaboration groups are limited to size $\frac{1\cdot 1024\cdot 1024\cdot 1024}{4000\cdot 32} \approx 8400$ users, which is a substantial amount.

As the amount of users grows, fetching changes from collaborators becomes a bottleneck. Assuming an average git repository of size 1 MB and a 54 Mb/sec lower bound on link speed, the "git fetch" operations would take over 90 seconds to fetch the entire repository from $\frac{90\cdot 54}{1\cdot 8} \approx 600$ other collaborators. Larger group sizes would simply introduce more lag, though this problem can be partially fixed by using a higher speed network.

The final bottleneck is based on user behavior. Many processes, such as signing off on commit points and resolving conflicts manually, require cooperation across the collaborators. If a particular user is slow in responding or cooperating, the entire system may have to wait for that particular collaborator. As the number of users grows, the problem becomes more and more unavoidable.

4 Conclusion

Pinnacle is simple peer-to-peer text editor that is designed to have low conflict frequency, considerable implementation simplicity, and modest time and space requirements. OT and git merge produce a minimal amount of conflicts and perform efficiently enough to provide a smooth user experience. git successfully abstracts away many of Pinnacle's lower level implementation complications while providing a space-efficient document history that is used to ensure safety from failure. What makes Pinnacle a truly innovative text editor, however, is its ability to allow collaborators to agree on a final document and to transition seamlessly into an offline collaboration mode.

References

[1] Hal Abelson, April 2012. Personal Communication.

[2] Ben Bitdiddle. 6.033 2012 design project 2, April 2012. URL `http://mit.edu/6.033/www/assignments/dp2.html`.

[3] Yasemin Gocke, May 2012. Personal Communication.

[4] Operational Transformation. Operational transformation. URL `http://en.wikipedia.org/wiki/Operational_transformation`.

[5] John Weighley. Git from the bottom up, December 2009. URL `http://ftp.newartisans.com/pub/git.from.bottom.up.pdf`.

Word Count: 4377