weWrite:

A Distributed Approach to Collaborative Text Editing

By Jacinda Shelly, Anna Shcherbina, Deborah Hanus

< jshelly >, <annashch>, <dhanus> @mit.edu

Butler Lampson, 10 am

TA: Nate Kushman

6.033 Design Project 2

May 6, 2010

1 Introduction

We present a client-server design for a collaborative, distributed text editor that makes use of write-ahead logging, TCP and serialization algorithms. Clients are able to edit collaboratively while online and edit a local copy when offline. Updates are sent from a user to a central server, which serializes the data coming from multiple users. Write-ahead logging adds updates to the server's log file, the updates are added to the server's copy of the document, and the server sends the changes to the clients and to backup servers. Figure 1 shows an overview of the system as a whole.

We made a number of decisions to ensure that the text editor makes efficient use of resources. First, we chose a client-server design to simplify serialization in the absence of faults. Second, we use back-up servers to protect against loss of service. Finally, regular fingerprinting ensures consistency in the face of network failures.

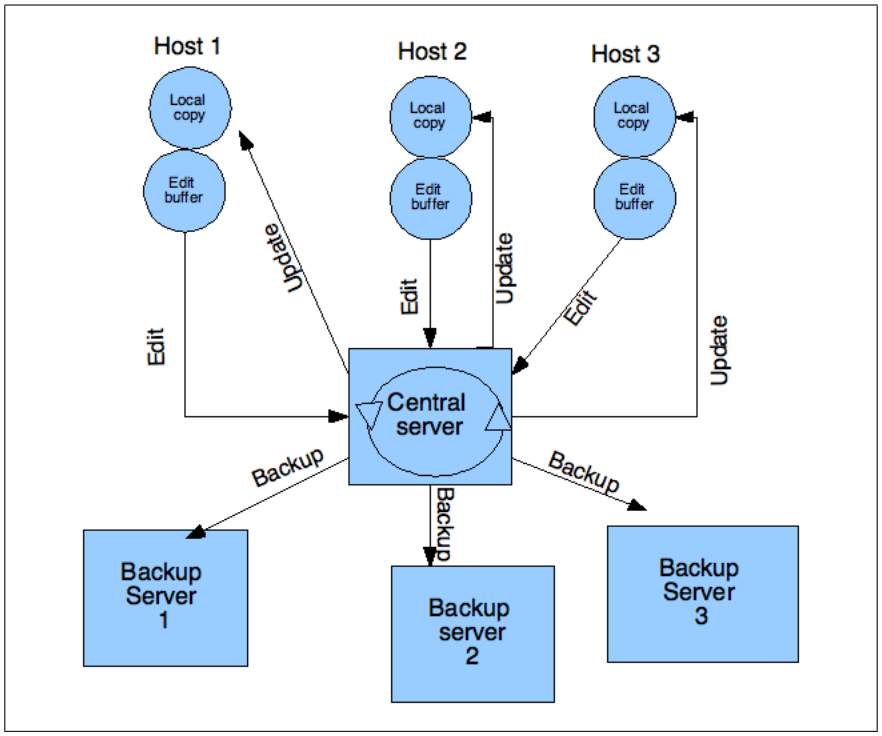

Figure 1: Edit Process Overview. Edits are stored locally in RAM buffer, before being transmitted to central server. The writes are added to the server's copy of the document, acknowledgement is sent to users that the addition was successful. Upon receipt of acknowledgement, the change is pushed from buffer to disk on the local host. The central server updates backup servers, allowing them to assume the role of central server in case of crash or network partition.

2 Design

2.1 Data Representation

2.1.1 Packet Sent to Central Server by a User

A packet sent by a user to a central server contains up to 1500 bytes of data. It consists of consecutive writes by the user. Each write corresponds to a character, which is uniquely identified by 5 bytes of meta-data and 3 bytes of Unicode data. In our notation, a character is fully specified by: **"action:visibility:user:chnum:val"**. Action and visibility technically each require only 1 bit, but the packet as a whole is easier to deal with if each update is in units of full bytes. The six unused bits could potentially be used to extend system functionality later.

Field	Type	Packet Space	Representative Capacity
action: *1 for for an insertion,* *0 for a deletion*	boolean	4 high order bits of first byte	NA
visibility: *1 for visible character,* *0 for invisible (deleted) character*	boolean	1 byte	NA
user: *identity of user who entered current character*	short int	1 byte	32000 users
character number (chnum): *tracks order of characters inserted by a specific user*	long	3 bytes	16.7 million chars/ user (7500 pages)
character value (val)	unicode encoding	3 bytes	

2.1.2 Update Packet Sent from Central Server to Hosts

Update packets have the same structure as the original packets sent by a user to the central server with an added timestamp field. A timestamp is added to updates at the time that the packet is processed by the central server, ensuring that each client knows the absolute ordering of all updates. This is important because TCP guarantees in-order processing between the client and server, but not between all clients.

2.1.3 Fingerprints

A fingerprint consists of a checksum calculated for the entire document (either the local copy or the shared copy on a central server). Fingerprints also contain a timestamp, indicating the time at which the checksum was calculated. Every 24 hours all active servers are synchronized with an atomic clock to ensure that timestamps remain consistent.

2.1.4 Server Hierarchy

The network consists of a central server, which integrates writes from all users into a central copy of the document, and backup servers, which keep backup copies of the document and can assume the role of the central server in case the central server fails or the network is partitioned. The number of backup servers will vary based on the congestion in the network and the number of users that are editing a document. A circular linked list is used to store the server hierarchy – the order in which backup servers assume the role of the central server. All hosts are given a copy of this list when they come online. Every node stores a pointer to the current central server. When a new node assumes the role of central server it broadcasts this to all connected hosts, who shift their pointers accordingly.

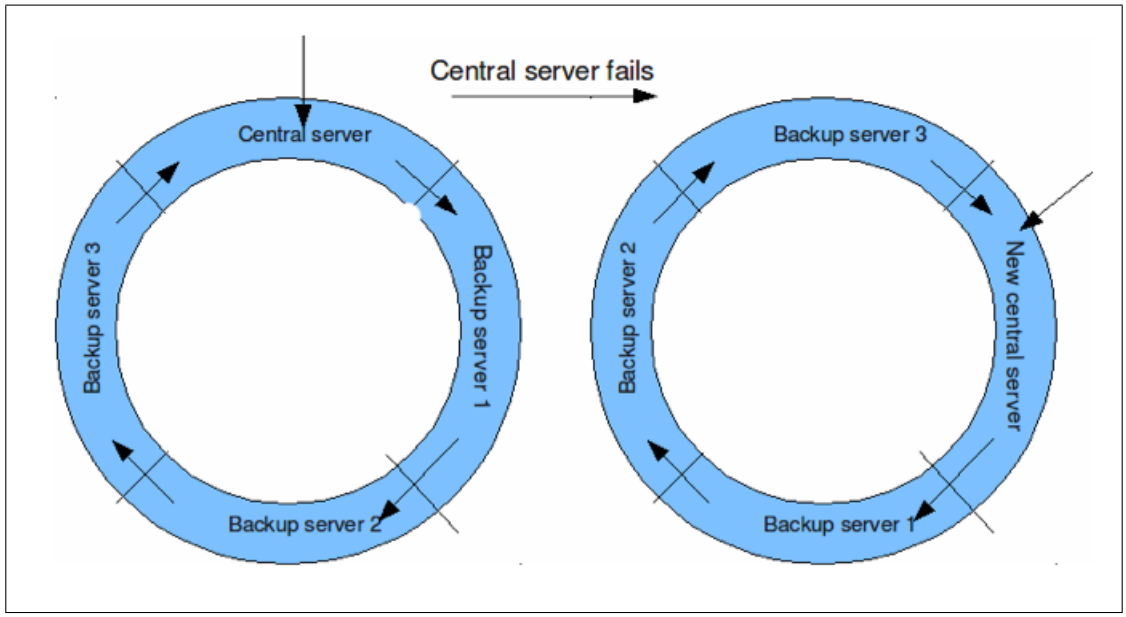

Figure 2: Server transition is implemented as a circular linked list of which all hosts possess a copy. Users point to the current central server. Upon failure, the next active server in the linked list will take over as the central server.

2.1.5 Logs

A write-ahead log protocol is used to maintain logs on the servers. Updates are committed when they are written to the log on the central server's disk. When a node in the network crashes and

subsequently comes online, the log file is used to restore the node's copy of the document to the state at the last commit.

Users maintain logs of all updates received from the central server. However, they do not guarantee write-ahead logging. We are willing to tolerate a small amount of user data loss, as most lost writes can be restored with the fingerprinting protocol (see section 2.3).

2.2 Atomic Actions

2.2.1 INSERT

When a character is inserted into the document, the user creates a new character by incrementing chnum, appending the appropriate character value, and setting the visibility field to 1. The UID field should be constant for a given user. The insertion location is specified using the unique identifier of the immediately preceding visible character. Below, User 1 has inserted characters "A", "B", and "C" into the collaborative document (Figure 2).

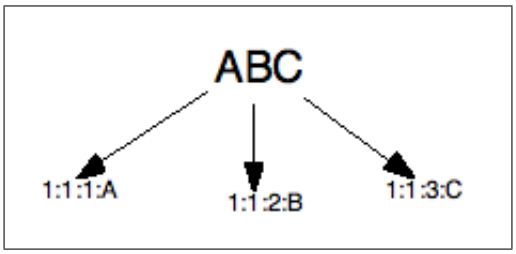

Figure 3: This figure demonstrates the state of the working copy after User 1 has entered characters a,b and c in succession. All fields are marked visible. The second field indicates user id, the third field is an index of a users characters according to the order they were entered and the fourth field contains the physical characters unicode encoding.

2.2.2 DELETE

A deletion will be indicated by an action field of 1 and a visibility field of 0. From the perspective of a host, a deletion is treated identically to an insert with the exception that only a single bit needs to be overwritten, instead of adding 8 new bytes to the file. Deletions only require 8 bytes of data to be sent of the network and logged, since it is unnecessary to include a preceding character. The character then becomes invisible to the users editing the document. However, the character is not actually deleted, in order to more easily ensure eventual consistency if some users have received a deletion while others have not, since all users will still know where to place updates. The logging procedures for a deletion are identical to an insertion.

If a user inserts and deletes a character in quick succession, it is possible for the delete command to reach the central server before the insert command (this also assumes the TCP connection is disrupted between the inertion and deletion). In this case, the central server should store the delete

command until the corresponding insert arrives, and then process the delete. The server will not add a timestamp to the delete until after the corresponding insert has been processed.

All updates are placed in a buffer in RAM on the user's machine, and the user's own insertions and deletions will immediately appear to them. At five second time intervals all new updates contained in a user's buffer get sent to the central server via the TCP protocol as a packet containing concatenated updates of 16 bytes each. If we assume packets can hold up to 1500 bytes, a packet can contain up to 93 updates. If a user has inserted or deleted more than 93 characters in the previous 5 seconds, the updates are split into multiple packets. When a user receives an update response from the central server, those updates are removed from the buffer and written to disk.

In the case of poor connectivity, an update buffer contained entirely in RAM could pose a data loss hazard. Therefore, once every minute all new unacknowledged updates are copied to disk. This prevents clients from losing more than a minute's worth of data.

An update buffer can also grow large when faced with poor connectivity, which could pose a problem on clients with a small amount of available memory. Therefore, if the size of the update buffer exceeds exceeds a certain maximum capacity determined as a percentage of available RAM, all further updates will be written to disk and marked as unacknowledged until the server responds.

To insert characters "DEF" between "B" and "C," User 2 sends a packet to the central server containing the information in Figure 4.

New Character	Preceding Character
1:2:1:D	1:1:2:B
1:2:2:E	1:2:1:D
1:2:3:F	1:2:2:E

Figure 4: Example packet: If User 2 wishes to insert the characters "DEF" between characters "B" and "C," User 2 sends a packet containing these three updates to the central server.

The central server timestamps packets as they arrive, and packets are processed in order of timestamp. If more than one packet has the same timestamp, a random number generator will be used to determine a processing order. The server will iterate through each character in a packet. An example is presented in Figure 4.

Once a server has committed the changes in a packet, it sends a packet containing the edits to every host. TCP ensures packets arrive whole, so all updates in a given packet will be received. The timestamp appended to the update packet corresponds to the time when all updates in a single packet have been written to the server's log, i.e. the commit time. This implies that packets are the unit of atomicity for the server. A copy of the original packet with the timestamp appended will then be sent to the backup servers and clients as described above and all hosts will update their local copies accordingly. The timestamp is what ensures consistent inter-user ordering.

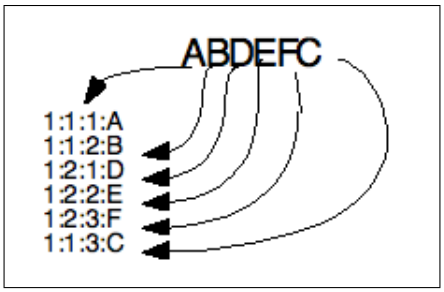

Figure 5: This figure demonstrates the state of the working copy after User 1 has entered characters a,b and c in succession. All fields are marked visible. The second field indicates user id, the third field is an index of a user's characters according to the order they were entered and the fourth field contains the physical character's unicode encoding.

Previously, the host that initiated the inserts wrote them to a buffer in RAM or to a separate location on the disk. When it receives an update from the central server, the inserts will be written to disk, and will be deleted from the RAM buffer or the temporary location on the disk. Thus, the update from the central server also functions as an acknowledgment that insertion completed successfully. If the host does not receive this acknowledgment within a timeout interval, it will resend changes for which it has not received update confirmations. At this stage, all hosts should have synchronized copies of the document.

2.3 Fingerprinting Protocol

A fingerprinting protocol, (e.g. the Raven scheme or a suitable alternative) is used to ensure eventual consistency in the face of failures discussed in further detail in sections 2.6, 2.7 and 3.3.

Every 30 seconds, the central server creates a fingerprint of the document as it was two minutes previously. The resulting fingerprint is sent to all users and backup servers, who also calculate a fingerprint of their document at that time. If the two values are consistent, the backup server or client and central server are synchronized as of that checkpoint. The host sends an acknowledgement to the central server, indicating that it is synchronized.

If the fingerprints do not match, the host indicates this in the acknowledgement packet, along with the most recent time synchronization was confirmed. The central server responds with all updates between that synchronized time and the checkpoint currently being calculated. These updates are compared with the updates in the host's log. If the host receives update packets it did not previously possess, it appends them. Conversely, if the host possesses updates that the central server does not (which could occur if a backup server had previously been acting as a central server during a network partition), it sends these updates to the central server. If both hosts have the same update packet but with different timestamps, the most recent timestamp is used. The fingerprint is then recalculated at the central server and resent to all other hosts. This process may need to occur multiple times before consistency is finally reached.

If the central server does not receive an acknowledgement from a host within a time-out period, it will resend the fingerprint. Since fingerprints are idempotent, a server can send the fingerprint

for the same set of edits to a node multiple times. However, after a certain number of unsuccesfull attempts, the central server will assume that the node is offline.

Fingerprints are calculated at checkpoints in the past because the system is likely to remain in flux with respect to the most recent set of updates and the current state of the document, making it impossible to check for consistency in the present. It is not useful to calculate a fingerprint until enough time has passed for the correct set of updates to propagate to the majority of nodes in the network. The document is consistent for a certain set of updates when all fingerprints corresponding to that timestamp have returned as synced.

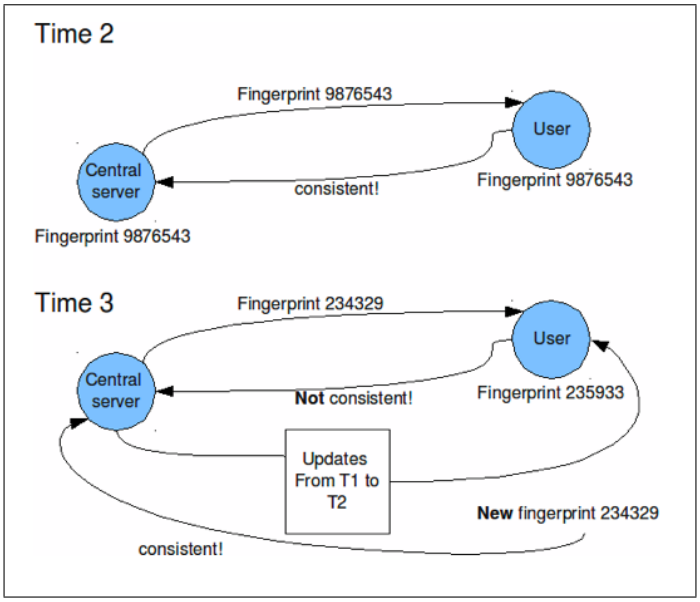

Figure 6: The fingerprinting protocol. Fingerprints are calculated to reflect the state of the document as of some time in the past (at T3 the fingerprint is calculated for the time interval from T1 to T2). If two documents are inconsistent, they exchange updates until consistency is achieved.

2.4 Adding New Users to the Document

When a new user is added to the document, it sends a message to the central server announcing its presence. The central server responds by sending the current server hierarchy to the user, and request the first backup server to send the new user the most recent consistent copy of the document. As a performance optimization, multiple backup servers could split the load, but the advantage of a single server is that TCP will ensure that the entire document is sent correctly.

As the new user is downloading the document, it will continue to receive updates from the central server, which will be written to a separate location on disk until download is complete. While a user is downloading a document, it will not receive fingerprints because it is highly likely that any fingerprint sent during this time interval will result in inconsistent checksums between the local and central copies. When the user has finished downloading the document, it will incorporate all updates received during the download period. It will also send an announcement to the central server indicating that it is ready to receive fingerprints.

2.5 Working Offline

Our system supports users working offline is a relatively straightforward manner. When a host cannot connect to a server for an extended period of time (e.g. longer than a minute or another appropriate threshold), the host begins to log all changes made to disk instead of to RAM. It also indicates in the log that a sync should be conducted when the host is next contacted. A user also has the option to deliberately move their work offline.

When a user reconnects to a server, all changes are sent to the server with an indication that these are changes that are being resynced. Large inserts are not a problem because most of the characters will be inserted relative to local characters and so will appear as full sentences, paragraphs, etc. in the document. What the sync protocol does check for are potentially unintentional duplicate entries, illustrated in Figure 5.

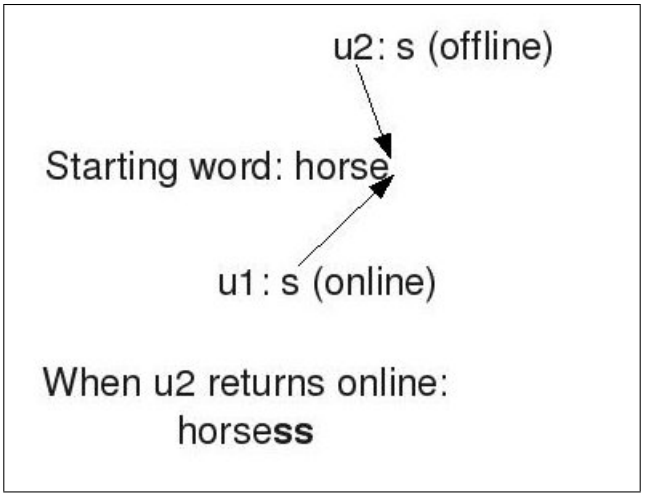

Figure 7: When a user changes something offline, the same change may be made by another user online. To aid users who might not want to manually find and remove these duplicates, the server checks for potential occurrences of this when a host returns from an offline state.

2.6 Crash Recovery

Every server pings all other servers on the network at 1 minute intervals. Ping messages include an identifier that marks the sender as either a central server or a backup server. This allows every active server to maintain awareness of which servers are currently online.

2.6.1 Central Server Crashes

Assume User 1 has sent Packet A to the central server, which we refer to as Server I . The server updates its copy of the document and commits the write succesfully, but crashes afterward. The system may respond in several ways, all of which result in eventual consistency among all components.

• Server I was unable to send update packets to any of the backup servers or users. A backup server, server II, will then take on the role of the central server. User 1 did not receive confirmation (in the form of an update) for packet A, so it retransmits packet A to server II. Server II will commit packet A and send updates to all users/ backup servers. These updates will contain a timestamp. Thus, the timestamp for packet A will be the same among all users and backup servers, but it will be different from the timestamp on the original, crashed central server. When server I comes back online as a backup server, it will receive a fingerprint from server II, revealing an inconsistency in the timestamp of the two writes. Server I will change the timestamp of packet A to reflect the more recent value.

• Server I was able to send an update/confirmation to user 1, but a subset of the other users did not receive the update. In this case user 1 will not retransmit packet 1. When sever II assumes the role of the central server, it will send fingerprints to the users, realize that user 1 has an update for packet A, and will add packet A to the central copy of the document and timestamp updates with the same time as recorded on user 1.

• Server I was able to send an update/confirmation to a subset of the users, but not to user 1. Thus, user 1 will retransmit packet A to server II, which may or may not have received the update for packet A from server I. If server II already has the update, it will disregard the retransmit of packet A. However, if server II does not have this update, it will add packet A to the central copy of the document, log it, and send an update to all users/ servers. The subset of users who had received the original update will replace the timestamp with the more recent one. Server I will replace the timestamp with the more recent one when it comes back online and receives a fingerprint.

2.6.2 Backup Server Crash

When the backup server comes online again, it is considered less reliable than other backup servers and thus gets added to the end of the server hierarchy. When the backup server receives a fingerprint from the central server, it will be unsynchronized with the central server. Consequently, the central server will send to the backup server the writes made to the central copy since the time the two servers were last synchronized.

2.7 Dealing with Network Failure

If the central server crashes, the hosts will resend their packets a specified number of times before assuming that the central server is down. The hosts will then begin to send packets to the backup server next on the list, and this backup server will send out a message to all users informing them to change their central server pointer to itself. The number of backup servers will vary based on the congestion in the network and the number of users that are editing a document. If more than one backup server is used, they will assume the role of central server in the order specified in the server hierarchy linked list.

2.7.1 Network Partition

In the case of a network partition, the users on one side of the partition will send packets to the next specified backup server, which will assume the role of the central server. Thus, it is possible to have multiple central servers at one time, if they are on opposite sides of a partition. When the partition is removed, one central server will receive a fingerprint from another central server. When this occurs, the server that assumed the role of central server most recently maintains this role, while the other central server assumes the role of a backup server. The fingerprint will also indicate that there are differences in the two servers' copies of the document and the fingerprinting protocol described in Section 1.3 will allow formerly partitioned hosts to resynchronize.

2.7.2 Backup Server as Router

Another failure case occurs when the central server does not become completely partitioned but becomes inaccessible to a fraction of the hosts, while the remaining fraction is able to continue sending edits. In this case, the backup server will act as a router, forwarding the disconnected user's writes to the central server and forwarding updates/ fingerprints from the central server to the user. Alternatively, we could have enabled the backup server to assert control and become the central server, but minimizing changes in server status makes it easier to achieve consistency more quickly.

3 Analysis

In this section, we will discuss overall strengths and weaknesses of our design, give estimates of bandwidth consumption and latency, outline how we handle special cases where the system does function in the expected manner, and discuss the maximum amount of data that could be lost in our system - a minimal amount that is justified by the decrease in latency.

3.1 Overall Strengths and Weaknesses

3.1.1 Strengths

In the absence of failures, our system's primary strength is that the central server acts as a single arbiter who decides the canonical order in which updates are made to the document. This makes achieving eventual consistency considerably easier. In cases where the central server fails or the network is disrupted, our fingerprinting protocol ensures that after the network returns to a single central server, eventual consistency will prevail.

Periodic fingerprinting with checkpoints is also a strength of our system, because the detection of inconsistencies does not necessitate sending a copy of the entire document, but only the updates that have occurred between the last checkpoint and the current checkpoint.

A client-server model also simplifies the process of choosing the central server and server hierarchy since those decisions can now be made at configuration time.

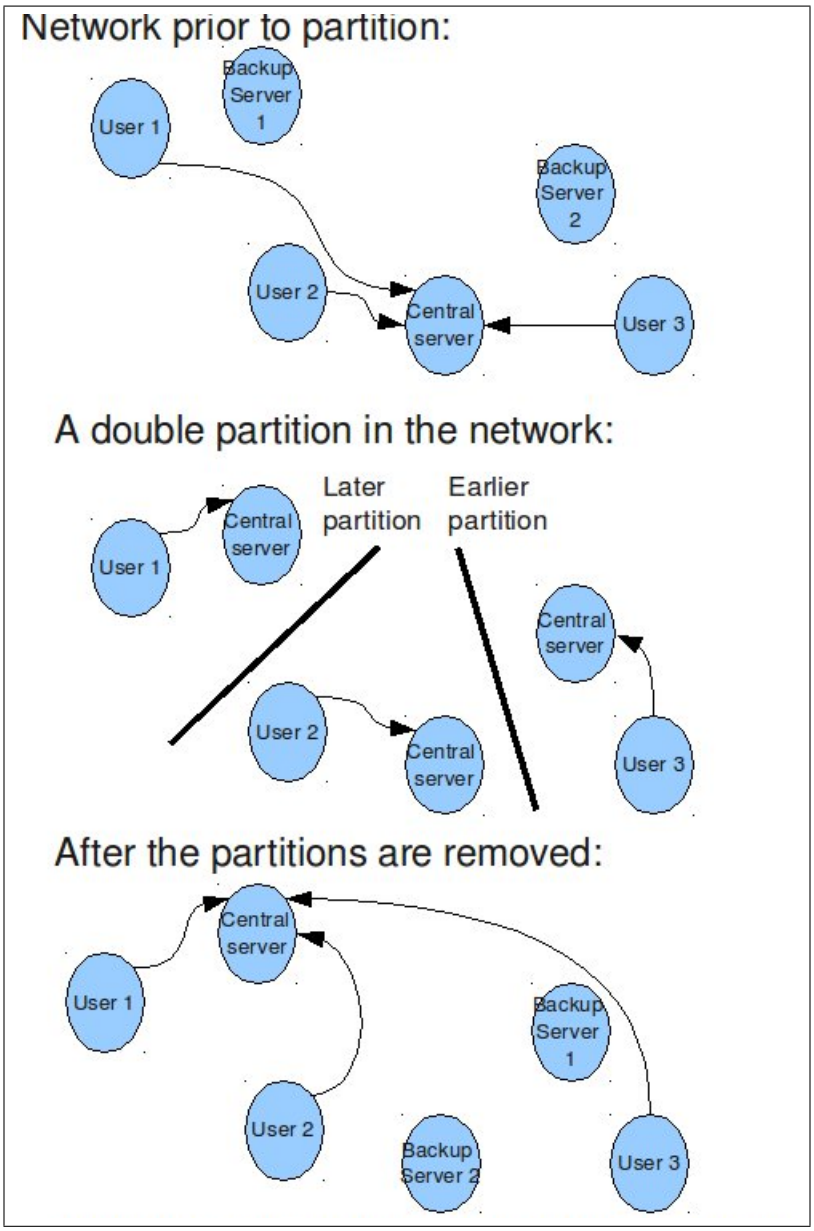

Figure 8: When a partition is created, the first backup server on each side of the partition assumes the role of the central server for the nodes on that side of the partition. When the partition is removed, the server that assumed the role of central server most recently maintains that role. The other central server gets shifted to the role of a backup server in the server hierarchy.

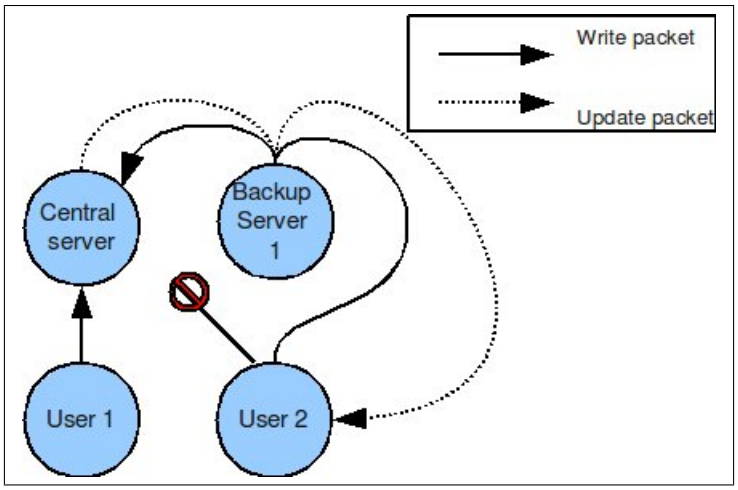

Figure 9: Backup server as router. When user 2 loses its connection to the central server, it sends writes to the first backup server in the linked list hierarchy. If the backup server is able to access the central server, it will forward all traffic to the central server from user 2 and vice versa.

Backup servers prevent a single point of failure, and logging all updates to disk at the central server before fowarding updates to the backup servers and other users ensures eventual recovery of all updates in the event of a crash. The fact that our system also uses TCP to ensure packet receipt guarantees that we cannot lose more than a minute of user data in the absence of disk failure. Indeed, even if the disk does fail it is possible to recover the vast majority of the document from one of the multiple other copies.

3.1.2 Weaknesses

This model increases the minimum latency of a system compared to a peer to peer model, where the minimum latency is simply the latency between two hosts. In our system, the latency between any two hosts is the sum of the average one-way latency between those hosts and the server.

A centralized model also has the weakness that the servers must be maintained and expanded as the system grows. This means that our system's maintenance must be funded in some manner, whereas the cost of a peer to peer model is only the cost of the initial development of the system and the cost of distributing the software to potential users.

We also recognize that having a single server is a bottleneck, especially if a document is shared by a large number of users, but we were told not to attempt to address this issue in our design.

3.1.3 Our Design Choice

In the end, we decided to use a centralized model because of the simplicity this introduced in terms of maintaining eventual consistency. A complex part of a peer to peer design is how to synchronize so it is possible to apply all updates in the same order across users. In a centralized design, this is easy.

On the other hand, when hosts fail, especially if that host is the central server, recovery methods can be complex. However, the ease of synchronization and ensuring consistency in the steady-state versus the complex syschronization problem of P2P makes up for the effort of determining a correct recovery protocol.

3.2 Bandwidth Consumption and Latency Analysis

The average latency for an update to be propagated to all users is the average one-way latency from the updater to the server plus the processing time at the server plus the average one-way latency from the server to all users and backup servers.

The average maximum latency of the system is the average one-way time from the furthest user to the server plus the average one-way time from the second-furthest user to the server plus update processing time at the central server.

3.3 Operational Scenarios

This section discusses how our system handles several different edge and corner cases with regard to updates. Though these behaviors can be extrapolated from our description of how the system generally works, it is useful to clarify in the context of specific scenarios.

3.3.1 Insertion and Deletion Corner Cases

Insertion of a character at the same location by two users

Users may observe temporary inconsistencies in their local copies until the central server processes all incoming packets and sends the corresponding acknowledgments. This will occur when users attempt to write to the same location in the document in quick succession. Consider the following scenario: the central server begins with document "ABC". User 1 wishes to insert "Y" after "A", creating document "AYBC". User 2 wishes to insert "X" after "A", creating document "AXBC." The following chart represents the state of the central server, the buffer of each host, and the disk copy of each host as these changes are processed. In this case, User 1 makes a change first, followed quickly by User 2 at the same location. The server processes and inserts User 1's information followed by User 2's information.

Users are able to view their updates onscreen as soon as these updates enter the buffer, so User 2 will observe a switch in ordering because it will receive instructions from the server to insert "Y" after "A" at time 2, then receive confirmation of the final state of its update only at time 3.

Thus, the order of the X and Y as observed by User 2 will change between timesteps 2 and 3, but eventual consistency is maintained, since both hosts have identical local copies at time 3. This issue presents a minor inconvenience to users, but it will only occur if two users write to the same spot in the document within several seconds of each other. Thus, we do not take steps to correct the temporary inconsistency.

Insertion of words or phrases at the same location by different users

Time	Central Server Copy	host 1 buffer/ disk	Host 2 buffer/ disk
0	ABC	AYBC / ABC	ABC/ABC
1	AYBC	AYBC/ABC	AXBC/ABC
2	AXYBC	AYBC/AYBC	AYXBC/AYBC
3	AXYBC	AXYBC	AXYBC

Figure 10: State of the buffer and disk for the working copy and the local copies of the document following a series of inserts

Time	What host 1 observes:	What host 2 observes:
0	AYBC	ABC
1	AYBC	AXBC
2	AYBC	AYXBC
3	AXYBC	AXYBC

Figure 11: Example of manner in which characters may change position in the users prior to eventual consistency

A different issue may arise if parts of a single word are allocated to different packets. Consider the scenario in which the central server copy consists of empty quotation marks “”. User 1 wishes to write“hello”, while User 2 wishes to write “world” (Figure 11)

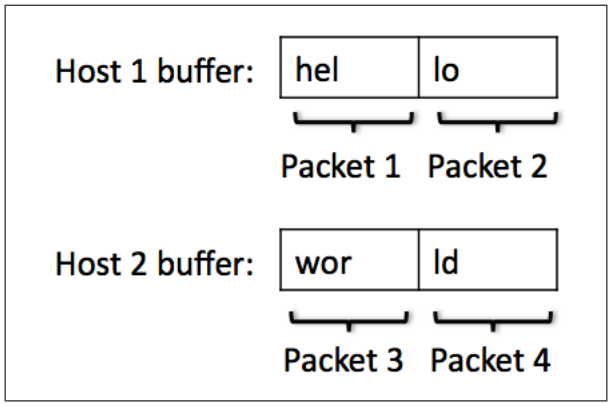

Figure 12: An example of the packets sent to the central server by two users editing the same location in the document. Though the word order may be either ”helloworld” or ”wordhello” depending on the order packets are received, the fact that character insertion uses the preceding character means that users typing continuously should always see full words, but the ordering of those words will be based on which initial fragment first arrives at the central server.

If these writes occur in quick succession, it is possible that the server will receive packets in the order 1, 3, 2, 4. The document on the central server will change as shown in Figure 12.

Packets received	Document on Central Server
0	""
1	"hel"
1,3	"wohel"
1,3,2	"wohello"
1,3,2,4	"worldhello"

Figure 13: Example of unfavorable packet splitting and the impact on the working copy of the document

Thus, the individual words will be coherent, but they will be out of order. Any ordering of packets will result in either "helloworld" or "worldhello". In fact, in all cases a contiguous set of characters written by a single user without moving his cursor to another location in the document will be placed in order such that all characters a single user typed will remain a set. However, sets of characters written by different users cannot be guaranteed a specific order, since all messages are processed as they arrive at the server. This error will occur only when users edit the same location in the document in rapid succession. Therefore, it will be the responsibility of the individual users to correct such errors manually.

Deletion of the same character by multiple users

Since all characters are uniquely identified, and deletion merely involves setting the visibility bit to 0, multiple deletions are idempotent. To maintain consistency in our fingerprinting scheme, a deletion updates in the log will change the most recent timestamp, but not the appearance of the document.

Insertion by one user followed by deletion by a second user

If the document holds a string"mnp" and user 1 inserts a"1" after the "m" to produce "m1np," this will be propagated to all users in the normal manner. When any other user sees "m1np" they are free to delete this and have it propagated to the rest of the system in the usual manner. The central server will arbitrate.

3.3.2 Handling Crashes

User Crashes

If a user crashes, when they come back online they will inform the server of the time of their last consistent checkpoint and request all updates that have occurred since that time. This is a special case of the fingerprint protocol where the user knows that it is inconsistent and requests to synchronize.

Server Crashes

If the central server crashes, control passes to the first backup server on the linked list as described in section 2.7. When it does recover, it uses our fingerprint protocol to resynchronize. A backup server crash is treated in the same manner, but no transfer of control is necessary.

If multiple servers crash, the linked list is traversed until the first active server is reached - this server then becomes the central server.

3.4 Maximum possible loss of data

The maximum possible loss of data that can occur in the system is what has accumulated in the RAM buffer but has not yet been received by the central server or written to the disk by the updating host. When the user is online, this can be at most one minute's worth of data which has not yet been received by the server or written to disk. When the user is offline, these numbers remain the same.

We assume that packet losses will not occur because we are using TCP for transmission. Therefore, we assume data can only be lost due to a crash.

3.5 Fingerprinting: The Ultimate Bulwark against Inconsistency

In determining how often to conduct fingerprints, it is necessary to balance the CPU and bandwidth consumption resulting from frequently sending fingerprints against the cost of resending the entire document or all updates that have occurred.

We used rather a rather arbitrary example of performing fingerprint checkpoints every 30 seconds at points 2:00 minutes prior to the current time (as determined by the central server - the server will not attempt to create a new checkpoint until the time it is checking is consistent). This seems reasonable assuming that fingerprints are relatively easy to compute, as frequent checkpoints reduce the probability that large numbers of updates will need to be sent in order to synchronize.

3.6 Performance Optimizations

A potential performance optimization would involve deleting invisible characters from the central copy of the document. This would be helpful in reducing the size of the file and the corresponding latency when portions of the document are sent to backup servers and clients during fingerprint checking. Invisible characters can be removed if they have not been accessed for a long time (the specific time interval depends on the particular parameters of the document).

Another potential optimization involves designing users that send active requests to the central server for updates that they do not have. For example, if a user wrote a character but did not receive an update/acknowledgement within one minute, it would request the central server to send the ack. This optimization would decrease the likelihood of any node falling more than a few minutes behind in updates. Thus, when fingerprints discover any discrepancies between the local and central copy, these discrepancies would generally be minor and not require transmitting large

portions of the document. In implementing this optimization, it is important to balance this benefit against potential increases in bandwidth consumption resulting from additional requests packets being transmitted across the network.

3.7 Conclusion

weWrite's design is designed to provide a simple method of synchronization in the absence of failures, while continuing to provide eventual consistency in the face of network losses and host crashes. The central server synchronizes updates and uses write-ahead logging to prevent data loss. Backup servers provide continual service in the face of a central server crash or network failure.

Our design ensures eventual consistency via a fingerprinting protocol that will ensure that all hosts eventually possess the same set of all updates, even in the presence of failures. In the absence of failures, fingerprints would not be necessary.

This design would be an attractive option for organizations that already have a server infrastructure which could support this type of system, as it works hard to ensure consistency quickly in a low-failure environment.

3.8 Acknowledgments and References

Many thanks to Nate Kushman for his discussion and critique of our ideas, which influenced the development of our final design. We would also like to thank Nicholas Shelly for helping proofread our near-final version of this report.

WORDS-IN-TEXT: 4956