CollabDraw: Real-time Collaborative Drawing Board

Prakhar Panwaria Prashant Saxena Shishir Prasad
Computer Sciences Department, University of Wisconsin-Madison
{prakhar,prashant,skprasad}@cs.wisc.edu

***Abstract* —** Designing a scalable and fault-tolerant distributed system is a challenging pursuit. We took up this challenge to develop CollabDraw, a distributed real-time collaborative drawing board. CollabDraw provides a framework which is fault-tolerant and highly scalable. It provides a mix of strong and eventual consistency [1] to users depending on load characteristics of the system. Our system is also capable of operating in disconnected mode, making it operational at all the times.

Keywords — distributed systems, collaborate, drawing

1. INTRODUCTION

We worked on designing and developing Collabdraw, an application which allows multiple users to collaborate on the same drawing board in real time. Such an application can also be built using a single server system. A clear advantage of single-server model is that it provides strong consistency without any additional effort, which is the most desirable characteristic for such an application. But the problem with using single server is that it has minimal scalability, as it cannot handle large number of users. Also, since it would be a single point of failure, it has negligible fault tolerance. CollabDraw, built over a distributed system, overcomes these limitations by making the system highly scalable and fault tolerant, while retaining strong consistency.

A major motivation was that current state-of-the-art collaboration systems like Google Docs [2], do not allow clients to make changes while being offline. We believe this is a pressing problem to solve if we want to make our system truly fault-tolerant and robust. In order to overcome this problem, CollabDraw has added capability of operating in offline mode, where it allows users to make drawings even in the drastic event of a network failure or absence of any worker servers in the system.

To have a consistent view of a single board shared among different users, the system needs to communicate latest updates to the clients in real-time. And, in order to make the system robust, it also needs to have multiple servers synchronized with latest changes made to all the boards. Our implementation of CollabDraw has the following salient features -

1. *Hybrid Consistency*

For each drawing session, CollabDraw provides two level of consistency - Strong and Eventual. Users connected to a set of preferred servers are guaranteed strong consistency while we provide eventual consistency between all servers outside the preferred server group.

2. *High availability*

CollabDraw replicates session data on multiple servers, ensuring that the data is accessible in case of network partitions. As servers come and go, our system migrates clients to different machines, assuring access to the drawing board.

3. *Scale out architecture*

CollabDraw is designed with scalability in mind. The registration process of a worker server with the system has been kept extremely simple which adds flexibility to the system. Hence, our system can be easily scaled by adding additional worker servers depending on the anticipated load on the system.

4. *Offline mode*

In case of extreme server outages or network failure at client end, our design supports disconnected mode of operation which allows client to work in offline mode. As soon as the connectivity is restored, our system seamlessly reconciles the data with other users.

Section 2 provides details of CollabDraw design. It also gives an outline of client-server interaction. We evaluate our design in Section 3. In Section 4, we describe the related work being done while building collaborative editing systems. In Section 5, we talk about the future work that can be done to further improve the performance of our system. We list our conclusions and key takeaways in Section 6.

2. DESIGN AND IMPLEMENTATION

2.1 Components

CollabDraw architecture is shown in Figure 1. There are three main components in the system –

1. *Client*

Clients are users who join the drawing session using some HTTP client, most commonly a web browser. However, other forms like embedded HTTP clients within mobile applications are also possible. Clients use HTML5 local storage [3] to store events performed during disconnected mode.

2. *Session Manager*

Session manager is the heart of CollabDraw architecture. It is an always available service which provides an initial landing interface to the clients. It is similar to Master server in Google File System[4]. Session manager is responsible for maintaining metadata information about the currently active sessions. It also maintains periodic heartbeat with worker servers and stores their system load profile.

3. *Worker Servers*

These are the workhorse of CollabDraw. After contacting Session manager, clients are redirected to one of the worker servers which handle all subsequent interaction from clients. They are responsible for rendering the drawing board along with storing all drawing board related data.

Before our system can begin serving clients, we need to bootstrap it by adding worker servers. The number of worker servers to be added depends upon the anticipated initial load on the system. Later, worker servers can be added or removed easily depending upon the future load on the system. A worker server registers itself with the session manager by sending a registration request to the session manager. On receiving a registration request, Session Manager stores the information of the new worker server in its persistent database and initiates a heartbeat with it.

2.2 Workflows

In this section, we discuss client-server interaction during different mode of operations. In our design, clients and servers interact over HTTP protocol. For persistent connections, we use WebSocket [5] protocol to provide full-duplex communication channels over a single TCP connection.

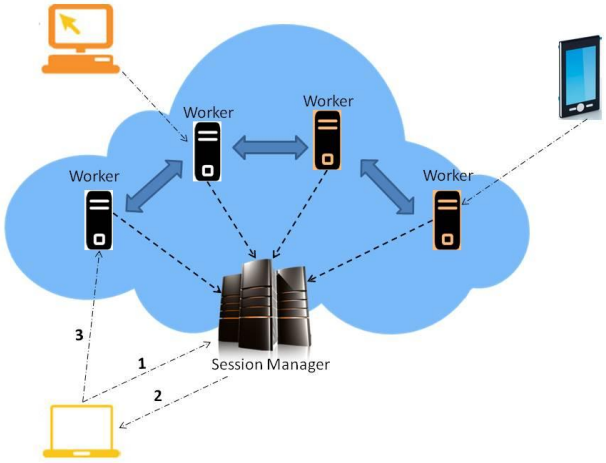

Figure 1: Architecture of CollabDraw

a) Normal Mode A - Client starting a new session

A client initiates a connection by sending a request to the Session Manager, marked as step 1 in Figure 1. Since Session Manager maintains a list of active Worker servers along with a list of active sessions from other users, it is aware of the exact load on worker servers. The Session Manager looks for the least loaded N (configurable) worker servers similar to LARD[6] and assigns them as preferred servers for this new session. It then returns the IP of the least loaded worker server within the preferred server group for this session to the client as shown in step 2. Beyond this point, clients directly interact with worker servers assigned to them. This is marked as step 3 in Figure 1. The overall workflow is shown in Figure 2.

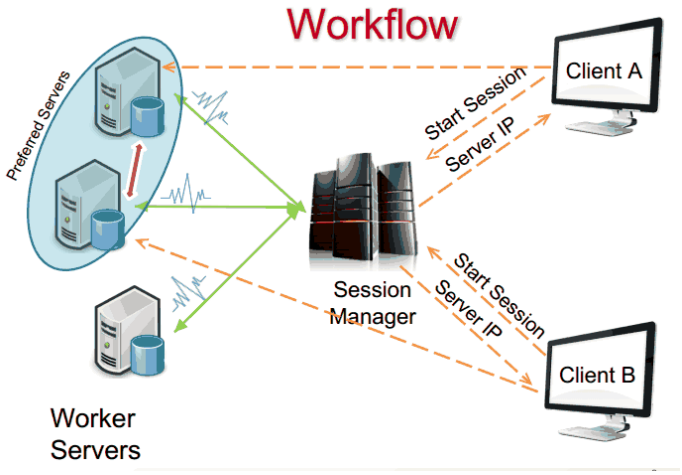

Figure 2: Workflow in normal mode. Client A start a new session and Client B joins the same session.

Within a given session, all preferred servers are kept strongly consistent while other servers maintain eventual consistency. Any changes to the board will be broadcasted to all the servers in the preferred servers list, which in turn would update the clients using full-duplex connection established via WebSocket. To ensure eventual consistency across all servers, we also invoke pair-wise synchronization similar to anti-entropy protocol in Bayou [7] to propagate changes to rest of the servers. Maintaining different consistency levels across different worker servers helps reduce network overhead.

Session Manager is responsible for pruning the preferred server list of all the sessions, ensuring that all dead servers are replaced by healthy one's periodically. A noteworthy point is that Session Manager is contacted only to initiate a session or to join an existing session, all other tasks are subsequently handled by a farm of worker servers and hence Session Manager server does not act as a bottleneck in the system. Moreover, we can also replicate Session Manager server for higher reliability.

b) Normal Mode B - Client joining an existing session

Client sends a connection request to the Session Manager. The Session Manager identifies this session as existing session and consults it's database to find currently least loaded server within the preferred servers for this session and returns the chosen IP to the client. Other steps are identical to those defined above.

c) Worker Server Failure

When an active worker goes down, connections between the server and all its connected clients are severed. Clients identify severed connection and revert back to Session Manager requesting for a new worker server. Session Manager also knows about the failed worker server since it must have lost the heartbeat with the dead worker server. It then prunes the preferred server list for this session by removing the dead server and replacing it with another live server. Then it chooses the least loaded server within the preferred server list and returns the IP to the client. Client can now re-initiate the connection with the new worker server. Since we already keep all worker servers eventually consistent, no data movement is necessitated by this new allotment. This scenario is described in Figure 3.

d) Disconnected Mode

In our system, clients are capable of operating in disconnected mode, similar to Coda [9]. Disconnected mode could trigger when the network connection at

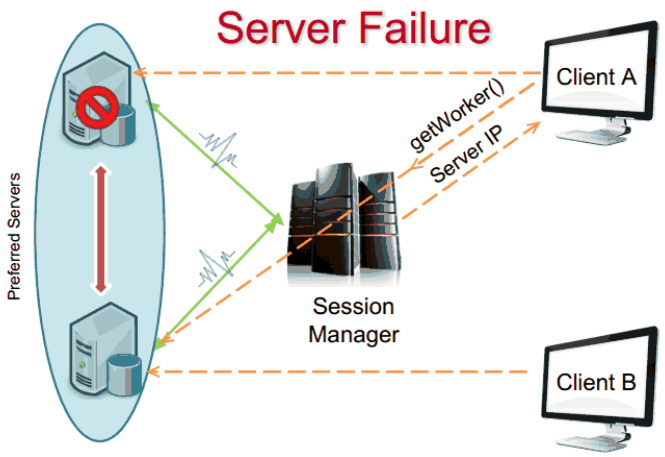

Figure 3: Worker Server Failure. Clients migrate to new worker server via session manager.

client ends fails. In such scenario, clients will not be able to interact with either Session Manager or the Worker servers. In rare scenarios, disconnected mode can also be triggered when all worker servers within preferred server group for a given session are unreachable. In such situations, client requests to session manager to assign new worker servers are replied with a disconnected flag. This signals to the clients that they have to operate in disconnected mode. This is shown in Figure 4.

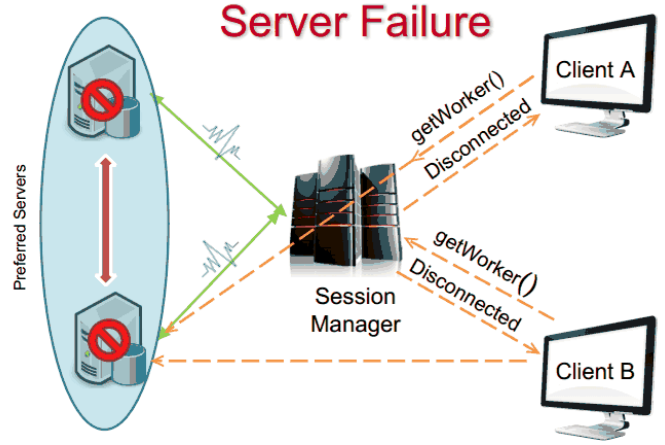

Figure 4: Disconnected Mode. When no worker servers are available, clients fall back to disconnected mode.

While operating in disconnected mode, clients store the drawing board data locally using HTML5 local storage. HTML5 local storage permits about 5MB of persistent data storage. The clients periodically ping the session manager to check if the connectivity is restored or any worker server is alive. As soon as the connectivity is restored or few worker servers come alive, clients join the new worker servers and reconcile their local data with other clients and worker server.

3. EVALUATION

We performed evaluation of CollabDraw on different grounds. We evaluated the performance of our system in terms of time taken by the system to serve client's request and also how much time does the system take to synchronize between the preferred servers (strong consistency convergence time) and non-preferred servers (eventual consistency convergence time). We studied the fault tolerance of our system in the situation where one of our worker server fails and then restarts.

3.1 Experimental Setup

To conduct our experiments, we used three Amazon EC2 [10] instances, each with 600MB RAM and 8GB disk storage running Ubuntu 12.04LTS Linux operating system. Two of these machines worked as Worker Servers, whereas the third one played the role of Session Manager.

3.2 Results

We describe the results from our experiments below.

3.2.1 Strong Consistency Convergence Time

As mentioned in Section 2.2, CollabDraw provides strong consistency by replicating data between preferred servers. So, time taken to make our system strongly consistent is the time taken to push the changes made by a client to all the clients connected to the preferred servers for a particular session. Figure 5 shows that the strong consistency convergence time varied across the number of preferred servers.

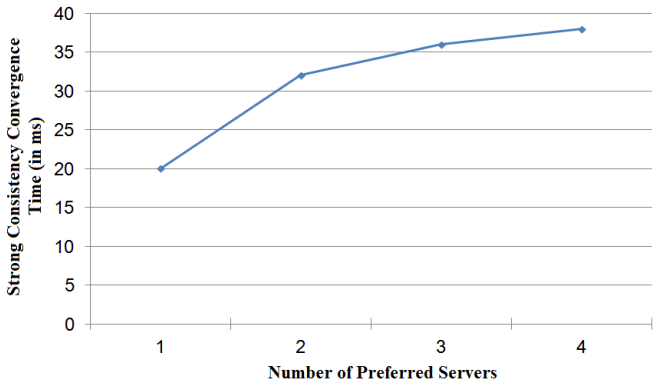

Figure 5: Strong Consistency Convergence Time vs. Number of Preferred Servers.

From Figure 5, we can see that the convergence time increases with the number of preferred servers for a particular session. But, the convergence time per preferred server (gradient in the graph) decreases, which implies that our system is highly scalable and can be easily extended with a large number of preferred servers for a particular session

3.2.2. Eventual Consistency Convergence Time

In this experiment, we evaluate the time taken to achieve eventual consistency by our system. Eventual consistency convergence time is the time taken by our system to push the changes made by a client to all other clients connected to non-preferred servers for a particular session, using anti-entropy protocol [7]. To get more accurate results, we set the number of preferred servers to zero, and calculated the average time took by our system to have all clients in a consistent state. Figure 6 shows the results from our experiments that we ran across different number of total servers in the system, with no servers acting as preferred servers.

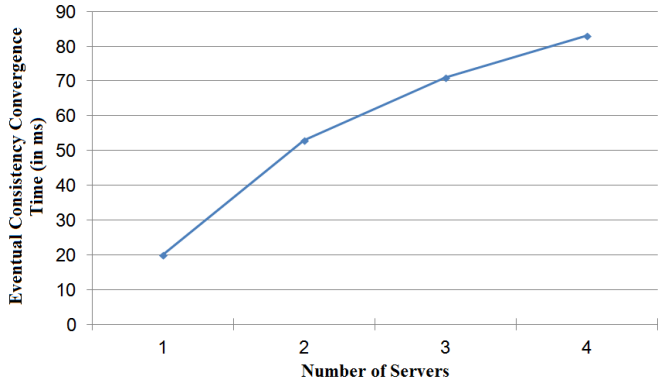

Figure 6: Eventual Consistency Convergence Time vs. Total Number of Servers.

3.2.3 Mean Time to Recovery

Since CollabDraw is an application built over a high fault tolerance system, it is important to measure how much time it takes to recover from a worker server failure. If this would have been a single server system, mean time to recovery (MTTR) would imply the time taken by the server to restart. In our case, MTTR means the time taken by our system to redirect the client to a new server in the event of failure of the worker server, which the client was initially connected to. It is important to note that MTTR does not depend on the number of servers in the system. Table 1 shows the results comparing MTTR between single server system and our distributed system. We can see that our system performs over 9 times faster than the conventional single server systems.

Type of System	MTTR (seconds)
Single Server System	18.37
CollabDraw	1.96

Table 1: Comparison of MTTR between Single Server System and CollabDraw

4. RELATED WORK

In our system, we want to provide a consistent view of the drawing board to the clients, irrespective of the server they connect to. Thus, as explained in Section 2.2, we would like to ensure strong consistency among the servers (preferred list of servers) to which client may connect for that particular session. But, we don't really need to have strong consistency among all the servers, and thus, eventual consistency guarantees should be sufficient for the servers which are not part of preferred list. In past, different solutions have been proposed for propagating updates in a replicated system ensuring eventual consistency, one of which is anti-entropy protocol used in Bayou [7].

There are many commercial solutions which share the similar idea of collaborative editing as CollabDraw, for e.g., Google Docs [2], CollabEdit [11] and Stypi [12]. But the problem with these collaborative text editors is that they don't support disconnected mode, i.e., a client cannot write text if it disconnects from the network or in the absence of any server in the system. The idea of clients working even in offline mode has been inspired from Coda [8], where it allows clients to make changes to any file while being offline, via disconnected operations [9]. To allow clients to make drawings on a board even in the absence of servers, CollabDraw leverages HTML5 local storage [3] and stores the points drawn by the clients locally. It regularly checks with Session Manager about the presence of any worker server it can connect to. Whenever any worker server, even if non-preferred server for that particular session, comes up, all the clients get in sync again.

To ensure locality of requests made to the servers and keeping them load balanced, we have used an approach similar to LARD [6]. Here, for each new session requests from clients, Session Manager dynamically identify a (limited) list of preferred servers, which can now handle subsequent requests for this session. A part of our system design is also inspired from Google File System [4]. Like master server in GFS, Session Manager is a lightweight meta-data store, responsible of keeping track of servers and sessions in the system. Client queries Session Manager regarding the information of servers they should connect to. And, after Session Manager returns information about an appropriate server to a client, the client then directly connects to that worker server bypassing Session Manager.

5. FUTURE WORK

In this course project, our goal was to build a prototype distributed collaborative system and gain experience of building a truly distributed application for low workloads. We believe that with some simple design changes and relevant optimizations, our system would scale to real-world heavy workloads. Following are some of the optimizations, we plan to integrate in our prototype application in the near future:

- Currently our application captures every single brush event on the canvas. This creates a lot of events to be distributed across the network to other clients. We can reduce the number of such events by storing every Nth point instead of every single point and then fitting a line between these points using well-established techniques like Bezier curves [13]. This would lead to a significant performance improvement and decrease the network lag across sessions spanning multiple servers.

- Our current design sends every single brush event to the server. We can batch together a couple of points together and send it together to the worker servers. This would reduce the network latency. A human eye is not able to distinguish events that take less than 200ms to finish [14]. Considering a single brush event takes roughly 30ms, we can thus batch together 6 points without affecting the interactivity of the application.

- The onus of data replication which currently is being handled by our backend application can be transferred to well-established industry techniques like MySQL replication [15]. This would reduce some of the complexity of our distributed application and would lead to simpler overall architecture.

6. CONCLUSION

Our course project has given us keen insight into the design of a truly distributed system, capable of handling real-world workloads. We were able to implement and appreciate many of the distributed system design patterns

that we studied throughout the course. Our design achieves high availability by allowing any client to connect to any worker server. Since the state of drawing sessions is replicated, our application provides fault tolerance against server failures. In the event of network failure at client end, we also provide disconnected mode of operation, in which a client can continue to work offline using local storage and synchronize with others whenever the network connection is restored. The empirical results indicate that our distributed application addresses an important challenge of fault-tolerant real-time collaboration relatively better than single-server systems.

REFERENCES

[1] Terry, Doug. *Replicated data consistency explained through baseball*. Technical Report MSR-TR-2011-137, Microsoft Research, 2011.

[2] Google Docs. https://docs.google.com/

[3] Casario, Marco, et al. "HTML5 Local Storage." *HTML5 Solutions: Essential Techniques for HTML5 Developers*. Apress, 2011. 281-303.

[4] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS Operating Systems Review*. Vol. 37. No. 5. ACM, 2003.

[5] Fette, Ian, and Alexey Melnikov. "The websocket protocol." (2011).

[6] Pai, Vivek S., et al. "Locality-aware request distribution in cluster-based network servers." ACM Sigplan Notices. Vol. 33. No. 11. ACM, 1998.

[7] Petersen, Karin, et al. "Flexible update propagation for weakly consistent replication." ACM SIGOPS Operating Systems Review. Vol. 31. No. 5. ACM, 1997.

[8] Satyanarayanan, Mahadev, et al. "Coda: A highly available file system for a distributed workstation environment." *Computers, IEEE Transactions on* 39.4 (1990): 447-459.

[9] Kistler, James J., and Mahadev Satyanarayanan. "Disconnected operation in the Coda file system." *ACM Transactions on Computer Systems (TOCS)* 10.1 (1992): 3-25.

[10] Amazon EC2 Instance Types. http://aws.amazon.com/ec2/instance-types/

[11] CollabEdit. http://collabedit.com/

[12] Stypi. https://code.stypi.com/

[13] Forrest, A. Robin. "Interactive interpolation and approximation by Bézier polynomials." *The Computer Journal* 15.1 (1972): 71-79

[14] @GreWeb. http://greweb.me/2012/03/play-painter-how-ive-improved-the-30-minutes-prototyped-version/

[15] MySQL Replication. http://dev.mysql.com/doc/refman/5.0/en/replication.html