Lets3D-C: A Robust Collaborative 3D Editing Tool Utilizing Distributed Consensus Protocol

Yeoun Ui Ha
Department of Electrical/Electronic and Computer Engineering,
University of Ulsan, 93, Daehak-ro, Nam-gu, South Korea.
E-mail: gkdus23@gmail.com

Jae-Hwan Jin
Department of Electrical/Electronic and Computer Engineering,
University of Ulsan, 93, Daehak-ro, Nam-gu, South Korea.
E-mail: jjhok2000@mail.ulsan.ac.kr

*Myung-Joon Lee
Department of Electrical/Electronic and Computer Engineering,
University of Ulsan, 93, Daehak-ro, Nam-gu, South Korea.
* Corresponding Author E-mail: mjlee@ulsan.ac.kr

Abstract
As 3D modeling has been popular, collaboration in 3D modeling is also getting more attention. As one of such efforts for collaboration support, Lets3D enables a group of users to edit 3D scenes in real time with the help of collaborative middleware. In this paper, Lets3D-C, a new version of Lets3D is introduced, which utilizes the Raft distributed consensus protocol. Lets3D-C is composed of two modules: editing module for 3D editing functionalities, and consensus module for supporting the robustness of collaboration in case of node or network failure, whereas Lets3D is exposed in a single point of failure due to the use of centralized middleware.

Keywords: Lets3D, Consensus Algorithm, Raft, 3D Collaborative Editor.

Introduction
As of now, many famous 3D editors such as blender[1], FreeCAD[2] and Google’s sketchup[3] are being used popularly to support the increasing 3D modeling activities of many people. In addition, according to the increase of 3D modeling activities, collaborative environments for 3D modeling are announced as commercial Web services[4] or independent editing tools such as Lets3D[5]. Lets3D is developed as an extension of Three.js editor[6], enabling a group of users to edit 3D scenes in real time.

In this paper, Lets3D-C, we introduce a new version of the Lets3D editor named Lets3D-C based on the Raft distributed consensus protocol[7], which is recently announced as a consensus protocol which is easier to understand compared with traditional consensus protocols such as Paxos[8]. This paper is the extension of our previous work[9]. To manage user/group information, the Openfire XMPP messaging system is used in Lets3D, and the whiteboard service of C3ware[10] is also used in Lets3D to support collaborative editing functionality. But the use of these centralized middleware might cause the single point of failure, which can be overcome by state machine replication provided by distributed consensus. Lets3D-C is composed of two modules: editing module for 3D editing functionalities, and consensus module for supporting the robustness of collaboration in case of node or network failure. The consensus module is developed as an extension of Skiff[11] which is an open source JavaScript implementation of the Raft protocol.

Collaborative editing in Lets3D-C can progress when a majority of participating users are available. As in Lets3D, participating users share their editing screen through Web browsers in Lets3D-C with being aware of which objects are being edited by which users. Any editing operations are coordinated by the distributed locking mechanism supported in Lets3D-C. Unlike Lets3D, the users to participate in collaborative editing are invited by a conventional email.

Background
A. Distributed Consensus
Distributed consensus is a fundamental problem in a distributed computing system to agree on shared values required by the members participating in the distributed system, surviving the failures of some members. To solve the consensus problem, a lot of consensus algorithms are announced. Among those, Paxos[8] had been the basis of many algorithms and implementations for various consensus problems over a couple of decades, supporting state machine replication. Unfortunately, Paxos is famously difficult to understand and implement, which leads to the introduction of a new consensus algorithm Raft[7]. Raft decomposes the consensus problem into independent subproblems: leader election, log replication, and safety, which are significantly easier to understand and implement. As of now, many implementations[12] of Raft are available for various programming environment.
B. Lets3D
Lets3D[5] is a 3D collaborative editor developed as an extension of the Three.js editor [6], which is an open source JavaScript application running on Web browsers. To provide Three.js with collaborative features, Lets3D uses the whiteboard service of C3ware[10], which enables a group of users to edit and share 3D objects and 3D scenes in real time. Lets3D also relies on the Openfire XMPP messaging server to create a group for collaborative editing and to synchronize the editing screens of the members of the group through XMPP messages. Fig. 1 shows the structure of Lets3D.

![Figure 1: System Structure of Lets3D](image)

C. Node.js
Node.js[13] is a cross-platform runtime environment built on Chrome’s V8 JavaScript engine for developing server-side Web applications. Node.js enables developers to write these applications in JavaScript, supporting an event-driven architecture and non-blocking I/O API for real-time Web environment such as real-time multiplayer games or communication tools. Thanks to Node.js, Web application developers can build both client modules and server modules using the same language: JavaScript.

System Structure
To achieve consensus, a set of participating nodes builds a cluster by running the Raft protocol. Each node in the cluster acts as one of the three roles: leader, follower or candidate. The leader plays a distinguished role both in handling requests to the cluster and in log replication. When the leader fails or becomes disconnected from other nodes, the leader election process is triggered in each partition in the cluster as shown in Fig. 2.

![Figure 2: Role Transition in Raft Cluster](image)

To achieve consensus, two main tasks are performed in the cluster. The first one is leader election. For the task, the leader periodically sends a heartbeat message to followers for ascertaining network connection. If a follower receives no heartbeat message from the leader within timeout specified to the follower (each follower has different waiting time), the follower has an opportunity to become a leader as candidate.

At this moment, it starts the election process. When a candidate gets agreement from majority of servers, the candidate becomes a new leader. The second one is log replication. A log entry is committed only when the entry has been replicated on a majority of nodes in the cluster by the leader that created the entry. Uncommitted log entries could be discarded according to the cluster event such as leader change caused from network status.

Lets3D-C is composed of two modules: editing module and consensus module. The editing module runs on a Web browser as is the case with Three.js editor. To deliver editing operations arising from the associated user to the consensus module, the editing module communicates with the consensus module in the same node through WebSocket[14] as shown in Fig. 3.

![Figure 3: System Structure of Lets3D-C](image)

A. Consensus module
The consensus module in each Lets3D-C application participating in collaborative editing acts as a member of the Raft cluster associated with the collaborative editing group. The consensus module is built using node.js and Skiff[11] which is a JavaScript implementation of the Raft consensus algorithm. To coordinate accesses to shared 3D objects, the consensus module provides the functionality of concurrency control through the lock table for 3D objects. The lock table in each consensus module is also managed by the consensus algorithm with proper modification. The methods and event listeners in the consensus module are described in Table 1.
Table 1: Methods and Event Listeners for Consensus Modules

<table>
<thead>
<tr>
<th>Methods</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendEditingModule</td>
<td>Send the information of a 3D object to the editing module via WebSocket</td>
</tr>
<tr>
<td>requestLock</td>
<td>Generate a requestLock event for the leader to obtain the lock on a 3D object</td>
</tr>
<tr>
<td>leaderCall</td>
<td>Generate a leaderCall event for the leader to update the log with an operation on 3D object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Listener</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>On('requestLock')</td>
<td>Process a requestLock event as the leader, handling the lock table and adding the related entry to the log.</td>
</tr>
<tr>
<td>On('leaderCall')</td>
<td>Process a leaderCall event, the related entry to the log</td>
</tr>
<tr>
<td>On('needUpdate')</td>
<td>Process a needUpdate event which is generated when there comes an update on the log, performing different operations depending on the content of update</td>
</tr>
</tbody>
</table>

B. Editing Module
The editing module transfers the operation from user to the consensus module through WebSocket. Also, it updates its screen according to the information received from the consensus module. The editing module transfers information on editing operation which is described in Table 2. Table 3 shows the methods for collaboration with the consensus module.

Table 2: Information Transferred to Consensus Module

<table>
<thead>
<tr>
<th>Task</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing Object</td>
<td>&quot;create@Uuid@Object Data&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;modify@Uuid@Module Id@Object Data&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;delete@Uuid@Module Id&quot;</td>
</tr>
</tbody>
</table>

Table 3: Methods for Collaboration with Consensus Module

<table>
<thead>
<tr>
<th>Methods</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendRequest</td>
<td>Send information on edit operations to the consensus module</td>
</tr>
<tr>
<td>Receive</td>
<td>Receive a message from the consensus module.</td>
</tr>
<tr>
<td>Parser</td>
<td>Parse the message from the consensus module.</td>
</tr>
<tr>
<td>handleObject</td>
<td>Add, modify or delete a 3D object actually according to the parsed message</td>
</tr>
</tbody>
</table>

B. Concurrency control
To ensure the consistency of 3D objects, the consensus module of each node maintains a lock table, supporting distributed locking mechanism through consensus on the lock on each object. Basically, each modify/delete action for a 3D object consists of 3 steps: locking the object, perform the actual modify/delete action and unlocking the object. The request for the modify/delete action is delivered to the leader if it satisfies the lock condition described in Table 4, where M(A,X) and D(A,X) stand for “Modify Action on object X by User A” and “Delete Action on object X by User A”, respectively. The consensus module of the leader is supposed on shared objects, concurrency control, reliable storage resources. Lets3D-C satisfies these requirements as follows.

A. Managing operation histories on shared objects
Asynchronously generated operations on 3D objects from different users are transferred to the consensus module of each node. Thanks to the Raft consensus algorithm, the entries storing these operations are managed as log by the leader of the cluster in a consistent way, and eventually copied into the log in each node as shown in Fig.4.

![Figure 4: Process of Log Update by Request from Follower](image)

When finishing the update of a log entry by committing the entry, the leader generates a needUpdate event. If the participating consensus modules are notified with the event, each consensus module triggers the update of the related editing module. Figure 5 describes this process of update.

![Figure 5: Process of Update Screen](image)
to put the action only when the action satisfies the log condition, which means there is no modify/delete action of other users on the same object from the last committed entry to the current log entry in the leader’s log. This distributed locking mechanism ascertains the consistency of shared 3D objects in a simple and clear way. Fig. 6 explains this process more thoroughly with activity diagram.

Table 4: Definition of Lock Condition and Log Condition

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock Condition</td>
<td>(X is not locked) or (X is locked by A) for any user B different from A,</td>
</tr>
<tr>
<td></td>
<td>not (M(B, X) in Log[curIdx:comIdx]) and not (D(B, X) in Log[curIdx:comIdx])</td>
</tr>
<tr>
<td>Log Condition</td>
<td></td>
</tr>
</tbody>
</table>

D. Implemented Screen

Like as the case with Lets3D and the Three.js editor, editing screen of Lets3D-C is composed of 4 parts: Viewport, Menu bar, Side bar and Tool bar. Fig. 7 shows a typical editing screen. Viewport is used seeing 3D object. Menu bar has functions dealing with 3D objects. Side bar shows detail information and hierarchical structure of a 3D object. Tool bar supports editing functions for 3D objects such as creation and movement of objects.

C. Reliable storage resources

Resources created in a collaborative 3D editing environment such as 3D objects, operation history, and information on users participating in a collaborative work are stored in a reliable way at each node through the state machine replication ensured by the log replication mechanism of Lets3D-C.

Figure 6: Processing Modify Action

Figure 7: Lets3D-C editing screen

Using Lets3D-C, a group of users can perform 3D editing collaboratively, sharing their editing screens as follows:

1. To form a collaborative editing group, a key user starts the Lets3D editor with its consensus module, sending the URL(IP, port number) of the module to the other planned users through invitation email.

2. Each invited user connects to the consensus module of the key user using the URL. When all members have connected the module, the planned collaborative group is formed. Now a Raft cluster is composed of the consensus modules of all members.

3. Sharing the same editing screen as shown in Fig.8, each member of the collaborative group perform edit operations on 3D objects. Any conflicting edit operations on the same objects generated by the members are automatically coordinated through the concurrency control mechanism supported by the Raft cluster.

Figure 8: Collaborative Editing through Lets3D-C
Conclusion
Distributed consensus has been extensively studied to build robust distributed applications under the inevitable existence of network and node failures. Recently, the Raft consensus protocol is announced as a more understandable and implementable one than the previous consensus protocols. In this paper, we applied the Raft protocol to the Lets3D collaborative editing tool. As the result, Lets3D-C contains the consensus module to replicate 3D operations and data of participants according to the Raft protocol. To coordinate the accesses to shared 3D objects, it also provides the concurrency control service utilizing the Raft protocol. The utilization methods developed in Lets3D-C can be usefully applied to other large-scale collaborative editing scale under widely distributed environment.

Acknowledgement
This research was partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2013R1A1A4A01004459)

References

Authors:
Yeoun-Ui Ha is an M.S student in the School of Electrical Engineering at the University of Ulsan in Korea. She received her BS in School of Computer Engineering and Information Technology at the University of Ulsan in Korea. Her research interests include Collaborative System and middleware.

Jae-Hwan Jin is an M.S. student in the School of Electrical Engineering at the University of Ulsan in Korea. He received his B.S. from the University of Ulsan in Korea. His research interests include Cloud Storage Service, Collaborative System, and Messaging System.

Myung-Joon Lee is a professor in the School of Electrical Engineering at the University of Ulsan in Korea. He received his Ph.D. from the KAIST (Korea Advanced Institute of Science and Technology) in Korea. He has (co-)authored more than 200 research publications including numerous works on collaborative system, distributed system and biological system.