A Browser-based Distributed Framework for Content Sharing and Student Collaboration

Shikhar Vashishth, Yash Sinha and K Hari Babu
Dept. of Computer Science and Information Systems,
Birla Institute of Technology and Science, Pilani, India.

***Abstract*—The utilization of the networks in education system has become increasingly widespread in recent years. WebRTC has been one of the hottest topics recently when it comes to Web technologies for distributed systems as it enables peer-to-peer (P2P) connectivity between machines with higher reliability and better scalability without the overhead of resource management.**

In this paper, we propose a browser based, asynchronous framework of a P2P network using distributed, lookup protocol (Chord), NodeJS and RTCDataChannel; which is scalable and lightweight. The design combines the advantages of P2P networks for better and sophisticated education delivery. The framework will facilitate students to share course content and discuss with fellow students without requiring any centralized infrastructure support.

***Index Terms*—Chord, Peer-to-peer, NodeJS, WebRTC, RTCDataChannel, Decentralized Distributed Systems**

I. Introduction

Several technologies like computing, database systems, networking and web technology play an important role in the field of education technology. In the field of networking, several advancements has led to boom in online education. Beginning with traditional centralized systems, the trend has moved on to decentralized distributed systems. The use of such distributed systems promises several advantages, such as higher reliability and better scalability.

The peer-to-peer technology is superior to traditional client-server model because it doesn't require setting up a third party server which makes it much more economical and practical in many situations. The peer-to-peer architecture offers the promise of harnessing the vast number of nodes connected to the network. Other significant features are redundant storage, permanence, efficient data location, anonymity, search, authentication, and hierarchical naming. Thus, this technology can be highly beneficial in supporting teachers and students to collaborate and share information together within a community.

The contribution of this paper is design and successful implementation of a browser based, asynchronous framework of a P2P network using distributed, lookup protocol (Chord [1]), NodeJS and RTCDataChannel. We have chosen Chord as it provides efficient lookup in a dynamic peer-to-peer system with frequent node arrivals and departures. Additional features can be layered on the top of the framework based on the practical usage to gain robustness and scalability. We have also illustrated some of the cases wherein this framework can be beneficial.

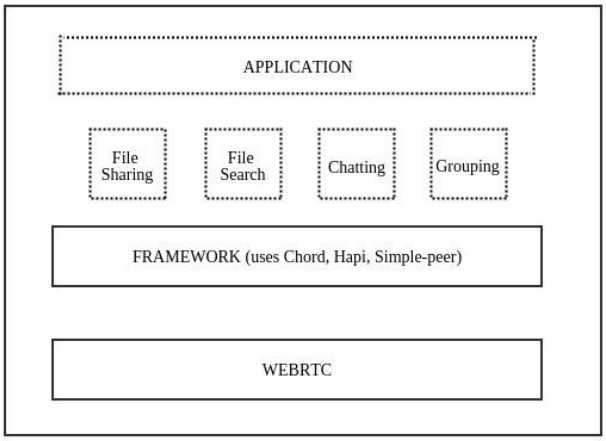

Fig. 1: Layers in the designed framework

Current frameworks are not very suitable for web applications because the web browsers are generally single threaded. We have enlisted the reasons in a subsequent section and advocated the need for a framework based on event driven programming paradigm along with asynchronous calls for a web browser. We have also improved the fault tolerance by making dependence on the bootstrap server weaker, which is a point of network failure. Therefore, this framework will help P2P technology to enter in more areas.

We have explained at length, the implementation details of the framework in the rest of the paper. Section II compares this asynchronous implementation to related work. Section III presents the reasons for this framework. Section IV of the paper describes the base Chord protocol, notations used, the bootstrap server, join operations of new peers, data structures used and handling of RTCDataChannel connections between peers. Section V presents the implementation details of the framework. Section VI highlights certain ways in which this work can be used in to enhance the learning experience in a community. Finally we conclude in Section VII.

II. Related Work

CHEWBACCA (CHord, Enhanced With Basic Algorithm Corrections and Concurrent Activation) [2] is a P2P network framework in Java using sockets based messaging. Although, this framework uses synchronous calls, it is not suitable for event driven, non-blocking systems like web-browsers.

WebRTC-Chord is an asynchronous implementation of Chord protocol on Node Package Manager [3]. Although made for web browser, WebRTC-Chord is not as efficient because it doesn't keep bootstrapping server lightweight. The server stores all information about the peers and helps them to update their entries as the network changes dynamically

over time. Thus it is not as scalable as this framework. This framework reduces the involvement of bootstrap server only up to the initial phase of joining network for a peer. Establishing connection with other peers later doesn't involve bootstrap server at all.

joonion-jchord [4] is simple implementation of Chord protocol written in Java is implemented for single virtual machine not for multiple systems.

III. Need for Framework

A. Web browsers incompatible with synchronous calls

Current frameworks of Chord are in C++ and Java and they use synchronous calls and the socket programming APIs. Processes, facilitated by the kernel, can wait for the synchronous calls to return, by getting suspended. Also threads can be used for foreground and background tasks.

But synchronous calls in a web browser framework are discouraged for three reasons. Firstly, browsers are built using event driven programming paradigm, where asynchronous calls are suitable. Secondly, spinning or busy waiting locks the browser and other processes start crawling. [5] Thirdly, most browsers are single threaded and don't do anything on screen while Javascript code is running. [6] This hampers user experience badly. So, a need for an asynchronous framework was felt, designed and implemented.

B. Weakly Connected Bootstrap Server

There are frameworks, wherein the bootstrap server is actively involved to connect new peers and stabilize the network. The server stores all information about the peers and helps them to update their entries as the network changes dynamically over time. [3] Thus, there is a point of failure in the network, if the bootstrap server fails, new peers cannot join and the network cannot be stabilized. New RTCDataChannel connections are dependent on server to facilitate handshakes between the new peer and the network.

This framework, however, makes the dependence on the bootstrap server weaker by facilitating handshakes between new peer and the network via the boot peer and other peers in the network. The bootstrap server is only involved in assigning a new, unique id to a new peer and connecting it with a peer from the network chosen randomly. This also balances the load on peers to facilitate handshakes. But this requires more messages to be sent, forwarded and accepted between peers in an asynchronous way.

We have modified the function calls of the Chord protocol [1], to accommodate these changes.

IV. Framework Design

A. Base Chord Protocol

Chord is protocol for peer-to-peer distributed hash table. It assigns each peer an m-bit identifier using base hash function such as SHA-1. Each peer maintains small amount of routing information that makes chord scalable by avoiding every peer to know about every other peer. In N-peer network each peer maintains information about only O(log N) other peers and a lookup requires O(log N) messages.

Nodes and keys are arranged in an identifier circle that has at most 2^m peers, ranging from 0 to 2^m-1.

B. Notations Used

TABLE I: NOTATIONS AND DEFINITIONS USED.

Notation	Definition
peer	a node in network
boot Peer	the first peer (in network) with which a connection is established for a new peer
predecessor	predecessor of peer in network
successor	successor of peer in network
peerId	unique Identifier of a peer
destPeerId	identifier of destination peer
succPred	predecessor of peer's successor
N	current peer
B	boot peer of current peer
msgId	unique identifier for every remote message
"next state"	the state of joinNetwork function to be called when remote call is complete
callBackFunc	function to be called when remote call is complete
signal	signaling data of peer which is required during handshake to establish connection with peer
path	stores the information about the route which message took to reach final destination
acceptSignal()	make peer to accept signaling data of other peer to establish connection with it
exec	executes the desired operation
finger	finger table entry
responseTable .new()	generates a new message Id and allocates space in memory for response of remote call
responseTable .get(msgId)	stores the response of remote call of given id
finger.start	for kth finger, n + 2
"update entry"	the operation which updates the finger table entry when the remote call is complete

C. Message Format

The following message format has been used to send information to other peers:

TABLE II: MESSAGE FORMAT

Field	Description
srcPeerId	Identifier of sending peer
type	Message type
msgId	Unique identifier for message
path	Contains the path information which message took
callBackFunc	Function to be called when request is complete
data	Response data
signal	Signaling information of sending peer

D. The Bootstrap Server

It allows new peer to join network by establishing its connection with one of the peer which is already in network. For a new peer, that peer becomes its boot peer whom it contacts for joining the network and establishing connection with other peers. Bootstrap server has no other role than to help peer in making connection with its boot peer, this helps to keep server light weight and makes the framework more scalable.

It is implemented using hapi framework. [7]

```
peerJoin():
    generate_PeerId();
    send_address_boot_peer();
forwardOffer()
forwardReply()
peerLeave()
```

E. Peer Joins

Whenever a new peer joins a network, these invariants should be kept:

1) It must have connection with at least one peer in network (boot peer).

2) Each peer's successor points to its immediate successor correctly.
3) Each key is stored in successor(k).
4) Each peer's finger table should be correct. If finger table entries are not correct than query cost becomes O(n) instead of O(log(n)).

Following steps are involved for making a new peer join network:

1) Initialize peer - assign it an identifier, initialize finger table etc.
2) Find the successor of new peer by querying bootstrap peer.
3) Find the predecessor of new peer by asking peer's successor to send information about its predecessor.
4) Notify other peers to update their successor, predecessors to maintain correctness of the framework.
5) Calling *fixFinger* operation after regular interval to keep finger table entries up to date. Fix Finger operation involves calling *findSuccessor* operation for each finger table entry taking O(mlog(n)) time to update entire finger table.

F. Connection objects: the special case

Chord protocol specification assumes that a peer can connect to a peer as long as it has its peer id. But in the real world a connection needs to be established so that messages can be sent and received. We have used WebRTC's RTCDataChannel API to connect peers. The npm module Simple Peer [8] has been used to connect two machines. The initial handshake to connect to boot peer is facilitated by the bootstrap server, whereas subsequent handshakes to connect to other peers are handled by boot peer and other peers (as and when they get connected).

The signals is sent from the source peer in the *findSuccessor* and *findPredecessor* functions. The function calls hop over peers till a peer finally resolves the answer (some peer) of the given function call. This peer forwards the function call to the *answer peer*, which accepts the signal of the source peer. It then sends its own signal to the source peer, via the same path by which the function call arrived at it. Thus, the handshake is complete.

V. Implementation Details

A. Data structures Used

Following tables were used to store various information about the state of the network and connection objects.

1) Response Table:

Associative data structure maps message id to response received on making a remote call.

2) Waiting Connection Table:

Stores information about connection with which peer is currently in process of making connection.

3) Connected Connection Table:

Stores connection information about peer with which the peer already has established connection.

4) Finger Table:

Contains m entries of peers with which peer has connection. It helps to avoid linear search. The i^{th} entry of peer n will contain successor($(n + 2^{i-1}) \mod 2^m$). The first entry of finger table is the peer's immediate successor. Using finger table each query operation in network can be completed in O(log(n)).

B. Pseudo Code

1) joinNetwork

Synchronous:

```
joinNetwork():
    n.predecessor = null;
    n.successor = b.findSuccessor(n.peerId);
    n.succPred = n.successor.getPredecessor();
    n.stabilize();
    if n.succPred == null
        n.successor.stabilize();
    else
        n.succPred.stabilize();
```

Asynchronous:

This is asynchronous implementation of *joinNetwork* operation which allows it to run in non-blocking, asynchronous frameworks such as NodeJS. This works same as synchronous version but is based on event driven programming paradigm.

```
joinNetwork(state, data):
  state 0:
    n.predecessor = null;
    msgId = responseTable.new();
    n.initFindSuccessor(b.peerId, n.peerId, msgId, "next state")
  state 1:
    n.successor = responseTable.get(msgId);
    msgId = responseTable.new();
    n.initFindPredecessor(n.successor, msgId, "next state")
  state 2:
    n.succPred = responseTable.get(msgId);
    if n.succPred != null AND n.succPred ∈ (n.peerId, n.successor]
      n.successor = n.succPred;
    n.notifyPredecessor(n.successor, n.peerId, msgId, "next state");
  state 3:
    if n.succPred == null
      n.stabilize(n.successor, msgId, "next state");
    else
      n.stabilize(n.succPred, msgId, "next state");
```

2) initFindSuccessor

A peer executes *initFindSuccessor* to know the immediate successor of desired *id*. It generates signalling data of peer attaches it with a call to *findSuccessor*. It also tells the *callBackFunc* operation to be called once request is completed.

```
initFindSuccessor(destPeerId, id, msgId, callBackFunc):
  signal = generateSignal(initiator = true)
  findSucessor(destPeerId, id, path, msgId, callBackFunc, signal)
```

3) findSuccessor

findSuccesor operation asks *destPeer* to tell successor of desired *id*. *destPeer* can contact other peers if it doesn't know the response to the query. When desired peer is found then signalling data of peer n is passed to that peer to allow it to form connection with it. The information about the order in which peers are contacted is stored in *path* variable of the call. It is used for returning final result and signalling data of successor of *id* to peer n. Once response is reaches the peer *callBackFunc* is invoked.

```
findSucessor(destPeerId, id, path, msgId, callBackFunc):
  if destPeerId == n.peerId
    if n.peerId == n.successor
      exec(callBackFunc);
    else if id ∈ (n.peerId, n.successor]
      n.successor.acceptSignal()
    else
      if n.closestPredecedingFinger(id) == n.peerId
        n.findSucessor(n.successor, id, path, msgId, callBackFunc, signal);
      else
        n.findSucessor(n.closestPredecedingFinger(id), id, path, msgId, callBackFunc);
  else
    path.append(n.peerId)
    destPeerId.findSucessor(destPeerId, id, path, msgId, callBackFunc, signal);
```

4) initFindPredecessor

Peer n executes *initFindPredecessor* to know the predecessor of desired peer and to form a connection with that peer. It generates signalling data of peer and attaches it with *findPredecessor* call. It also attaches *callBackFunc* that will be invoked when request is complete.

```
initFindPredecessor(destPeerId, path, msgId, callBackFunc):
  signal = generateSignal(initiator = true)
  findPredecessor(destPeerId, path, msgId, callBackFunc, signal)
```

5) findPredecessor

findPredecessor operation gives the immediate predecessor of given peer. When the predecessor of the desired peer is found, signalling data of peer is given to it which allows it to form connection with the calling peer.

```
findPredecessor(destPeerId, path, msgId, callBackFunc, signal):
  if destPeerId == n.peerId
    if n.peerId == n.successor
      exec(callBackFunc)
    else
      n.predecessor.acceptSignal()
  else
    destPeerId.findPredecessor(destPeerId, path, msgId, callBackFunc, signal)
```

6) notifyPredecessor

When peer calls *notifyPredecessor* it asks *destPeer* to update the value of its predecessor to the given value of predecessor and *callBackFunc* is called when this update request is complete.

```
notifyPredecessor(
  destPeerId, predecessor, path, msgId, callbackFunc):
  if destPeerId == n.peerId
    n.predecessor = predecessor
    exec(callBackFunc)
  else
    destPeerId.notifyPredecessor(destPeerId, predecessor, path, msgId, callBackFunc)
```

7) stabilize

When a new peer joins the network then stabilize operation is called on given peer, its successor and predecessor to allow them to correct their predecessor and successor entries. This operation is essential for the framework to maintain its correctness.

```
stabilize(destPeerId, path, msgId, callBackFunc):
  if destPeerId == n.peerId
    n.succPred = n.initFindPredecessor(n.successor, msgId, "next state")
    if n.succPred == null AND n.succPred ∈ (n.peerId, n.successor]
      n.successor = n.succPred
    n.notifyPredecessor(n.successor, n.peerId, msgId, "next state")

  else
    destPeerId.stabilize(destPeerId, path, msgId, callBackFunc)
```

8) fixFinger

This operation is invoked by peers at regular intervals to update the entries of their finger tables which allows network to answer query in O(log(n)). If entries in finger table are not correct then query is resolved through successors which takes O(n) time to respond to a query.

```
fixFinger():
  for each finger : fingerTable
    msgId = responseTable.new();
    n.initFindSuccessor(n.peerId, finger.start, msgId, "update entry")
```

VI. NETWORKING EDUCATION

P2P technology is superior to traditional client-server model because it doesn't require setting up a third party server which makes it much more economical and practical in many situations. Setting a third party server involves expense on its security, maintenance, performance. It doesn't allow network to expand beyond a limit but such limitations are easily overcome by P2P technology because there is no central dependency on which whole network has to rely and security is also simplified because files' location are invisible to P2P peers and thus remains protected.

This framework doesn't require any kind of setup or installation on peer's system. Thus, web browsers can be part of the network, and very little expertise is required to create and join a network.

We have illustrated some of the cases wherein this framework can be beneficial.

A. Classroom Community

In classroom community, this framework can support teachers to share knowledge with students. In a classroom, students and teachers can connect to the same network and the documents which teacher wants to share with students can get distributed easily without the need to serve those documents 24hrs on a Learning Management system. This can help to reduce institution's cost of hosting its education resources online for its on-campus students. It can also help teachers to organize evaluation components within the class.

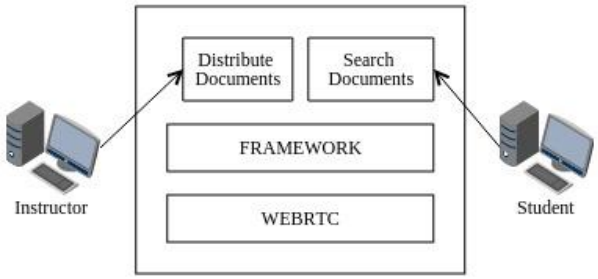

Fig. 2: Usage in a classroom community

Within students, this framework can become an easy way to share resources and education experience without any restrictions. Thus, this framework can extend support for group projects and discussions.

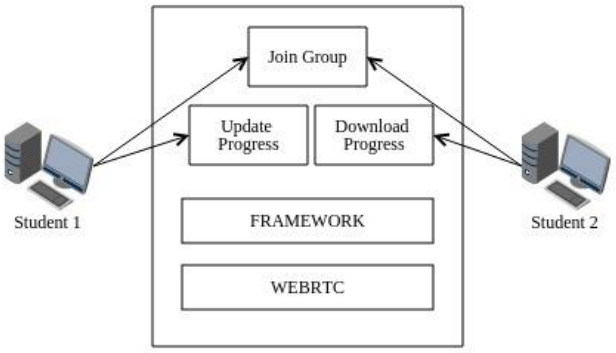

Fig. 3: Usage within project groups

Within instructor community, this framework can work as base to host many applications which can help them to work together in research. Since all instructors can get connected to the same network, one can access files along with their real time modifications by others. This can also save them from paying heavy prices to various commercial sites and plugins such as Zotero [9] which charge for sharing of research data through their servers.

B. Off-campus Distance Education

For people enrolled in distance learning program, who are limited by poor network connectivity, this framework can be aid for them. It can allow them to exchange resources from their colleges and instructor using P2P connection which will give them better performance in terms of bandwidth than traditional centralized servers.

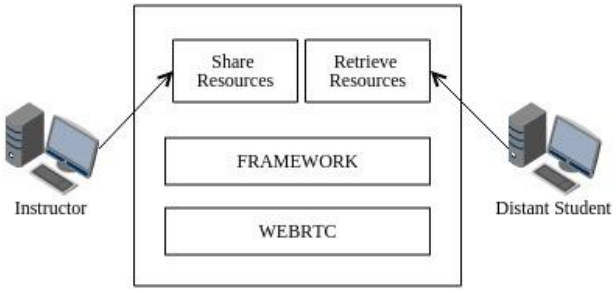

Fig. 4: Usage for Distance Education

C. Workflow management

This framework can also be used in workflow management as well. Decentralized workflow management systems are more cost-effective, scalable, and reliable and are able to deal with dynamic changes better. It also allows user to have more control over the network.

D. Peer based Information Retrieval

The network's performance in information retrieval is also very encouraging that allows it to be used at places where queries need to be responded, like in a library management system. A network of computers containing information about books can be formed and students can query them to get required information. Such setup doesn't require a big-centralized server to be installed.

VII. Conclusion

Modern day browsers are single threaded applications which cannot support synchronous calls to remote hosts. Busy waiting is not a feasible solution. We have explored the plausibility of a browser based Peer to Peer Network. This paper presents an asynchronous framework for P2P network built using distributed and lookup protocol called Chord, NodeJS and RTCDataChannel.

Benefits of P2P networks include scalability, redundant storage, permanence, efficient data location, anonymity, search, authentication, and hierarchical naming. This allows for the framework to be easily used for promoting education in multiple scenarios. The framework enables institutions and students to share course content and discuss without overhead of resource management.

The framework design includes scope for improvement. Security enhancements such as encryption can prevent a malicious peer to affect the network.. The framework can be a part of browser integration such as plugin or extension.

VIII. References

[1] Stoica, Ion, M. Robert, K. David, K. M. Frans and B. Hari, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149-160, 2001.

[2] Baker, Matthew, F. Russ, T. David and W. Adam, "Implementing a Distributed Peer to Peer File Sharing System using CHEWBACCA--CHord, Enhanced With Basic Algorithm Corrections and Concurrent Activation," 2003.

[3] D. Dias, "webrtc-chord," [Online]. Available: http://www.npmjs.com/package/webrtc-chord.

[4] "joonion-jchord," [Online]. Available: http://code.google.com/p/joonion-jchord/.

[5] J. Wolter, "Javascript Madness: The Javascript Sleep Deficiency," [Online]. Available: http://unixpapa.com/js/sleep.html.

[6] H.-G. Michna, "Sleep or wait in JavaScript | Windows Problem Solver," [Online]. Available: http://winhlp.com/node/633.

[7] "hapi.js," [Online]. Available: http://hapijs.com/.

[8] F. Aboukhadijeh, "simple-peer," [Online]. Available: http://www.npmjs.com/package/simple-peer.

[9] "Zotero," [Online]. Available: https://www.zotero.org.