Krestianstvo Luminary: Decentralized Virtual Time for Croquet architecture

Conference Paper · October 2019

DOI: 10.13140/RG.2.2.29073.79207

Nikolai Suslov

Krestianstvo Luminary: Decentralized Virtual Time for Croquet architecture

Nikolai Suslov
Fund for Supporting
Development of RT,
Vologda, Russia
SuslovNV@krestianstvo.org

ABSTRACT

Croquet architecture is known by its radical synchronization system with notion of virtual time. It allows multiple peers to run computations together within a single shared distributed environment, and it guarantees that this distributed environment will remain bit-identical for every peer. Croquet architecture is ideal for developing collaborative serverless apps and running them on decentralized networks. But a tiny stateless server named Reflector, on which Croquet heavily relays on, still prevents doing that today. Reflector server is used for heartbeat, time stamping of messages, that are passing through it, and application's state snapshotting.

This paper presents the research, that transforms the only server related Croquet's part - Reflector into the peer-to-peer application, running just on a clients. Thus, making Croquet's Virtual Time to be fully decentralized, where timestamping of messages will be doing by clients themselves. The prototype described in the paper is developed in https://LiveCoding.space - Krestianstvo SDK, based on Open Source version of Croquet - Virtual World Framework. Krestianstvo Luminary identically replaces Croquet Reflector server in flavor of using offline-first Gun DB pure distributed storage system, that combines timestamps, vector clocks, and a conflict resolution algorithm. Deploying itself on peer's Web Browsers connected through Gun DB's Daisy-chain Ad-hoc Mesh-network for swapping in different transport layers: Web Sockets, WebRTC, etc. even on AXE blockchain.

Work in Progress presented at *AGERE'19, October 22, 2019, Athens, Greece*

CCS CONCEPTS

• Theory of computation~Distributed computing models • Computing methodologies~Distributed programming languages • Software and its engineering~Virtual worlds software • Human-centered computing~Collaborative and social computing

KEYWORDS

decentralized architecture, virtual worlds software, collaborative web applications

ACM Reference format:

Nikolai Suslov. 2019. Krestianstvo Luminary: Decentralized Virtual Time for Croquet architecture. In *Proceedings of SPLASH conference (AGERE'19). ACM*

1 Virtual Time in Open Croquet architecture

Croquet introduced its own architecture, that allows anyone to create massively scaled decentralized collaborative applications [1]. Croquet radically differs from well-known p2p and client-server architectures, but unfortunately, it still has a tiny server - Reflector, for timestamping and heartbeat. All available versions of Croquet from Smalltalk to JavaScript (including the latest Croquet V by https://croquet.studio) are using such Reflector servers [2].

For those who are not familiar with Open Croquet architecture, just want to mark key principals behind it.
Croquet introduced the notion of Virtual Time: looking on objects as stream of messages, which leads to deterministic computations on every connected node in decentralized network. All computations are done on every node by themselves while interpreting an internal queue of messages, which are not replicated to the network. But these queues are synchronized by an external heartbeat messages coming from Reflector - a tiny server. Also any node's self-generated messages, which should be distributed to other nodes are marked as external. They are

explicitly routed to the Reflector, where are stamped with the Reflector's time now and are returned back to the node itself and all other nodes on the network.

Moreover, Reflector is not only used for sending heartbeat messages, stamping/reflecting external messages, but is also used for holding a list of connected clients, a list of running virtual world instances and for bootstrapping the new client connections, storing application snapshots, Figure 1.

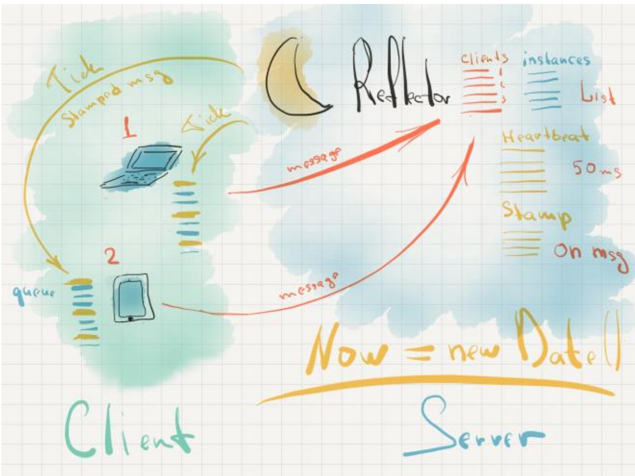

Figure 1: Croquet with Reflector server

In Croquet architecture for decentralized networks, the Reflector while being a very tiny or even being a micro service - it remains a server. It uses Web Sockets for that purposes.

Let's look how it is implemented in Virtual World Framework (VWF) - an Open Source version of Croquet [3]. Here is a function returning Time Now by a Reflector. Time is getting from a machine, hosting a Reflector server (server-side code from lib/reflector.js):

```
function GetNow( ) {
   return new Date( ).getTime( ) / 1000.0;
}
```

Then this function is used to make a timestamp for a virtual world instance:

```
return ( GetNow( ) - this.start_time ) * this.rate
```

Reflector stamps messages, that passed through it and sends them back to the clients by using Web Sockets. On a client side, VWF implements a method for dispatching the received messages (client-side code from public/vwf.js):

```
socket.on( "message", function( message ) {
        let fields = message;
        fields.time = Number( fields.time );
        fields.origin = "reflector";
        queue.insert( fields, !fields.action )
}
```

Clients use Web Sockets to send external messages back to the Reflector for timestamping:

```
var message = JSON.stringify( fields );
socket.send( message );
```

2 Introducing Decentralized Virtual Time

Now, let's look at how Krestianstvo Luminary could identically replace the Reflector server in Croquet architecture by introducing the notion of Decentralized Virtual Time, Figure 2.

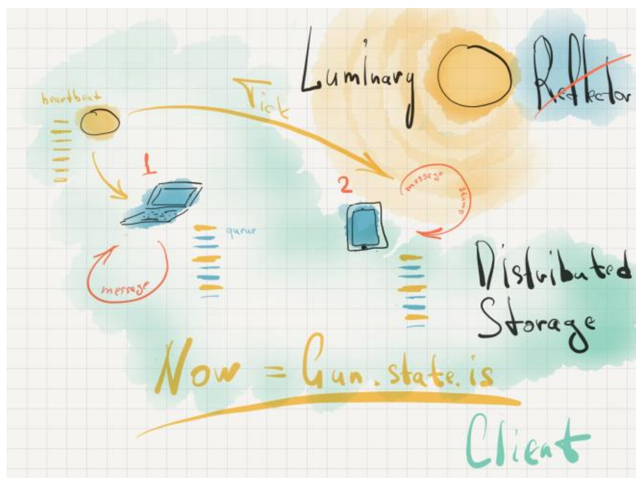

Figure 2: Croquet with Krestianstvo Luminary

Krestianstvo Luminary is replacing Reflector server in flavor of using offline-first Gun DB pure distributed storage system. That allows instead of 'Reflecting' messages with centralized Croquet's time now, depending on Server's Machine time, to 'Shining' time on every connected node using Gun's Hypothetical Amnesia Machine, running on decentralized peer-to-peer Web.

In Krestianstvo Luminary clients are never forced to use Web Sockets directly from the application itself for sending or receiving messages. Instead Gun DB responds for that functionality internally. All operations which previously relay on Web Socket connection are replaced with subscribing to updates happening on a Gun DB nodes and properties, accordingly to Functional Reactive programming. So, worlds instances, clients are becoming just a Gun DB nodes, that are available to all connected peers. Finally, the required by Croquet a Reflector’s application logic is moving from the server to the peers. Now every client on any moment of time could get actual information about world’s instance, it is connected to, amount of clients on that instance, etc. Doing that just by subscribing to a corresponding node on Gun DB.

Instead of using server machine’s time,

```
new Date().getTime()
```

Krestianstvo Luminary uses the state from Gun’s Hypothetical Amnesia Machine:

```
Gun.state.is ( node, property )
```

For calculation of the machine state, Gun DB HAM combines timestamps, vector clocks, and a conflict resolution algorithm. So, every written property on a Gun’s node stamped with HAM. This state is identical for all peers. That means, that we could get a state just on any client. Taking into consideration, that Gun DB guarantees that, every change on every node or property will be delivered in right order to all peers [4].

Let’s see how we could make a heartbeat node and subscribe peers to its updates. Here is the code for creating a simple heartbeat for VWF:

```
Gun.chain.heartbeat = function (time, rate) {
        // our gun instance
        var gun = this;
        gun.put({
          'start_time': time,
          'rate': 1
        }).once(function (res) {
          // function to start the timer
          setInterval(function () {
            let message = {
              parameters: [],
              time: 'tick'
            };
            gun.get('tick').put(JSON.stringify(message));
          }, 50);
        })
      return gun;
}
```

Client, which start firstly or create a new virtual world instance could create a heartbeat node for that instance and run a metronome (that part could be run on Gun DB instance somewhere on network for anytime availability):

```
let instance = _LCSDB.get(vwf.namespace_); //
instance.get('heartbeat').put({ tick: "{}" }).heartbeat(0, 1);
```

So, every 50 ms, this client will writes to the property ‘**tick**’ the message content, thus changing it, so Gun HAM will move forward the state for this property, stamping it with the new unique value, from which the Croquet time will be calculated later. The start time will be the state value of HAM at ‘**start_time**’ property of heartbeat node. Please notice, that actual Croquet timestamp is not calculated here, as it was in Reflector server. The timestamp used for the Croquet internal queue of messages will be calculated on reading of the ‘**tick**’ by the VWF client in its main application.

Here is the simplified core version of dispatching ‘**tick**’ on VWF client main app, just to get the idea (full code on public/vwf.js):

```
instance.get('heartbeat').on(function (res) {
        let fields = self.stamp(res);
        queue.insert(fields, !fields.action);
}

this.stamp = function(source) {
  let message = JSON.parse(source.tick);

  message.state = Gun.state.is(source, 'tick');
  message.start_time = Gun.state.is(source, 'start_time');
  message.rate = source.rate;

  let time = (message.state - message.start_time)*message.rate/1000;

  message.time = Number( time );
  message.origin = “reflector";
  return message
  }
```

The main point here is the calculation of Croquet time using Gun’s HAM state. Time for updating tick is getting from the HAM state on ‘**tick**’ property. The start time of the world instance heartbeat is getting from the HAM state stamp on ‘**start_time**’ property. These stamps are identical for all connected peers, that is guaranteed by Gun DB. Then the actual Croquet time is calculated. All calculations are done by every peer by themselves, no server involved in.

So, all peers will calculate exactly the same Croquet time on getting an update from Gun DB, regardless of the time when they get this update (network delays, etc.).

Sending external messages will be as simple as just writing the message by any peer to a world instance heartbeat with a new message's content:

```
instance.get('heartbeat').get('tick').put(JSON.stringify(newMsg));
```

Being subscribed to the '**heartbeat**' node, all connected peers and a peer itself will get that message, stamped with an identical Croquet virtual time.

3 Conclusions

Table 1: Comparison table of Virtual Time and Decentralized Virtual Time implementation internals

	Croquet Reflector	Krestianstvo Luminary
Architecture:	Client-Server	Peer-to-Peer
Croquet time stamp:	on server	on peer
Time now is:	server machine's time	GunDB HAM state
source code	new Date( ).getTime( )	Gun.state.is (n, p)
Heartbeat messages:	by server	by selected peer
Reflector app logic:	on server	on peer
Hosting:	dedicated server	peer's Web Browsers
Security:	by server	by P2P identities

Let's summarize, what Krestianstvo Luminary brings to Croquet architecture in **Table** 1.

1. Reflector server is no longer required for running virtual worlds (any existed Gun DB instance on a network fits, could know nothing about Croquet and clients)
2. Clients, world instances, connecting logic are hold by a decentralized DB
3. Timestamping of the messages are doing by clients themselves using Gun's HAM
4. One dedicated peer is selected to produce a metronome empty messages for moving time forward (could be anywhere and movable)

Gun DB storage system allows to deploy Krestianstvo Luminary and Croquet applications just on peer's Web Browsers connected through Daisy-chain Ad-hoc Mesh-network suited for swapping in different transport layers: Web Sockets, WebRTC, etc. That makes Croquet architecture compatible with novel Decentralized Web standards and technologies.

For building the prototype of Krestianstvo Luminary, the open source code of https://LiveCoding.space was used. It is a collaborative, live programming environment based on tight integration of A-Frame, Croquet (VWF), Cell.js, Gun DB storage system and Ohm language [5]. It provides all-in-one solution for development of collaborative applications for Web XR. Besides replacing Reflector server in LiveCoding.space prototype, Krestiasntvo Luminary has shown a lot of other perspectives. So, all advantages that Gun DB provides, could be applicable inside an applications, that relays on Croquet Architecture. One of the scenarios could be the use of Gun's HAM Time Graph. That will allow to store and retrieve the history of messages for recording and replaying later. Using SEA Security, Encryption, & Authorization library, will allow to create a highly secure instance's heartbeats using peer-to-peer identifies and being deployed anywhere, anytime available on AXE blockchain.

ACKNOWLEDGMENTS

I would like to express thanks for the valuable insights that Victor Suslov, Sergey Serkov and to all others, who have helped in the realization of the prototype, described in this paper.

REFERENCES

[1] D. A. Smith, A. Kay, A. Raab and D. P. Reed, 2003, Croquet — A Collaboration System Architecture. In Proceedings of the First Conference on Creating, Connecting, and Collaborating through Computing (C5' 03), pages 2–9. IEEE CS.

[2] D. A. Smith et al., Croquet V SDK documentation, Retrieved August 15, 2019 from https://croquet.studio

[3] N. Suslov, 2014, Virtual World Framework & OMeta: collaborative programming of distributed objects with user defined languages. The Future Programming Workshop at SPLASH 2014, Portland, Oregon, USA, video demo screencast http://vimeo.com/97713576

[4] M. Nadal, Gun DB documentation, Retrieved August 15, 2019 from https://gun.eco/docs/

[5] N. Suslov, 2019, LiveCoding.space: Towards P2P Collaborative Live Programming Environment for WebXR. In Proceedings of the Fourth International Conference on Live Coding (ICLC 2019), Medialab Prado, Madrid, Spain, http://iclc.livecodenetwork.org/2019/papers/paper133.pdf