Big-Two Project

Repository: https://github.com/klaevv/Big-Two

Tommi Koivula, Leevi Heino, Lasse Virtanen

1. The project's goal(s) and core functionality. Identifying the applications / services that can build on your project.

Our goal was to Implement a distributed multiplayer card game. The project can be expanded to include multiple game types and to form an online arcade. The project was implemented as a react-application, which uses react-liowebrtc to communicate with other nodes.

The current ruleset implements Big Two, which contains a subset of the most common poker hands.

2.The design principles (architecture, process, communication) techniques.

Clients work in a distributed fashion to carry on a card game. One node is used for node discovery. The clients communicate with each other in peer-to-peer fashion through the WebRTC API. Communication is done through simple events with type and payload.

3.The key enablers and the lessons learned during the development of the project.

Creating an application with react and using WebRTC API is convenient to get started with, but it takes a lot of fiddling to get the nodes working in unison. The programming model of React isn't very convenient for fine-grained control of events and how they happen or games in general.

4.How do you show that your system can scale to support the increased number of nodes?

Each node is independent and communicates with peer–to–peer fashion with one node being the discovery node. We can scale the amount of instances without exponential growth in network traffic. The current game would have to be modified by adding more decks/cards to support a bigger number of clients due to the nature of the card game in question.

5.How do you quantify the performance of the system and what did you do (can do) to improve the performance of the system (for instance reduce the latency or improve the throughput)?(Extra points for improvement).

Here latency is one of the key metrics since the throughput isn't as critical considering the amount of data being sent. In a peer-to-peer system like this the slowest link is usually the

bottleneck. Potential improvements here could be multihop communications where the sent event messages propagate through other peers instead of direct connections between each peer. This could let us avoid long links and slow connections in some capacity. The computing performance of the application could be improved by porting it to WebAssembly instead of using the fairly slow Javascript engine

6.What functionalities does your system provide? For instance, naming and node discovery, consistency and synchronization, fault tolerance and recovery, etc? For instance, fault tolerance and consensus when a node goes down.

The system provides naming, node discovery, synchronization and fault tolerance.