Vrije Universiteit Brussel

Proposal bachelor thesis

Title: Automagically Shared and Offline Available Data for the Web

Promotor: Wolfgang De Meuter

Contact: Tim Coppieters

Includes preparation course: Yes

Context

Today, people commonly collaborate with each other across the world using collaborative applications such as wiki's, documents, social networks, etc. However, when looking at all of these applications, it is perhaps surprising to find that almost none of them can be used when the user is disconnected from the network. And yet, the ability to collaborate while being disconnected ranges from being useful (e.g. working on an airplane) to being critical (e.g. in disaster-relief scenarios where network connectivity is sparse).

The primary reason why most distributed applications do not support disconnected clients is because supporting such functionality requires deep changes to the architecture of the application. The application needs to transition from a centralized notion of application state to a distributed, possibly replicated, application state, requiring a consistency model to keep the distributed state consistent. This requires a lot of extra code to: distribute the shared data to all clients, pushing back changes from the client to the server whenever connected and making sure all clients eventually get an updated version of the data, i.e. updates are streamed back to the clients.

In the Cloud Types model [1] developers can declare certain data as "cloud". The programmer can then simply program by using this data as if it where local, while in fact it is automatically shared between all participating clients. All the tedious tasks are taken care of by the underlying model (de-/serialization, network transport, network failures and consistency). This allows the programmer to write its applications as if it were a local program (i.e. focusing only on its data and the operations), while in fact the program automagically is distributed across all clients.

Proposal bachelor thesis

The Cloud Types model is implemented by keeping a log of all the operations users performed on the cloud data, called the GLUT (Global Log of Update Transactions) algorithm [2]. The current state of the program can then (conceptually) be created by replaying this log of operations. Thus, when a client wants to read a cloud type, it will (conceptually) go through the whole log, find all the operations that concern this value and replay them to get the current value (this is of course optimized). In order to get the same state across all participating programs, you need to get the same log of operations on all participants. This is achieved by the clients sending their new, delta operations to the server, the server adds the operations to its log and then sends the necessary information back to the clients so that they can rearrange their logs to comply with the server. This makes sure that all clients eventually have the same state, the state can be modified while disconnected and synchronization happens automatically (when connected).

In this bachelor thesis the student will implement the GLUT algorithm in an end-to-end JavaScript library. This involves a Node.js server library where the users can declare their data types and a JavaScript client library that can connect to such a Node.js server. The client can then start using the declared cloud data their API that is automatically shared between all connected clients.

Prerequisites

A strong knowledge of JavaScript (client-side, but also server-side Node.js and WebSocket for communication between the two) and its typed friend TypeScript [3] (or at least a great interest in and willing to learn independently) and interest in distributed programs.

Requirements

The student is required to fulfill the following requirements using TypeScript:

1. Implement a server-side Node.js library that fulfills the server protocol of the GLUT model. This library will have a clear API that allows the users to declare their cloud data and start the server.
2. Implement a client-side JavaScript library that fulfills the client protocol of the GLUT model. This library will have a clear API that allows the user to connect to a GLUT server in order to get the shared data and start using it through their respective API's.

Contact
Tim Coppieters – tcoppiet@vub.ac.be
http://soft.vub.ac.be/soft/users/tcoppiet

Preparatory course bachelor thesis

For this bachelor thesis the student needs to get acquainted with the literature in distributed systems and more specifically eventual consistency. Eventual consistency is the common term for programs that replicate data over multiple

sites (e.g. computers, phones…) and allow these sites to perform operations on the data while disconnected from each other. Consequently, the replicated data will diverge (e.g. one client will set a variable to 2 and the other to 3). Using an eventual consistency model, the replicas will then eventually converge to the same values so that they become consistent again. Since this is a fairly new field of research, the preparatory course will include:

- Reading papers about eventual consistency to get familiar with the common problems (what is the context, where are they used for, what are the common problems for these algorithms, etc.).
- Reading papers about the different existing solutions in eventual consistency.
- Perform a comparative study about the existing solutions and the GLUT model that will be implemented in this thesis. The study elaborates on the pros and cons of the model in comparison to the others and puts the GLUT model in context of the spectrum of eventual consistency solutions.

References

[1] Sebastian Burckhardt, Manuel Fähndrich, Daan Leijen, and Benjamin P. Wood. 2012. Cloud types for eventual consistency. In *Proceedings of the 26th European conference on Object-Oriented Programming* (ECOOP'12), James Noble (Ed.). Springer-Verlag, Berlin, Heidelberg, 283-307. DOI=10.1007/978-3-642-31057-7_14 http://research.microsoft.com/pubs/163842/final-with-color.pdf

[2] Sebastian Burckhardt, Daan Leijen, and Manuel Fähndrich. 2014. Cloud Types: Robust Abstractions for Replicated Shared. http://research.microsoft.com/apps/pubs/default.aspx?id=211340

[3] http://www.typescriptlang.org/