An open implementation of Cloud Types for the Web

Tim Coppieters
Vrije Universiteit Brussel
tcoppiet@vub.ac.be

Laure Philips
Vrije Universiteit Brussel
lphilips@vub.ac.be

Tom Van Cutsem
Vrije Universiteit Brussel
tvcutsem@vub.ac.be

Wolfgang De Meuter
Vrije Universiteit Brussel
wdmeuter@vub.ac.be

Abstract

Cloud Types is an interesting EC model that closely integrates its consistency model with the programming language, rendering EC programming more feasible for the average programmer. We have created an open implementation of the model in a JavaScript end-to-end implementation that can act both as an experimental platform for researchers and as an open EC model for the web. Furthermore we are experimenting with additions to the model such as an RDBM-like authorization system in combination with Views in order to give more control over the state and to allow partitioning.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Distributed Programming

Keywords Cloud Types, JavaScript, Web Applications, Authorization, Eventual Consistency

1. Introduction

During the last few years first attempts have been made to present easier Eventual Consistency (EC) programming models by integrating it more closely with the programming language. Cloud Types[1] is one of the most recent models that takes such an approach. It allows developers to create an EC application (providing both server and client replication, and thus offline availability of the data) by using automatically replicated data types with commutative operations (as also introduced by CRDT[2]) or last-writer semantics and strong consistency whenever required. While the programming model restricts the developer to certain EC choices (e.g. no conflict detection/resolution), it does provide a more understandable environment for the developers to create EC applications.

2. Open Web Implementation

We believe such an approach is the way to go to make EC programming more feasible. More intelligent language abstractions that cover the complex problems of EC will help and attract more developers to the EC community. In order to further progress our research in that direction, it is key to experiment with and build on previous work. Although the mentioned Cloud Types model is already utilized in TouchDevelop, a closed-source Microsoft product supported by the creators of Cloud Types, it was not yet operationalized in such way that researchers/developers can experiment with it, hands on.

Therefore, we decided to implement the complete model in a JavaScript end-to-end library approach[1]. In short, it allows you to declare the replicated data types using a Node.js server and use the EC data on the client-side (e.g. in your browser) with another JS library, using WebSockets for communication. The library has a clear API, an extensive documentation and an implementation that is kept as close to the original description of the language model as possible. This (1) allows researchers to better understand and experiment with the model and (2) makes the model available in a widely used environment, namely the web platform.

3. More Control

We are currently using this implementation to examine the boundaries of the model for web application programming. Thus far we encountered two main problems in order to use it for our day-to-day web applications. First, there is no way to control what happens with the shared data, i.e. we do not want everyone to see, create, update and delete all data. Second, there is no way to partition the state in such way that the client can decide to only access a part of the data. By allowing to retrieve and synchronize only the necessary data on demand, it becomes feasible to encode larger data sets in the cloud types.

These two concerns can be easily exemplified. For the first concern, take the grocery list example from the original paper[1]. In order to port this example to a mature and scaleable grocery list web application, we want users to register, login, create new lists and join other users' lists either by providing a special token supplied by the owner or by being added explicitly by a user of such list. Another example that demonstrates our need for control would be the typical in-house company application with managers, employees, payments, etc. (e.g. as used in [3]). A user can only start using this type of application if somebody gives him/her explicit access and adds him/her to a group with certain privileges. Furthermore we also want to add fine-grained dynamic constraints such as

[1] https://github.com/ticup/CloudTypes-paper

PaPEC '14, April 13 - 16, 2014, Amsterdam, Netherlands.

http://dx.doi.org/10.1145/2596631.2596640

”an employee can only see the salary of another employee if it is smaller than its own”.

For the second concern, take an event management system that contains a lot of related information about each event and participating artists. The last thing we want is to always send all the data to all clients. In contrast, we only want to send and synchronize the data that the client needs at a particular moment. By default, this could be all the data within the upcoming year. If the user needs data of previous events or events that occur more than one year from now, he/she could then request this on demand.

In order to solve these two issues we drew inspiration from the database community and created the following additions to the Cloud Types model:

1. An **authentication system**, adding the notion of users and groups and built-in functions *login* and *register*.
2. **Views** that allow for context dependent filtering by creating a filter on an Entity or Array where the inclusion of an entry can depend on the current user or user-provided parameters.
3. RDBM-like **privileges** on Entity, Array, Property, View and View Property level, based on [3]. These privileges are built on top of the system, using Cloud Types, meaning they also benefit from the eventually consistent properties of Cloud Types.
4. Statically defined, **data-driven server execution** (upon creation or deletion of an Entity or upon updating a Property).

These additions cover previously mentioned concerns and effectively allows us to implement proposed examples. While this is still work-in-progress, they are currently successfully implemented in the web library. We will not elaborate on the extensions here, but we are planning to publish them in the future using the formal Cloud Types model.

4. Discussion

Adding the notion of users to the system opens up the possibility of adding user-level guarantees for the replicated data using session guarantees[4]. The original Cloud Types model provides all guarantees (Read Your Writes, Monotonic Reads, Writes Follow Reads and Monotonic Writes) for a particular client, but if a user goes from one client to another (and thereby switches servers), only Monotonic Writes and Writes Follow Reads can be guaranteed for the user. Adding the Session Guarantees at user level, either all by default or either by specification as suggested in [5], would provide a more consistent view on the state for the user and consequently improve the experience with the system.

RDBM privileges, as described in [3], have a stricter notion of granting and revoking the privileges than we currently have. Namely, when a user A grants a privilege X to user B, a user C can not revoke that privilege from B, because user C did not grant that privilege to B. In our current implementation this is not the case, namely user C could revoke that privilege (at least if it has permission to grant/revoke such privilege). As a consequence we also do not perform recursive revoking, which is the act of revoking the privileges granted by a user if the privilege that allowed that user to grant those privileges is now revoked. This was done because our use-cases had more benefit from the simple, more relaxed, version. This stems from the fact that the notion of a user is somewhat different in our context from a user in relational databases.

The addition of statically defined, data-driven server execution allows us to execute code on the server in a very strict, but simple manner. The main purpose of this addition was to enable a user to grant himself privileges (code on the server is executed as root, which has all privileges) by creating the correct data instead of getting the privilege from another user. This allows us to implement the grouped grocery list requirement, where we want to be able to grant access to a user for a particular list if the user supplies the correct token for that list. While the data-driven approach is a nice and simple way to add server execution (results can be reported back to the client by creating data again), a more complex interleaved client/server execution model such as that from HOP[6] would be more powerful.

5. Conclusion

We introduced the JavaScript end-to-end open web implementation of the Cloud Types model. Experience from this implementation learned us that we needed more control over the shared data in order to use it for mature web applications. This was demonstrated by providing some example applications we wanted to create with the model, but which we could not with the current state. This lead to a brief introduction of some additions we currently made, allowing us to effectively implement the desired applications.

References

[1] S. Burckhardt, M. Fähndrich, D. Leijen, and B. P. Wood, ”Cloud types for eventual consistency,” in Proceedings of the 26th European Conference on Object-Oriented Programming, ECOOP’12, (Berlin, Heidelberg), pp. 283-307, Springer- Verlag, 2012.

[2] N. Preguica, J. Marques, M. Shapiro, and M. Letia, ”A commutative replicated data type for cooperative editing,” in Distributed Computing Systems. ICDCS ’09. 29th IEEE International Conference on, pp. 395-403, 2009.

[3] E. Bertino, P. Samarati, S. Jajodia, ”An Extended Authorization Model for Relational Databases,” in IEEE Transactions on Knowledge and Data Engineering, vol. 9, no. 1, pp. 85-101, January-February, 1997.

[4] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch, ”Session guarantees for weakly consistent replicated data,” in Proceedings of the Third International Conference on Parallel and Distributed Information Systems, PDIS ’94, (Washington, DC, USA), pp. 140-149, IEEE Computer Society, 1994.

[5] D. B. Terry, ”Replicated data consistency explained through baseball,” in Communications of the ACM, vol. 56, pp. 82-89, Dec. 2013.

[6] M. Serrano, E. Gallesio, and F. Loitsch. ”Hop: a language for programming the web 2.0,” in OOPSLA Companion. 2006.