SuperCopair: Collaborative Live Coding on SuperCollider through the cloud

Antonio Deusany de Carvalho Junior
Universidade de São Paulo
dj@ime.usp.br

Sang Won Lee
University of Michigan
snaglee@umich.edu

Georg Essl
University of Michigan
gessl@umich.edu

ABSTRACT

In this work we present the SuperCopair package, which is a new way to integrate cloud computing into a collaborative live coding scenario with minimum efforts in the setup. This package, created in Coffee Script for Atom.io, is developed to interact with SuperCollider and provide opportunities for the crowd of online live coders to collaborate remotely on distributed performances. Additionally, the package provides the advantages of cloud services offered by Pusher. Users can share code and evaluate lines or selected portions of code on computers connected to the same session, either at the same place and/or remotely. The package can be used for remote performances or rehearsal purposes with just an Internet connection to share code and sounds. In addition, users can take advantage of code sharing to teach SuperCollider online or fix bugs in the algorithm.

1. Introduction

Playing in a live coding ensemble often invites the utilization of network capability. Exchanging data over the network facilitates collaboration by supporting communication, code sharing, and clock synchronization among musicians. These kinds of functions require live coding musicians to develop additional extensions to their live coding environments. Due to the diversity of the live coding environment and the collaboration strategies settled for performances, implementing such a function has been tailored to meet some ensemble's requirements. In addition, networking among machines often requires additional configuration and setup, for example, connecting to specific machines using an IP address. In order to overcome these constraints, our goal is to realize a platform that facilitates the collaboration among live coders with minimal efforts of configuration, utilizing cloud computing.

There are many advantages to replacing a traditional server-client system with a cloud server. First of all, the collaboration scenario could be extended to the live coding ensemble whose members are distributed over different locations, enabling a networked live coding performance. Not only does this enable telematic performances, but it will also make a live coding session take place in a distributed manner, which will change the rehearsal process of live coding ensembles, whether it is remote or co-located. In addition, using the cloud server minimizes the amount of setup needed for networking as long as each computer is connected to the Internet. The amount of setup required is equivalent to creating a shared document in a Google Drive.

To that end, we present **SuperCopair**, a package for the Atom text editor, that offers code sharing and remote execution over the Internet. In this paper, we introduce the background that the idea is built upon, articulate our motivations for using cloud computing, and describe the implementation of the system. Finally, we suggest multitudes of new performance practices enabled by the system.

2. Networked collaborative live coding

Networked collaboration in live coding was present from the inception of live coding where multiple machines are clock-synchronized exchanging TCP/IP network messages (Collins et al. 2003). Many live coding ensembles also utilize network capability to share data and communicate within the ensemble (Collins et al. 2003; Rohrhuber et al. 2007; Brown and Sorensen 2007; Wilson et al. 2014; Ogborn 2014a). However, most of the time, they are based on the local network communication and not designed for remote collaboration attempted in the tradition of Network Music. Remotely connected music systems not only create a number of unique challenges and aesthetic opportunity as a performance in public, but also provide a base for musicians in different localizations to collaborate over the network synchronously.

Telepresence performance recently emerged as a new collaboration practice in live coding. Swift, Gardner, and Sorensen (2014) conducted networked performance between two live coders located in Germany and United States using an SSH server located in Australia. Extramuros, a language-neutral shared-buffer, is a web-browser based system to share code among connected machines (Ogborn 2014b). Gibber, a live coding environment on a web browser, supports collaborative editing and remote execution similar to Google Docs (Roberts and Kuchera-Morin 2012). Commodity softwares (such as Google Docs, CollabEdit, or DropBox) can be useful for remote collaboration and are convenient since users do not need to perform any configuration. However, these systems were either not designed or offer at best limited support for remote music performances.

3. Designing a collaborative solution

Although in the past it was difficult to think of thousands of people interacting at the same time on a musical system, the actual situation is in the quest of the best way to use the services and technologies offered day after day. For the last several years, we have witnessed an impressive advancement in the quality of services of cloud computing systems. Cloud servers distributed worldwide are connected through fiber optic broadband, and its cloud services have many advantages for computer music and collaborative works. We are taking some benefits from these characteristics in this study.

The cloud computing suggests itself as the next logical step in network capability, ready to be used for musical applications and performances. 'CloudOrch' is one of the first attempts to utilize the advantages of cloud computing in musical ways (Hindle 2014). The idea was to deploy virtual machines for client and server users, create websockets for intercommunication, and stream audio from cloud instruments to both desktop computers and mobile devices using web browsers. The author dubbed his idea 'a sound card in the cloud' and presented latency results from 100 to 200~ms between the Cybera cloud and the University of Alberta, Canada using the HTTP protocol. Using cloud computing as opposed to using server-client option has multiple advantages. Once an online instance is configured, a user can connect to or disconnect from the cloud at any time and can share the same resources within a group of connected users and take advantage of the reliable and scalable resources provided. Indeed, this solution can be useful for live coding network music.

The authors had already discussed models and opportunities for networked live coding on a past work (Lee and Essl 2014). The paper introduces diverse approaches in networked collaboration in live coding in terms of the type of data shared: code sharing, clock synchronization, chat communication, shared control and serializable data (such as audio). We draw upon ideas of existing systems that enable 'code sharing' and 'remote execution' re-rendering program state by evaluating code fragments in both the local machine and the remote machines in many live coding environments and extensions (Brown and Sorensen 2007; Rohruber and Campo 2011; McKinney 2014; Roberts and Kuchera-Morin 2012). These systems are similar in the sense that they need a separate server installed and configured by their users.

It is a general trend to have software application distributed over the Internet. Cloud computing is the central tool to realize the distributed softwares. However, the use of cloud computing is underdeveloped in computer music and we believe that it is the next logical step to put computer music applications in the cloud as a mean to realizing network music. The cloud computing provides a set of services that are beneficial to scale the computer music performance. For example, we can imagine a small scale ensemble co-located in the performance space, in which case the cloud computing will create a virtual machine based on the data center nearby the performance location. In the opposite case where large-scale participants are expected on a collaboration session, the cloud service will easily scale its computational power, network traffic bandwidth and storage space automatically to meet the spontaneous needs, although it will have some monetary cost.

In terms of network latency, we have achieved, an average round-trip time of 230~ms between Brazil and United States, and a minimum of 166~ms (Carvalho Junior, Queiroz, and Essl 2015). These tests were done using mobile devices connected to Pusher, a cloud service described below, but it can be extended to almost any device connected to the Internet. The strategy of transferring code (textual data) and re-rendering the program state remotely instead of streaming audio makes the latency less critical particularly for the scenario of live coding. However, it should be noted that the sound outcome from local machines and remote machines may not have exactly the same sound for many reasons (e.g., latency, randomness, asynchronous clock, packet loss).

The use of cloud computing resources became easier after the introduction of some cloud services that create an abstraction of the cloud computing set up and offer simple APIs for users, as we can find on Pusher. Pusher offers a cloud computing service that delivers messages through web sockets and HTTP streaming, and support of the HTTP Keep-Alive feature. The service has a free plan with some daily limitations such as 100,000 messages and a maximum of 20 different clients connected. Another limitation of the free plan is that we can only exchange messages through the US-East cluster server situated in Northern Virginia. The paid plans are more flexible and they make possible to have more users connected, send more messages, and use other clusters. In spite of that flexibility, all plans have a hard limit of 10

messages per second for each user. This limitation is due to the overhead of message distribution among a thousand users, but it really suits most needs to common use cases. Every message has a size limit of 10 kilobytes, but one can request an upgrade if larger messages are needed. Although it has limitations, we do not need to set up any cloud instance to benefit from the cloud computing infrastructure provided by this service.

The service works with push notifications, so every message sent is going to be received by all devices assigned to the same channel. A SuperCollider programmer can evaluate the whole code, a selected part, or just a line using keyboard shortcuts. SuperCollider programming language supports real time audio synthesis and is used extensively by live coders. These characteristics turn the language very suitable to be used with a push notification cloud service.

4. SuperCopair

The solution presented in this paper was created as a package to the Atom.io IDE. Defined as 'a hackable text editor for the 21st Century' on its site[1], Atom is a text editor created using web technologies and has its development powered by the github community. This IDE has numerous packages for many programming languages and presents some solutions for coding, debugging, and managing projects. Atom packages are programmed in CoffeeScript, which is a programming language that can easily be converted to Javascript and can also integrate its libraries. The developers can install Atom packages to enable various functionalities in the IDE such as: communicate through chats, use auto-complete in certain programming language syntax, interact with desktop and web applications, integrate with the terminal command line, and have many options based on other packages. These features have motivated the development of SuperCopair package for Atom.

SuperCopair is based on two Atom packages: atom-supercollider and atom-pair. The first package turns Atom.io as an alternative SuperCollider IDE and permits users to openly communicate locally with SuperCollider audio server through OSC in the same way we can do on SC-IDE. Moreover, the users can take advantage of other Atom packages additionally to quarks packages. The latter package is used for pair programing through the Internet. The atom-pair package is based on Pusher cloud service and its default configuration is based on the community free plan, but a user can modify the settings and use the user's own keys within the personal free or paid plan.We decided to merge both packages to add new features for collaborative live coding, and finally had dubbed it the SuperCopair package.

The main idea is that all participants have the opportunity to evolved into a collaborative performance.

The IDEs for SuperCollider have, by default, shortcuts to evaluate a line, a block, and to stop all sound process that is running. In addition to these options, the SuperCopair package includes methods and shortcuts to broadcast these events and execute them on all users connected at the same pairing session. Through the shortcuts, one can decide to evaluate selected code either only in the local machine, or in all computers of the session. One can also turn on and off a broadcast alert option in the settings in order to be asked or not before evaluating every broadcast event sent by another user in the same session. This allows each individual has control over which code to be evaluated in the local machine.

The broadcast events are diffused through the cloud service and are expected to be evaluated as soon as each device receives the event message. The message includes the code to be evaluated and the user identification. A representation of a session using SuperCopair package is shown at Figure 1.

4.1. Package installation and use

One can have the package properly installed and ready to be used in two different ways. Via the Settings View in Atom.io, the user can search and install the package. It is also possible to install the shell commands during Atom.io setup and use the line below to install the package:

```
apm install supercopair
```

After the installation, the user needs to start a new session before inviting others. An instructional step-by-step setup is presented on the package page. Then one can get initiate a performance by opening a SuperCollider file and starting a pairing session. The session ID string needs to be shared with collaborators so they can use the string to join the same session.

The shared session ID is based on the channel created at the cloud service and it contains application keys. It is recommended to change the app keys after each session. As the keys are linked to the account used during the performance, other participants can use the keys for other activities and the number of events will deducted from the main account.

[1] Atom.io website: http://atom.io/

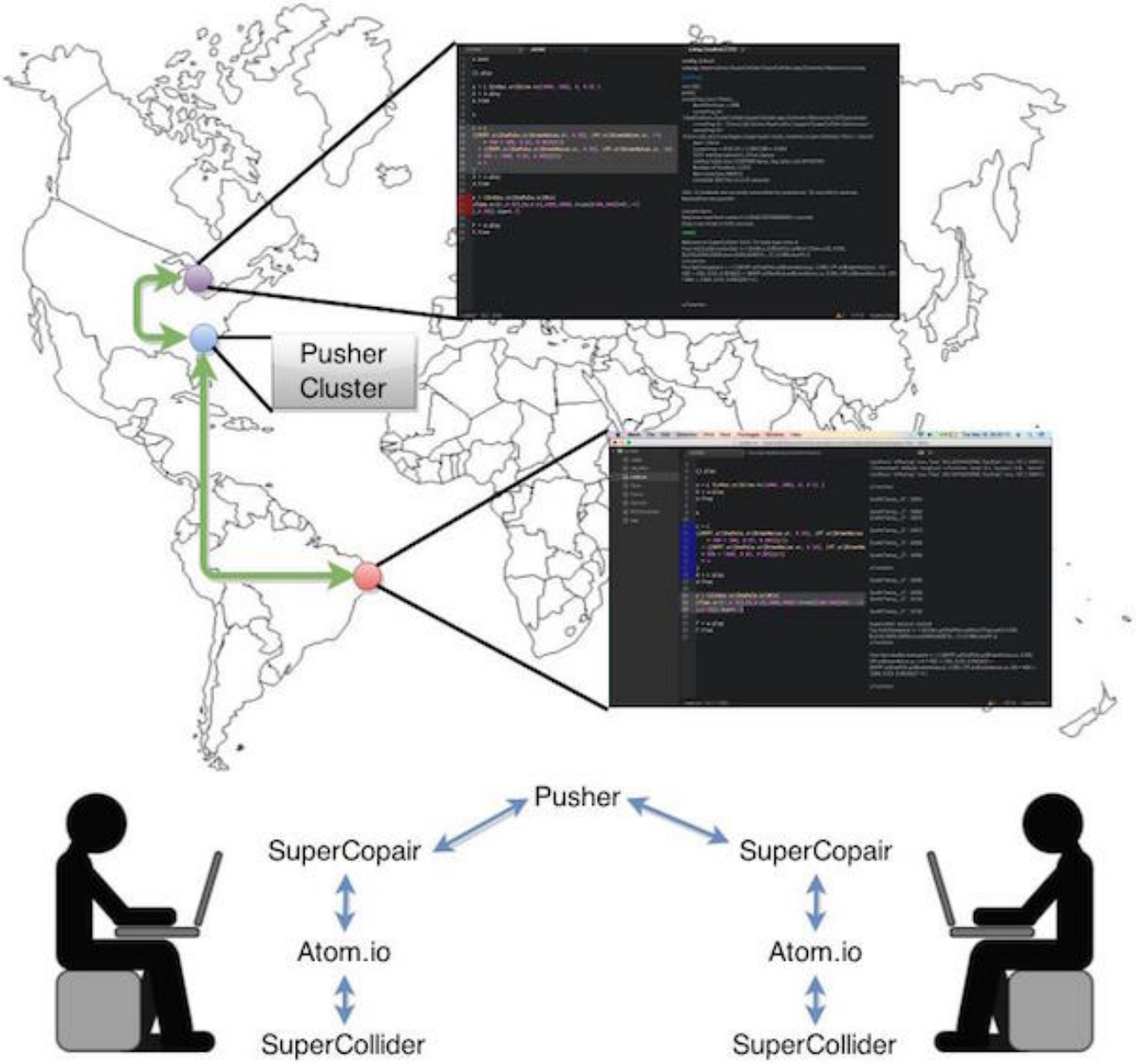

Figure 1: *Example of a session using SuperCopair and the architecture of interactions. The central bullet is the localization of the cluster server in Northern Virginia, and the other bullets represents users connected to the cloud server. The screen has the code on the left and SuperCollider post window on the right. The architecture presents only two users but it can be replicated to many, with diffusion on Pusher cloud service.*

The users who joins later will see the most recent version of the shared code. The users are identified by different color markers, and they can identify what each member is writing on the file based in these colors. A pop up provides information about users joining or leaving the session. Furthermore, a message including the identification of the user and also the code evaluated is shown at SuperCollider post window right after each broadcast event is evaluated. In case the broadcast alert option is on, a dialog will appear whenever an event message is received from another and ask if the user would accept or reject the code evaluation. The alert dialog will have the sender's id and the code sent via broadcast. When a live coder leaves the session, he or she can keep the most recent updated file to save or edit offline.

The delay achieved on the free plan depends on the distance between every member and the US East Coast cloud server. This free plan from Pusher cloud service allow 20 different clients per day on the same session and 100 thousand messages per day, however we have higher limits on paid plans. The session will stop after reaching the daily limit of messages for all plans, but the daily quota is reset after midnight UTC. It is important to keep these details in mind while facing any problem or high latency. The user may want to try a paid plan to have distributed data center, clients more than 20, and larger sized messages.

4.1.1. Shortcuts

The users have some special shortcuts depending on the operating system, and they are related to these specific functions:

- Compile library (open post window if needed)
- Clear post window
- Evaluate selection or current line locally
- Panic ! Stop all music
- Broadcast a code evaluation to everyone (including oneself) in the session
- Broadcast a code evaluation to others (excluding oneself).
- Broadcast stop command to everyone (including oneself) in the session
- Broadcast stop command to others (excluding oneself).

These shortcuts can be used to interact with other participants during a performance. The broadcast methods will only be shared with users on the same session, so it is also possible to create multiple sessions and interact with different crowd teams at the same time using a distinct Atom.io window on the same computer.

4.1.2. Practices and performances

The authors attempted to test the application multiple times in the co-located setup and also tried remote sessions by recruiting SuperCollider users. From one of our practices, there were live coders from Ann Arbor, MI, and San Francisco, CA, in U.S., and also São Paulo, SP, and Fortaleza, CE, in Brazil. During the session, participants (including the author) shared the session ID using an Internet Relay Chat (IRC) channel and we had a brief discussion about the package before starting to code. Some users reported that it could be dangerous to use headphones if we had switched off the alert for broadcast events, because some user may send a louder code to be synthesized. In the end, the session is successfully carried out without many problems and we are on the improvement of the package based on comments and suggestions from the participants. Atom.io installation was cumbersome for some users of Linux due to recompilation requirements at some distributions, and a step-by-step guide is under construction. Additionally, Mac users need the newest versions of the system in order to install Atom.io, but the users can also use a virtual machine with Linux and get rid of this limitation.

The practice addressed above is to simulate a networked live coding performance where multiple remote performers join a SuperCopair session from each one's our location, that may not be the concert space. In the local concert space where the audience is, a laptop connected in the session is placed without a performer on stage. Each performer would evaluate code in broadcast mode so that the computer on the stage will generate the collection of sound that remote live coders make via the Pusher. At this performance, the spectators at the local concert hall may have a wrong impression about the performance in the beginning because there is no one on stage except the laptop. As the audience will watch the video projection of the laptop screen, they will understand the main idea of remote performers live coding locally, from the multiple concurrent edits and some live coders' explanatory comments shown on the editor.

5. Discussion and conclusions

Here we present advantages and opportunities enabled with SuperCopair in network live coding: remote collaboration, telematic performance, and crowd-scale networked performance. One interesting advantage of the application is that it supports remote pair programming. We have witnessed that users can teach the basics of a language each other or help in bug fixing using online pair programming. Beginners can invite someone from anywhere in the world for a pairing session and start coding together on the same file and also synthesize the same code in real time while learning some tricks about live coding. Additionally to the forums and mailing lists, one can invite professionals to help on fixing algorithms for audio synthesis and have another kind of experience like pair or group programming on the same code to come up with a solution collaboratively. This package supports only SuperCollider namespace, but in the near future we can have similar packages for Csound, Chuck, or any other computer music programming language.

SuperCopair also offers novel forms of networked performances based on message streaming. The work of Damião and Schiavoni (2014) is a recent attempt to use network technologies in order to share contents among computers for a musical performance. The authors send objects and strings through UDP using OSC and can control other machines running the same external on Pure Data. They opted for UDP and Multicast to get better results on message distribution if compared to TCP and broadcast, which usually include three way handshaking and relay server. Although their decision has been based on solid arguments, our solution takes advantages of HTTP persistent connections using a single TCP connection to send and receive messages, and we also bring a reliable option for broadcast delivery using cloud services capabilities.

The package presented in this paper can be extended as an alternative for other networked live coding APIs. One can cite the Republic quark package that is used to create synchronized network performances. and the extramuros (Ogborn 2014b), a system for network interaction through sharing buffers of any kind of language. The last solution needs to be configured depending on the language andit does not present any easy way to share control (e.g. stop synthesis on SuperCollider) at the moment. Another constraint of both solutions is the need to create and configure a server on one computer to receive connections from clients, and additionally it would be necessary to open network ports or change firewall settings before starting any interaction with remote users.

SuperCopair realizes accessible configuration of network music performances, utilizing the cloud services. There is no need to configure a server or manage any network setting, e.g. routing, firewall, and port. We expect that even inexperienced users will be able to create a session with lots of people. As long as one can install the Atom editor and the SuperCopair package, the creation and participation at remote performances become an easy sequence of one or two shortcuts. Eventually, SuperCopair will simplify the steps to create a collaborative performance and remote rehearsals, and be used by people without network knowledge.

6. References

Brown, Andrew R, and Andrew C Sorensen. 2007. "Aa-Cell in Practice: an Approach to Musical Live Coding." In *Proceedings of the International Computer Music Conference*, 292–299. International Computer Music Association.

Carvalho Junior, Antonio Deusany de, Marcelo Queiroz, and Georg Essl. 2015. "Computer Music Through the Cloud: Evaluating a Cloud Service for Collaborative Computer Music Applications." In *International Computer Music Conference*.

Collins, Nick, Alex. McLean, Julian. Rohrhuber, and Adrian. Ward. 2003. "Live Coding in Laptop Performance." *Organised Sound* 8 (03): 321–330.

Damião, André, and Flávio Luiz Schiavoni. 2014. "Streaming Objects and Strings." *Live Coding and Collaboration Symposium*.

Hindle, Abram. 2014. "CloudOrch: a Portable SoundCard in the Cloud." In *New Interfaces for Musical Expression*, 277–280.

Lee, Sang Won, and Georg Essl. 2014. "Models and Opportunities for Networked Live Coding." *Live Coding and Collaboration Symposium*.

McKinney, Chad. 2014. "Quick Live Coding Collaboration in the Web Browser." In *Proceedings of New Interfaces for Musical Expression (NIME)*. London, United Kingdom.

Ogborn, David. 2014a. "Live Coding in a Scalable, Participatory Laptop Orchestra." *Computer Music Journal* 38 (1): 17–30.

———. 2014b. "Extramuros." https://github.com/d0kt0r0/extramuros.

Roberts, C., and J.A. Kuchera-Morin. 2012. "Gibber: Live Coding Audio in the Browser." In *Proceedings of the International Computer Music Conference (ICMC)*. Ljubljana, Slovenia.

Rohrhuber, Julian, Alberto de Campo, Renate Wieser, Jan-Kees van Kampen, Echo Ho, and Hannes Hölzl. 2007. "Purloined Letters and Distributed Persons." In *Music in the Global Village Conference (Budapest)*.

Rohruber, J., and A. de Campo. 2011. "The Republic Quark." https://github.com/supercollider-quarks/Republic.

Swift, Ben, Henry Gardner, and Andrew Sorensen. 2014. "Networked Livecoding at VL/HCC 2013." In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, 221–222. IEEE.

Wilson, Scott, Norah Lorway, Rosalyn Coull, Konstantinos Vasilakos, and Tim Moyers. 2014. "Free as in BEER: Some Explorations into Structured Improvisation Using Networked Live-Coding Systems." *Computer Music Journal* 38 (1): 54–64.