Computer Science Tripos – Part II – Project Proposal

A Distributed Approach to Collaborative Live Music Performances

Elias Calocane, King's College

Originators: Elias Calocane, Sam Aaron and Martin Kleppmann

20 October 2017

Project Supervisor: Martin Kleppmann

Project Adviser: Sam Aaron

Director of Studies: Dr T. G. Griffin

Project Overseers: M. P. Fiore & T. M. Jones

Introduction

Making music using computers, is a well-established process. Digital Audio Workstation (DAW) software is available for users to create music on their machines using components such as real and virtual instruments, vocal tracks and other types of samples. In terms of collaborating, DAWs allow multiple users to synchronously work on a project, although it is convention for this to be done in the same location and machine.

In terms of composition or performance of music where users collaborate remotely (for example over a network), in both asynchronous and synchronous cases of contributions, the options are far more limited. One option for collaborative composition, is for collaborators to share project files and asynchronously build a musical piece e.g. via a network filesystem. However, this approach does not allow for several users to concurrently edit the files (in a synchronous fashion). Ableton use another (synchronous performance) approach through Ableton Link, an extension for their DAW and other Link-compatible software. Link allows musicians using compatible software to play in sync (live) whilst connected through a network – by ensuring that all players adhere to a global sense of time through tempo synchronisation, beat alignment and phase synchronisation. However, Link requires that all collaborators are connected to the same local area network.

This project sets out to overcome the local network restrictions of Link and provide a Link-inspired functionality that works in a wide area network, whilst still focusing on the element of live performances by making use of Sonic Pi – thus providing a synchronous and distributed method for musicians to collaborate.

A wide area approach naturally introduces challenges involving dealing with networking issues such as latency and jitter, routing messages through firewalls and NAT middleboxes, as well as (very critically) the problem of making sure everyone involved in the performance observes the same events in the same order. This distributed sense of coherence is vital to ensuring that everyone perceives the performance in the same way (so that it sounds the same for all users), and this coherence is the main focus of this project.

Starting Point

Relevant Courses include Networking and Distributed/Concurrent Systems from Part IB.

Access to Ableton Link's developer resources (including GitHub repository with source code).

Access to the source code for Sonic Pi through GitHub.

Key Concepts

These are the main concepts/technologies this project will try to bring together

Link

Link is an open source networking library for music applications that allows users to play in sync by synchronising tempo, aligning beats and synchronising phase. Below is an explanation of the main concepts behind Link.

Tempo synchronisation

Tempo can be described as the rate at which a musical piece progresses, measured in beats per minute (BPM). To allow different musicians to play with each other in a coherent manner, it is essential that their contributions to the musical piece advance at the same rate, by making sure that all musicians adhere to the same predefined tempo.

Beat alignment

Advancing at the same rate is not sufficient for contributions to a specific piece to sound coherent – it is also essential that the beats occur at the same time (that they are aligned). In order to clarify what is meant by this, we adopt a numbering system for beats such that the first beat is denoted by number 1, the second by number 2, the third by number 3, etc. Beat alignment ensures that the difference between the beats of any two users, is always an integer, such that it is possible for one user to be at beat 4 whilst another is at beat 2, but it is not possible for third user to be at beat 2.5 (halfway between beat 2 and 3).

Phase synchronisation

When dealing with loops or bars (consisting of a set number of beats) it is also useful to ensure that there is some consensus regarding the points at which they start and end. This is so that loops created by different users start/end at sensible places relative to each other. For loops of the same size, this means starting and finishing together (which may involve delaying the triggering of loops so their start in sync with others). For loops of different sizes, there may be more than one point at which a loop can be triggered – for example a 4-beat loop can start either at the start or the middle of an 8-beat loop.

Open Sound Control (OSC)

Open Sound Control defines a message-based protocol that allows communication between different multimedia devices such as computers and synthesisers. On networks, OSC messages are most commonly transported as UDP packets. OSC messages can be used to encode descriptions of sounds (or musically related events) that can then be used in various scenarios. We will make use of OSC messages to communicate between Sonic Pi instances (being run on different machines by different users) to ensure that they all agree on which sounds to produce.

Sonic Pi

Sonic Pi defines a well-established language and software that allows users to produce music through live-coding. OSC messages are used within the software to describe music related events that need to be relayed to a speaker (through an audio synthesis platform called SuperCollider, integrated into Sonic Pi). Sonic Pi also defines temporal semantics which provide ways of reasoning about when sounds produced are by the software as well as making sure this occurs in a deterministic manner (or relaying to the user when that is not a possibility).

Structure

Implementation

A user writes a script using Sonic Pi that generates OSC messages describing the sound they intend to make. These messages are then interpreted by the SuperCollider server to play the correct sound from the user's speakers. In the current implementation of Sonic Pi, these OSC messages flow only between the Sonic Pi process and SuperCollider running on the same machine. In this project we plan to extend Sonic Pi for distributed use by also transmitting these OSC messages over wide-area networks to other participants in the musical session. These messages need to be sent from each Sonic Pi instance to all other instances that are part of the performance so that other contributors are also able to hear the sounds that the original user intended to produce. A server will be used to setup a connection between Sonic Pi instances so that OSC messages can be transported between them.

Since we are focusing on a collaborative performance, it is vital that all users hear exactly the same sounds – or in other words, that the lists of OSC messages interpreted by each user's SuperCollider server, are exactly the same. It is therefore important to account for aspects such as latency and jitter in the network to ensure that we are able to provide a deterministic order of events for all users when they are all transmitting OSC messages to each other.

Sonic Pi schedules sounds to be played by SuperCollider ahead of time (the default schedule ahead time is 0.5 seconds). This is done to provide users a well-defined level of determinism describing the timing relationship between when an event described by a script is scheduled and its corresponding sound is played. This schedule ahead time gives us time to relay OSC messages over the network – assuming latency is low enough and jitter issues are dealt with, we aim for these messages to reach all users before their respective sounds are expected to be played. When users all play together and their OSC messages have been exchanged, the data sent to each SuperCollider instance is the union of all the messages generated by the participants. We wish to sort these OSC messages by their scheduled timestamp (and schedule them) so that everyone observes the same message sequence.

Evaluation

We predict that it is possible to maintain a coherent musical collaboration through a wide area network if latency and jitter are below some threshold – as mentioned above, Sonic Pi provides a scheduleAheadTime parameter set to 0.5s by default, and this parameter can be used to force Sonic Pi instances to adhere to the threshold we define. We will initially set our threshold as the default scheduleAheadTime, but will use network data from different sources (described below), as well as features from Links Phase Synchronisation concept, to help define a suitable upper bound for the threshold.

We will test our hypothesis by emulating different network conditions using dummynet (or a similar tool). Using dummynet, we can control packet delay and jitter and investigate the effect they have on the coherence of a distributed music session, thus determining whether coherence is observed when conditions are under the threshold. We will measure coherence by comparing the sequences of OSC messages received by each SuperCollider instance – looking at the events describing the sounds each user is about to play. If these sequences are identical, so that messages are in the correct order (before they are scheduled to be played), we will conclude that the session is in a coherent state. We will measure latency and packet loss rate in both the University network, and a home ISP to find out what kind of network conditions we are likely to encounter (on average) – this will help guide the emulation of network conditions using dummynet, ensuring the system is able to deal with realistic scenarios. Using these findings, we will attempt to improve our implementation to automatically tune the scheduleAheadTime parameter of the separate Sonic Pi instances to adapt to different network conditions – keeping scheduleAheadTime as low as possible so users experience the best possible response time (in audible terms) when submitting changes to their own scripts.

We aim to create different versions of the system using TCP and UDP and use the protocol that provides the best performance. We first need to determine how costly packet delay, jitter and packet loss are to the quality of the music session (in terms of coherence). We will measure how much adjusting these aspects individually (using dummynet) affects coherence – for example, by examining the OSC messages received by all users in a session, and counting the number of messages that were observed in an incorrect order (or not at all). Protocol performance will then be rated using measurements of the latency, jitter and packet loss rate that the network experiences when using the protocols – by comparing the performance measurements to their relative costs on session quality, we will be able to establish which protocol provides the better results.

Success Criteria

Implement system allowing at least three users to setup a connection and play together using Sonic Pi i.e. allow users to write scripts describing sounds they wish to play as contributions to the performance, broadcast OSC messages describing these sounds to other members of a session, allow users to hear each other's contributions to the performance, propagate changes to contributions to all users in the session.

Ensure coherence of each user's notion of the performance – so all users that are part of a performance hear the same thing. We wish to guarantee coherence for all users involved in a session when network conditions are acceptable i.e. when latency and jitter do not cause a delay that surpasses a defined threshold.

Ensure that when in a session, users' timelines don't drift relative to each other – as the clocks rates of different users are likely to differ, we wish to prevent clock drift as this would eventually render users unable to react to each other's contributions (even when coherence holds). We wish to implement a clock synchronisation method and ensure it works by regularly comparing clock timestamps of users in a session.

Possible Extensions

If I achieve my main result early I shall try the following,

Integrate WebRTC into the project instead of using a server to provide a peer-to-peer implementation – if such an approach were to reduce latency significantly, it may be possible to implement features such as making it possible for different users to update (different) aspects of the same sound at the same time (whilst still keeping a deterministic view of events across all users). Carrying out the evaluation steps for this new implementation would allow for a comparison between the original server-based and peer-to-peer approaches to the project.

Trying out different schemes for getting coherence.

Resources Required

I will be primarily relying on the use of my own laptop for the project – a laptop with an Intel Core i7 4720HQ Processor, 16GB RAM, 512GB SSD and NVIDIA GeForce GTX 960M GPU running Windows 10.

If my laptop should fail, I will use a MCS workstation in the Computer Laboratory.

I will run weekly backups to a personal USB stick as well as to GitHub and Google Drive.

Timetable

1. **Michaelmas weeks 2–4:** Read up on similar projects already out there and familiarise myself with Ableton Link's and Sonic Pi's source code and implementations.

2. **Michaelmas weeks 5–6:** Start first implementation of the server with first protocol (UDP or TCP). This initial implementation will be a simple echo server that will receive OSC messages from a Sonic Pi instance, and send them back.
   Milestones: Test echo server with connection to one Sonic Pi instance – make sure it can send received messages back to Sonic Pi.

3. **Michaelmas weeks 7–8:** Finish the first implementation of the server. By this point I aim to be able to: setup a connection between (at least two) users, send and receive OSC messages from a Sonic Pi instance, schedule received OSC messages (so that a user will be able to hear the sound corresponding to the messages sent by another user).
   Milestones: Build server (with integrated clock synchronisation method).

4. **Michaelmas vacation:** Carry out alternative implementation of the server using second protocol. Start second implementation of the server for both protocols. Start planning dissertation chapters. Take measurements of latency and packet loss rate in home ISP.
   Milestones: Collect home ISP network data. Write draft of dissertation introduction.

5. **Lent weeks 0–2:** Finish the second implementation of the project. By this point I aim to be able to: provide some sense of consensus across all users – this will involve merging the OSC messages produced by a user with those received from other users and doing so in a consistent manner for all instances that are part of the session.
   Milestones: Write progress report. Simulate session to test coherence and collect results for improvements. Write draft of preparation and implementation chapters.

6. **Lent weeks 3–5:** Run main experiments described in evaluation and achieve working project.
   Milestones: Test UDP vs TCP and collect results. Measure session coherence. Write draft of evaluation chapter.

7. **Lent weeks 6–8:** Absorb any incomplete work from the previous stages (if any of them were delayed). If possible, start integration of alternative WebRTC-based method of communication.
   Milestones: Write draft of conclusions chapter.

8. **Easter vacation:** Revise drafted chapters. Work on project extensions (optional).
   Milestones: Complete first version of dissertation.

9. **Easter term weeks 0–2:** Carry out any further evaluation required, add content to chapters regarding any completed extension work.
   Milestones: Complete final version of dissertation.

10. **Easter term week 3:** Proof read - assuming no considerable delays were experienced in the previous stages.
    Milestones: Submit Dissertation.