Subtlenoise: sonification of distributed computing operations

P A Love
Department of Physics, Lancaster University, LA1 4YB, UK
E-mail: p.love@lancaster.ac.uk

Abstract. The operation of distributed computing systems requires comprehensive monitoring to ensure reliability and robustness. There are two components found in most monitoring systems: one being visually rich time-series graphs and another being notification systems for alerting operators under certain pre-defined conditions. In this paper the sonification of monitoring messages is explored using an architecture that fits easily within existing infrastructures based on mature opensource technologies such as ZeroMQ, Logstash, and Supercollider (a synth engine). Message attributes are mapped onto audio attributes based on broad classification of the message (continuous or discrete metrics) but keeping the audio stream subtle in nature. The benefits of audio rendering are described in the context of distributed computing operations and may provide a less intrusive way to understand the operational health of these systems.

1. Introduction
The monitoring and operation of distributed computing consume significant effort in large HEP experiments. It is common for HEP computing to have dozens of monitoring systems each containing several key views. In order to digest the operational situation members of the experiment take 8-hour shifts to survey the information and report on any situation that requires attention. Since many of these views follow the pattern of time-series histograms it is easy for shifters to get overwhelmed by the information leading to a poor understanding of the current state of the systems. In this paper we explore the use of sonification as a means to reduce this cognitive overload. A prototype sonification stack will be described along with some perceived benefits to operating distributed computing systems.
2. Architecture

A simple architecture to implement and demonstrate the sonification requires collecting messages from the computing services, transporting messages, transforming the information, and finally rendering the information using a suitable synth engine. A schematic of these components and the broad message flow between components is described in figure 1.

2.1 Collection

Logstash [1] is used as the collector of service messages. Typically the service will write a syslog formatted logfile where each message describes an action taken by the service. Logstash is a popular choice to collect log messages and has a rich set of input, filter, and output plugins. Of particular interest here is the ZeroMQ output plugin described next.

2.2 Messaging

The choice of messaging system was determined by the need to have a high performance and scalable publish-subscribe model. ZeroMQ [2] fit these requirements and is described as ‘sockets on steroids’ with ubiquitous sockets being familiar to system administrators. ZeroMQ has a number of communication patterns and here we use the PUB-SUB pattern that allows publishing from many services with a central Forwarding device to republish the messages. Republishing allows the central python application to subscribe from any IP address.
2.3 Python application
The service log messages are consumed by a python application called `dj.py`, which is responsible for interpreting the message content and transforming the message such that a synth engine can render it. The core routine in the application is an event loop that waits for messages to arrive on the ZeroMQ socket. When a message is received it is filtered and information extracted to be used in an outgoing message via the Open Sound Control protocol.

2.4 Open Sound Control
The Open Sound Control protocol “is an open, transport-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices” [3]. It can be thought of as an alternative to the popular MIDI standard commonly found in electronic music. Various implementations of the specification have been produced in the form of language bindings, of which we use a python implementation. Once constructed the OSC messages are sent to an audio rendering synth engine.

2.5 Audio rendering – Renoise
The initial prototype uses a popular commercial synth engine called Renoise [4]. This application comes with a large library of audio samples, instruments, filters, and other powerful tools for manipulating sound. Renoise includes an OSC server capable of interpreting OSC messages.

3. Implementation
The architecture described above has been implemented by the ‘Subtlenoise’ project and is available on github [5] under the GNU Public Licence. The choice of software components was made in order to fit into existing infrastructure and is summarized in table 1.

| Component | Role             |
|-----------|------------------|
| Logstash  | Collector        |
| ZeroMQ    | Messaging        |
| Python    | App              |
| OSC       | Sound protocol   |
| Renoise   | Synth engine     |
| Overtone  | Synth engine     |
| Sonic Pi  | Hardware         |

The current implementation takes a rather naïve approach when sonifying the service messages. The messages can be complex with an unknown range of metrics and it is up to the python code (dj.py) to map these messages onto something suitable for audio rendering. Other sonification projects use this technique but a more extensible and general approach would avoid understanding the data a-priori but rather use all input data and attempt an algorithmic mapping, such as by using a locality-sensitive hashing for example. This would result in sounds representing the data without any bias coming from
the interpretation of the input during the mapping step. Various techniques are explored in The Sonification Handbook [6].

3. Benefits over visual representation

Working with this system to sonify real-world applications has highlighted a few clear benefits over visual representation of the same metrics.

- Service feedback is immediate and real-time which allow shifters to quickly conclude a service is healthy.
- Auditory displays have high levels of trust. If the monitoring chain has a problem then no sound is heard which avoids the visual problem of looking at stale information.
- Subtle nuances in the service performance can be identified by small changes in the audio stream. These can help pre-empt service faults rather than being reactionary.

4. Future development

Two main areas are the subject of future development. The first is to replace the Python and Renoise components with the Sonic-Pi [7] application. Sonic-Pi is an application originally deployed on the Raspberry-Pi hardware platform but now available on other platforms including OSX. Sonic-Pi is built using the Overtone [8] project and provides a programming environment (using Clojure) which interfaces with the excellent open source synth engine called Supercollider [9]. This stack is actively being developed and would allow deployment of a small physical device (Raspberry-Pi) as an auditory display.

The second area of development is to move away from Parameter Mapped Sonification to a Model-Based Sonification to provide a more subtle and natural experience. Combining this with Auditory Icons would be enough to match the two equivalent patterns seen in visual displays. Advanced technique are explored by the International Community for Auditory Display [10]

5. Conclusions

In this contribution an architecture for the sonification of distributed computing operations is described. The system uses components familiar to system administrators there by allowing easy integration to existing systems. Some benefits of auditory displays over visual displayed are described.

References

[1] Logstash https://www.elastic.co/products/logstash
[2] ZeroMQ http://zeromq.org/
[3] Open Sound Control http://opensoundcontrol.org/
[4] Renoise http://renoise.com/
[5] Subtlenoise project page https://github.com/ptrlv/subtlenoise
[6] The Sonification Handbook http://sonification.de/handbook
[7] Sonic Pi http://sonic-pi.net/
[8] Overtone http://overtone.github.io
[9] Supercollider synth engine http://supercollider.github.io/
[10] International community for auditory display http://www.icad.org/