CloudOrch: A Portable SoundCard in the Cloud

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
abram.hindle@ualberta.ca

ABSTRACT

One problem with live computer music performance is the transport of computers to a venue and the following setup of the computers used in playing and rendering music. The more computers involved, the longer the setup and teardown of a performance. Each computer adds power and cabling requirements that the venue must accommodate. Cloud computing can change all this by simplifying the setup of many (10s, 100s) of machines with the click of a button. But there's a catch, the cloud is not physically near you, you cannot run an audio cable to the cloud. The audio from a computer music instrument in the cloud needs to be streamed back to the performer and listeners. There are many solutions for streaming audio over networks and the internet, most of them suffer from high latency, heavy buffering, or proprietary/non-portable clients. This paper proposes a portable cloud-friendly method of streaming, almost a cloud soundcard, whereby performers can use mobile devices (Android, iOS, laptops) to stream audio from the cloud with far lower latency than technologies like Icecast. This technology enables near-realtime control over computer music networks enabling performers to travel light and perform live with more computers than ever before.

Keywords

cloud computing, web-audio, cloud instruments, cloud orchestra

1. INTRODUCTION

Cloud computing is the tech buzzword trumpeted from the rooftops, as the supposed savior of business and IT, Cloud computing offers organizations the ability to treat computing as a utility like they treat power generation and distribution. Few businesses within a city run any sort of power generation and distribution machinery as it is a utility provided by electric utility companies. One can get enough electricity to cook a turkey and recharge an electric car at the flick of switch, or you can use just enough to power a meager laptop and a desk lamp. Cloud computing follows the same model, you no longer have to host your own computers, you can rent virtual computers (*virtual machines* (VMs)) and dispatch them to do work for you [4]. If you have a lot of work, you provision a lot of computers, if you do not need them anymore, you release them back into the cloud. Thus what the cloud promises is elasticity (providing more computers on demand); as well it treats computing as a utility. Can the same be true for computer music and computer music performance?

Imagine now a soloist who walks on stage with a smartphone in their hand. They open a web-browser and navigate to the cloud-sound-card which proceeds to stream music produced by a real-time computer music synth composed of a network of 1000s of CPUs, back to the soloist. The soloist then, via the web, controls and commands these 100s to 1000s of virtual computers to produce computationally or information intensive computer music. With a mobile device, or a laptop, one could connect to the cloud and stream the audio back, as well as use the web to control the instrument itself [2].

The problems that face anyone interacting with computer music, soft-realtime performance, and the cloud are:

- Software support for streaming: not all methods of streaming have wide software and platform support (e.g. difficulty of deploying `jack` on iOS or Android).
- Network support for streaming: many networks exist behind NATs and firewalls. This means clients must initiate connections while keeping in mind many ports are block and UDP based protocols tend not to work.
- Latency of audio streaming: HTML5 audio tag gives no guarantee on the size of the audio buffer used by a browser. RTP clients often give no guarantees either. Icecast and similar MP3 streaming tools tend to incur heavy encoding latencies into streaming. Large buffers induce large latency.
- Remote control of cloud instruments: if an instrument exists on many computers how do they communicate with each other and how do they communicate back to the musician?

This work describes an implementation of a web-based "Cloud sound card" that allows reasonably low latency streaming (100ms to 200ms rather than 30 seconds) of audio to a web browser over HTTP. Descriptions of integration on the cloud-end, how an end point can be created, and how a cloud synthesis network can be interacted with are also provided.

Thus the contributions of this paper are:

- A proposal to use cloud-computing to compose large instruments;
- A description and implementation of free-open-source streaming software that allows one to listen to instruments situated in the cloud;
- An addressing of the software, network, and latency aspects with existing portable open-source software.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NIME'14, June 30 – July 03, 2014, Goldsmiths, University of London, UK.
Copyright remains with the author(s).

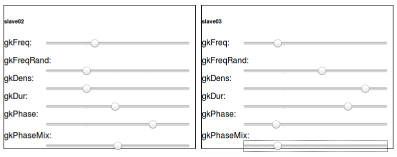
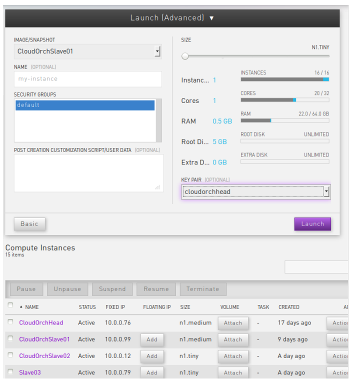


Figure 1: Screenshot of the granular synthesis interface.



IMAGE/SNAPSHOT: CloudOrchSlave01
NAME (OPTIONAL): my-instance
SECURITY GROUPS: default
POST CREATION CUSTOMIZATION SCRIPT/USER DATA (OPTIONAL)

SIZE: N1.TINY

Instanc...	1	INSTANCES	16 / 16
Cores	1	CORES	20 / 32
RAM	0.5 GB	RAM	22.0 / 64.0 GB
Root Di...	5 GB	ROOT DISK	UNLIMITED
Extra D...	0 GB	EXTRA DISK	UNLIMITED

KEY PAIR (OPTIONAL): cloudorchhead

Compute Instances
15 items

▲ NAME	STATUS	FIXED IP	FLOATING IP	SIZE	VOLUME	TASK	CREATED	AC
CloudOrchHead	Active	10.0.0.76		n1.medium	Attach	-	17 days ago	Actio
CloudOrchSlave01	Active	10.0.0.99	Add	n1.medium	Attach	-	9 days ago	Actio
CloudOrchSlave02	Active	10.0.0.12	Add	n1.tiny	Attach	-	A day ago	Actio
Slave03	Active	10.0.0.79	Add	n1.tiny	Attach	-	A day ago	Actio

Figure 2: Example of managing cloud virtual machines.

2. PREVIOUS WORK

This section discusses some of the previous work relevant to networked audio, networked computer music, networking latency, streaming technology and networked orchestras.

Oh et al. [9] in “Audience-Participation Techniques Based on Social Mobile Computing” show that smartphones are convenient for both audience participation and computer music performance. While Jordà [8] discusses methods used to play or interact with multi-user instruments and how instruments differ regarding shared collective control.

TweetDreams, by Dahl et al. [6], rely on Twitter's cloud to allow an audiences to interact with a musical instrument via a web interface (Twitter). The audience participates by tweeting at the performer's Twitter account.

From an infrastructural point of view, Jesse Allison et al. [1, 2] describe using web frameworks such as Ruby on Rails to distribute the performance among many users. They rely on HTML5 technologies to make interactive and musical web applications. Allison argues that adhering to HTML5 standards allows for client-side instrument UIs to be distributed without having to address many portability issues.

massMobile by Weitzner et al. [13] is a system meant to allow smartphones to interact with and Max/MSP via web standards (HTML5), over a variety of wireless networks. Their configuration includes a persistent internet accessible sound server running Max/MSP on a single machine.

Laptop orchestras [12, 11] are quite relevant to this work because the cloud can augment laptop orchestras. Furthermore technologies such as `jack`, that enable laptop orchestras can be used to aide the composition of cloud instruments. Barbosa et al. [3] show that networked computer music has been around since the 1970s and many of the techniques used in 2003 are still relevant to this day.

3. CLOUD SOUND CARD

One problem with virtual computers or virtual machines (VMs) that exist in the cloud is that they usually lack actual sound cards. Furthermore the lack of locality in the cloud dictates that even if the virtual computers had soundcards one would have to transmit the audio signals over a wide area network anyways.

`jack` [5] or similar sound daemons can be used in place of actual sound devices. A `jack` network allows applications to share synchronized audio and midi streams with each other. NetJack enables sending `jack` data over a UDP network.

Thus this strategy is to deploy NetJack on a collection of virtual machines in a cloud, then export and stream the final mixed audio back to the performer/listener without using `jack`. This means that the listener does not need to act as the master `jack` server, as this is heavy and onerous requirement that would exclude many mobile devices.

One avenue for portable audio streaming is web audio [10]. A limitation that faces web audio is that browsers and other web audio clients are often tuned for smooth heavily buffered playback. They will often have large buffers that fill up with audio data, of multiple seconds to allow for jitter in the timing of received packets. This avoids the problem of packets being late and audio cutting out due to lack of data, but induces seconds of latency. This latency is worsened by the large buffers of the encoders that need data ready for computationally intensive encoding and streaming. The sum of all these buffers produces heavy latency.

By reinventing the streaming wheel and leveraging websockets [7] to stream audio packets one can keep the latency to the end listener/performer down below 1 second. This is far better than using heavily buffered audio infrastructure behind the HTML5 `audio` tag. Furthermore websockets are part of the HTTP spec and are allowed through firewalls, solving the network configuration problem as well. By connecting to a websocket serving HTTP server the user creates the connection that is routable by the firewalls and NATs found on many networks.

3.1 Backend Implementation

The Cloud Sound Card is a web-application that acts as a `jack` playback sink, whereby any `jack` source that is connected to it will be linearly mixed into this sink. This sink is then read by the web-application for each small buffer of audio (usually 256 samples or more). For each listener every new audio buffer is copied to its queue and eventually sent to the listener over the TCP-based HTTP websocket connection that was established.

Websockets [7] are a bidirectional socket/pipe abstraction built on top of HTTP and TCP. They enable both sides to send messages back and forth over HTTP. Thus a client webbrowser will subscribe to the streaming websocket.

3.2 Cloud Sound Card Client

The cloud sound card is a JavaScript program run in a webbrowser that receives and plays audio, often raw audio, streamed over a websocket to the client. The client uses the web-browser's webaudio system and a webaudio `ScriptProcessor` node to fill the audio buffer being played on the soundcard. HTML5/JavaScript's `ScriptProcessor` lets you fill the audio buffer with arbitrary audio data.

Most modern web-browsers such as Firefox, Chromium, and Safari support webaudio streaming [1]. Furthermore any Android device with a newer Chromium browser or a Firefox Fennec mobile browser can listen to webaudio already. This cloud sound card has been tested and functioned well on a Galaxy Nexus smartphone running Mozilla Firefox Fennec also called Firefox Beta.

The client software negotiates with the web application, subscribes to a websocket, and then is streamed audio in buffer sized chunks that it it uses to fill the buffer of the `ScriptProcessor`. As long as the websocket can fill the buffer, the audio is played clearly and uninterrupted. If

[1]Is your browser supported? `http://caniuse.com/audio-api`

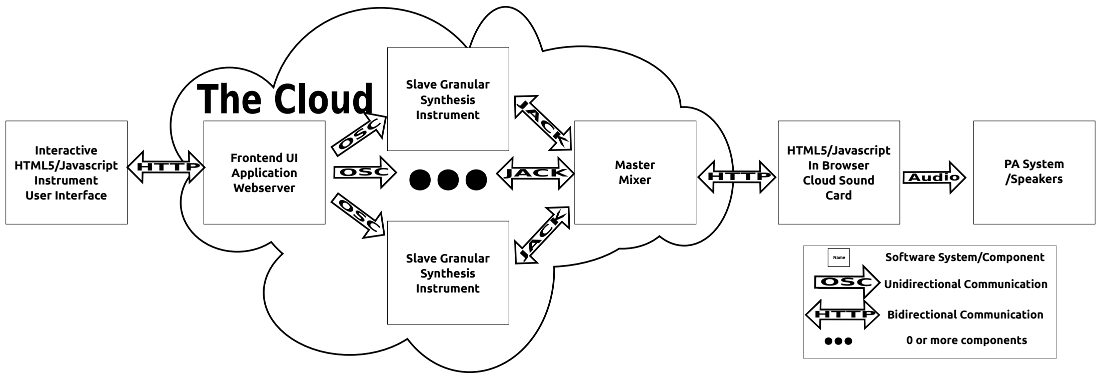

The Cloud
Interactive HTML5/Javascript Instrument User Interface
HTTP
Frontend UI Application Webserver
OSC
OSC
OSC
Slave Granular Synthesis Instrument
Slave Granular Synthesis Instrument
JACK
JACK
JACK
Master Mixer
HTTP
HTML5/Javascript In Browser Cloud Sound Card
Audio
PA System /Speakers
Software System/Component
OSC Unidirectional Communication
HTTP Bidirectional Communication
●●● 0 or more components

Figure 3: Architecture of an example Cloud Music Instrument and Cloud Sound Card Interface

there are networking issues, such as a poor wifi signal, there can be drop outs.

Thus the cloud sound card relies on a `jack`-client webserver and a websocket HTML5 JavaScript client. This HTML client is quite portable and in future years will become more portable. Latency is reduced by using the ScriptProcessor, thus the buffer size is controlled, rather than uncontrolled buffer sizes used in the web browser.

4. A CLOUD-BASED COMPUTER MUSIC INSTRUMENT

The demo cloud instrument [2] was a 15 computer cluster, with one master node that acted as the cloud sound card server and the web UI control server (see Figure 2). It had 1 external IP and 1 cloud IP. Clients could connect to the external IP, while the audio generator slaves would connect via the cloud IP. The architecture is depicted in figure 3.

The instrument was a time-stretching granular synthesis instrument that was creating grains from time stretching a recording Goldberg Variations, Aria Da Capo [3] Each granular synthesis instrument would be initialized with a different density of grains, different frequency of grain playback, different distribution of frequencies, and different grain lengths and phases. These parameters were controllable via Open Sound Control (OSC). But this would mean there would be 14 computers all listening to similar OSC commands on different machines. Thus any parameter in the network could be executed via a UDP OSC command. But the UDP routing was not exposed to the internet.

Those outside the cloud could control the instrument via a simple web interface, see Figure 1, that used XHTTP requests to send updated OSC messages to the machines in the cloud. This is equivalent to the passthru instruments discussed by Allison et al. [2] in their Nexus work. The XHTTP request would be JSon formatted to look something like a OSC packet. The JSon packet specified, a host, a path, and arguments and types. This JSon packet would be received by the OSC proxy web-application which would then interpret the packet and determine if the specified hosts and specified paths were allowed to be relayed to other hosts. Once the packet was approved a real OSC packet was formed and sent to the appropriate host. The latency of the web interface thus varied because the latency between the web UI and the cloud granular synthesis slaves succumbed to a lot of jitter. Nonetheless this simple web-interface allowed smartphones and desktop web-browsers to control parameters on multiple machines and still hear it.

[2]Download the source code of the cloud slave granular synthesizer as well as it's Web UI front end http://ur1.ca/gl788
[3]Public Domain rendition at: http://ur1.ca/gl78i

5. DISCUSSION

5.1 Experience with the Cloud Sound Card

The cloud sound card [4] worked surprisingly well for such a simple setup. Audio generated by flipping an audio back and forth in mplayer suffered latencies far below 1 second, 200ms or less, when operating on the streaming web-server.

14 different slave audio generator clients would stream audio via netJack from a CSound based granular synthesizer time-stretcher instrument. The audio streaming over the web-client was arguably the lowest bandwidth connection for the sound server. The audio was uninterrupted and worked both on a desktop computer connected to cable internet, on a laptop connected to university wifi, on a Galaxy Nexus phone on university wifi, and on a Galaxy Nexus phone on cable internet wifi.

For some of the interactive aspects latency was noticeable due to all of the routing required, but it was often under control and interactive enough, but the latency would be too high for trained musicians.

An audio ping script was built that records the start time and sends a ping-request to the granular synthesizer to play a loud and brief high frequency sine-wave. When the `ScriptProcessor` played the buffer that contained the tone it would register that the ping was received and calculate the latency. The request path was an XHTTP request to an OSC relay Web App, then a UDP OSC message to slave VM running CSound operating a 100Hz control-rate, then CSound produces the ping noise, communicates this audio over NetJack back to the streamer web app, which then sends the audio over websocket back to the client's web browser and soundcard.

On the same machine a median of 78ms of latency was achieved. a median latency of 124ms was measured using the Cybera cloud, and a web-client on the University of Alberta network. A median latency of 165ms was measured using a consumer cable modem connection to the Cybera cloud. These latencies are far less than 300ms or 1 second, low enough for many kinds of interactive video-games, but too high for low latency interaction.

The bandwidth requirements depend on the configuration. For 1 channel of 44.1khz audio, represented by floating point numbers, one needs a minimum bandwidth capacity of of 1.5 MBit/s or 190kB/s. This includes packet headers for Ethernet, IP, TCP and HTTP Websockets. The same bandwidth would be needed for 16bit 2 channel audio.

Good performance (latency, audio quality) was achieved with buffer sizes of 256, 512, 1024, 2048, and 4096 samples. 256 floating point samples is less than the size of common

[4]Download the source code to the cloud sound card: http://ur1.ca/gl77s

DSL MTUs (~1468), meaning that if the buffer is set to 256 samples then the TCP packets sent will not be fragmented. The problem with a small buffer is that there are often more packets for less data.Large buffers can accommodate more noise and jitter in the transmission of data. But 256 samples is only 6ms of audio thus the cost of skipping it is very low. A buffer size of 4096 samples incurs approximately 100ms of latency, on top of all the other latencies incurred. Recommendation: use 256 samples to avoid packet fragmentation.

5.2 Experience with an Example Cloud Instrument

One problem with the cloud is the bandwidth and the routing of packets internally within the cloud. The routing of packets can affect how much data can be sent at one time to a particular host as well as the latency. The 15 node cloud instrument's master node was receiving about between 21 to 76Mbit/s of audio when 14 machines were streaming audio to it via netJack. That bandwidth from multiple hosts can be very costly in terms of network infrastructure and can cause packet drops and jittering.

One difficulty was that many applications expected a terminal or did not run well without a terminal. Running both jackd daemon and CSound processes within GNU Screen terminal emulator solved the need for a terminal. Orchestrating processes on many computers requires automation.

5.3 Recommendations

If one is building a cloud music instrument consider:

Make master and slave images. Images are stored copies of a VM, with an image a user can bring up new machines that are configured in a manner similar to other machines. When building many slaves it is preferable to build 1 slave image and then provision as many slave instances (VMs) as are needed. This saves configuration effort per each slave.

The configuration of the sound network: should you choose a star configuration, or a tree? After testing your cloud computers for connectivity one should be able to determine the maximum number or the acceptable of audio streams that a host can send or receive. If the number of usable VMs is less than the maximum number of streams then perhaps a star configuration will work. A tree configuration incurs more latency, as every time one accesses the network the latency increases as there are more marshaling costs and transmission. But in terms of scalability a tree formation, plus good quality network switches, can balance the stress on a network caused by streaming.

Computer problems multiply in the cloud. Another problem is that sometimes VMs crash or become unusable, one must adapt to an ever changing environment where every machine might not be available, or the machine is on a slow host and is lagging behind the other machines. Some distributed computing systems like Hadoop (`http://hadoop.apache.org/`) address this by allocating some redundant workers to duplicate some computations in the hopes of finishing or serving earlier. The same can said for the cloud instruments. Some redundancy might be worthwhile in case 1 VMs lags and can be replaced by another existing VM.

The cloud tends to be better at computation than latency. Consider building instruments that are computationally heavy but with soft real-time constraints. For instance video analysis, image analysis, or motion tracking could be outsourced to cloud machinery and then statistics and events generated could be aggregated into a larger instrument.

The bandwidth and latency are best within a single machine. Larger, more powerful cloud VMs might be a better than many cloud VMs. Computer music is heavily limited by latency thus taking advantage of immediate locality within a VM is far preferable to leaving the VM.

6. CONCLUSIONS AND FUTURE WORK

In conclusion, this work proposes, implements, describs and shares a method of listening to the output of cloud instruments, enabling performers to avoid carrying excessive equipment to a venue, and saving on setup time.

The various challenges that face computer musicians when using the cloud were discussed with potential solutions mapped out. An example implementation of a many-core cloud instrument was implemented and demonstrated.

Since Websockets are bidirectional it would not be difficult to stream audio back to the cloud from the performer's device. Bidirectional input would enable non-trivial collaborations without much infrastructure to set up. Cloud computing could better leveraged by breaking down synthesis computations into low latency Hadoop-like Map Reduce computer music networks.

7. ACKNOWLEDGMENTS

Thanks to Cybera (http://cybera.ca) for their support of this project with cloud computing resources. Thanks to Scott Smallwood for his support.

8. REFERENCES

[1] J. Allison. Distributed performance systems using html5 and rails. In *Proc. of the 26th Conf. of the Society for Electro-Acoustic Music*, 2011.

[2] J. Allison, Y. Oh, and B. Taylor. Nexus: Collaborative performance for the masses, handling instrument interface distribution through the web.

[3] Á. Barbosa. Displaced soundscapes: A survey of network systems for music and sonic art creation. *Leonardo Music Journal*, 13:53–59, 2003.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[5] A. Carôt, T. Hohn, and C. Werner. Netjack–remote music collaboration with electronic sequencers on the internet. In *Proceedings of the Linux Audio Conference*, 2009.

[6] L. Dahl, J. Herrera, and C. Wilkerson. Tweetdreams: Making music with the audience and the world using real-time twitter data. In *International Conference on New Interfaces For Musical Expression*, 2011.

[7] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455 (Proposed Standard), Dec. 2011.

[8] S. Jordà. Multi-user Instruments: Models, Examples and Promises. In *NIME'05*, pages 23–26, 2005.

[9] J. Oh and G. Wang. Audience-participation techniques based on social mobile computing. In *Proceedings of the International Computer Music Conference 2011 (ICMC 2011)*, 2011.

[10] C. Rorgers and W3C. Web Audio API. `https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html`, 2012.

[11] S. Smallwood, D. Trueman, P. R. Cook, and G. Wang. Composing for laptop orchestra. *Computer Music Journal*, 32(1):9–25, 2008.

[12] D. Trueman. Why a laptop orchestra? *Organised Sound*, 12(02):171–179, 2007.

[13] N. Weitzner, J. Freeman, S. Garrett, and Y.-L. Chen. massMobile - an Audience Participation Framework. In *NIME'12*, Ann Arbor, Michigan, May 21-23 2012.