Request for comments: Supercomputing for Raspberry Pi projects

Hello All,

My name is Charlie West. I am an electrical engineering graduate student at NC State University. I am working on a (soon to be open source project) to make it really easy for smalled internet connected projects to be able to offload computing for really hard things (like face recognition) to servers and desktops in a completely decentralized way. Hopefully, this will make it easier to build robots which can do things we generally have trouble with today.

I am very fond of command line programs because they are easy to write and easy to use in ways that the author didn't really plan for. There are some things it is hard for robots to have enough computing power to do quickly. I think it would be really nice if you could simply call a command line program to do something hard and in the background it would get a server to do the work. That way, there wouldn't be many limits on what a low power ARM robot could do.

Face recognition is one of the things that this could be useful for. A robot could track a face, take a still shot, and call a server command line program that would figure out who the face belonged to. The name would come back and the robot could say something like “Hello Jenny!”. Or if the face is someone new, the robot could say “Hey I'm HUB. I don't think I know you. If I may ask, what is your name?”. The robot could then stream its audio to a speech to text program and then store the name/face pair back to its face recognition service (Could result in a lot of people being named “What?”).

On a basic level, you could do some of this stuff using a home server and a daemon called inetd. Inetd lets you take a normal command line program on the server and call it from the network just by enabling inetd and editing a text file. In fact, you can use it to make and use a espeak text to speech service just using command line:

(On Ubuntu Server or other linux)
1. Install inetd, espeak
2. Register espeak as a service by editing the /etc/services file. I added the following:

```
espeak        9003/tcp    #A espeak server to use with inetd
```

3. Add the following line to /etc/inetd.conf

```
espeak   stream tcp  nowait root /usr/bin/espeak espeak --stdout
```

4. Figure out what the IP of your service computer is (where you installed inetd).

Or you can use the one I already set up on my network (Please don't crash my server). It is at www.visionmosaics.com

(On the Pi)
1. Make sure you have netcat, echo and aplay installed and your audio setup.
2. Run the following (hopefully with your own server IP, if not try mine). This streams the text to the server using netcat and then streams the server response to aplay to be heard:

```
echo "This was a triumph" | nc www.visionmosaics.com 9003  | aplay
```

If all goes well, you should hear the familiar espeak voice talking to you. Inetd is cool because it makes it really easy to use a normally command line program remotely. However, it isn't really perfectly suited for the grid computing/robot computing services I think would be handy.

Things to extend/change:
1. Normally inetd uses one port for each service and you have to edit sudo files to set it. It has a mux setting, but /etc/inetd.conf is still a sudo file.
2. Command line arguments for inetd services are defined in the configuration file. You cannot change it on the fly when you are using the service.
3. There is NO authentication or encryption. If you put the service on a public computer (like I have …) then there is nothing to stop other people from using or abusing it.
4. Right now at least, you need to roll your own server to setup and use program services. There isn't really any public services for hire (or for free).
5. There isn't much of an incentive to develop, polish or deploy new sorts of robot service programs, as their use would be limited.

So my project idea (some of which has been implemented with the source code on my site) is as follows:

Make a server program which can run in userspace (no root), uses one port, and eventually offer potential clients a list of services it provides, SSL encryption and identity authentication, server loading statistics and what it knows about other servers like it. This identity would be linked to a person or organization ID, so the client knows who it is dealing with. It could also be linked to a Bitcoin (or other digital currency) account and offer rates for its computing services (Such as .0001 BTC per minute of speech to text). The client can look and decide which server it wants to use for a service based on load, identity and cost. Given current minimum transaction fees, it would probably have a prepaid account with a well known escrow service or the server organization.

Make a client program that forwards the service identifier and service arguments to the server. Eventually, the client would support SSL encryption and authentication and potentially be associated with a Bitcoin (or other digital currency) wallet. Upon being called, it would take the service identifier and look for the best service to use (using rules from a configuration file). It then sends the server the service ID, arguments and stdin and returns the servers response as stdout. If you don't want to have to copy the hex code a lot, just use an alias for the client

From a clients perspective, the espeak example would look like this:

```
echo "This was a triumph" | netespeak --stdout  | aplay
```

Or alternately:

```
netespeak –stdout "This was a triumph"  | aplay
```

You could also adjust the voice using more command line arguments (unlike with inetd).

The netespeak alias would expand to:

HUBClient AE58940F8A7398E74DE9AB80FF66702D

The idea with tying in the virtual currency is to use supply and demand allocate servers and services. If servers with a particular services are being overloaded with requests, they can up the rates they are charging for the service. Hopefully, that will motivate other servers to load that services and start offering it. As demand for the service goes down in response to more servers coming online (or clients deciding it isn't worth it), the different servers can lower their rates to try to attract more customers. This drives server balancing and also provides a motivation for people to share servers in the first place. Likewise, if someone creates a hot new service that everyone wants they have server options to get rewarded for it (or decide not to). They can either license the software to servers (classic closed source) or just start running their own servers and be the only source for it (or they could opensource :)). Organizations which have their own servers could choose not to charge their robots (which all have unique IDs) or just only accept requests from their robots (and cycle the money back and forth).

There are many different architectures you could potentially build on top of this or hide beneath it. In many ways, it could be used to do BOINC or Condor style embarrassingly parallel problems really well. You could also offer services allowing desktops behind a NAT to act as subcontractors by relaying requests to them and paying for their service. That capability would allow pretty much any device connected to the Internet to be part of the grid (and be paid for its service).

If I may ask, what do you think of the idea? Are there particular services that you think would be useful to your projects? Do you think it is worth finishing?

Thanks,
Charlie West

P.S.
Code is on my website.

http://www.charlesrwest.com/software

Client negotiation, authentication and encryption isn't done yet, so it just connects to one hard coded server for now. However, it is rough but works and can be used like the example.