Load Balancing Voice Applications with Piranha[1]

Mustapha Hadim & Pierre Manneback
Faculté Polytechnique de Mons - INFO
Rue de Houdain 9,
7000 Mons Belgium
Tel: 00 32 65 37 40 58
Fax: 00 32 65 37 45 00

Michel Bagein & Pierre Maon
Faculté Polytechnique de Mons - TCTSLab
Av. Copernic, 1
7000 Mons Belgium
Tel: 00 32 65 37 47 36
Fax: 00 32 65 37 45 00

Abstract

In this paper, we investigate the load balancing problem among a cluster of mobile and fixed devices in a voice enabled interface. We consider a design approach. The voice interface has to support up to a hundred simultaneous users. The load balancing criteria we consider are defined, on the one hand in terms of network, CPU and memory resources, and on the other hand in terms of the boundary between fixed and mobile devices. The solution we propose is based on a derivate of the Linux Virtual Server: the Piranha system. Piranha enhances the Linux Virtual Server with several features, in particular with monitoring aspects and fault tolerance.

We describe precisely the proposed architecture of the application. We present a new scheduling technique which has been designed to take into account dynamically resource loads

Keywords. Load balancing, scheduling techniques, Piranha cluster, voice applications, distributed algorithms.

1. Introduction

There is a growing interest for voice technologies because of their flexibility in man/machine interfaces, as for wireless systems because of their mobility. In this work, we consider load balancing of a cluster of mobile and fixed devices, in a voice enabled interface. The voice interface has to support up to a hundred simultaneous users. Mobile devices are connected through a wireless network to a fixed server cluster. The client requests use TCP/IP and UDP/IP connections. Hence, we consider the problem of request dispatching between mobile devices and a cluster of servers, on a wireless network.

Existing request dispatching techniques for the client/server-cluster model can be classified into the following categories:

. **The client side approach.** Berkeley's Smart Client suggests that the service provide an applet running at the client side [1,2]. The applet makes requests to the cluster of servers to collect load information of all the servers, then chooses a server based on that information and forwards requests to that server. This client-side approach requires modification of client applications, so it cannot be applied to all TCP/IP services.

. **The server-side Round-Robin DNS approach** The RRDNS maps a single name to the different IP addresses in a round-robin manner so that the different clients access equitably the different servers in the cluster [3,4,5]. However, due to the different loads and requests, it easily leads to load imbalance among the servers.

. **The server-side application-level scheduling approach.** To build scalable web servers**,** the Reverse-proxy [6] and pWEB [7] forward the HTTP requests to different web servers in the cluster, get the results, and finally return them to the clients. The first request is between the client and the load balancer, and the second one is between the load balancer and the server. This could be a penalty on limited bandwidth network like wireless systems.

. **The server-side IP-level scheduling approaches.** Berkeley's MagicRouter [8] and Cisco's LocalDirector [9] use the Network Address Translation approach to make parallel service on different servers to appear as a virtual service on a single IP address. The load balancer changes the destination address of the request packets and forwards

[1] This work has been funded by the Walloon Region, Belgium, under the Programme “RW-Initiative”, DGTRE, 2002-03 (Convention n° 01/1/4866 , MODIVOC)

them to the chosen server, then changes the source address of the reply packets back to the original destination address. However, the MagicRouter and, the LocalDirector only support part of TCP protocol.

. **The Linux Virtual Server** (LVS in short, cf. www.linuxvirtualserver.org.) is a highly scalable and highly available server built on a cluster of real servers (see Figure 1). It is a free software platform built on top of Linux. The architecture of the cluster is transparent to end users, who see only a single virtual server. The front-end of the real servers is a load balancer, which schedules Internet TCP/IP requests, coming from requesting clients, to the different real servers according to a scheduling technique. The front-end make parallel services of the cluster to appear as a virtual service on a single IP address. The scheduling technique defines, for an incoming requesting client connection, to which real server is assigned the IP connection. There are several scheduling techniques available, and three IP packet-forwarding methods. These later are LVS via Network Address Translation, LVS via IP Tunneling (or IP encapsulation) and LVS via Direct Routing. These techniques are detailed in the LVS URL. Each technique has its own strength and weakness.

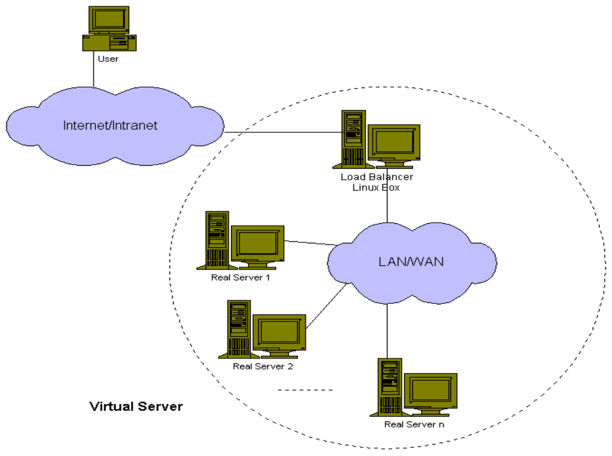
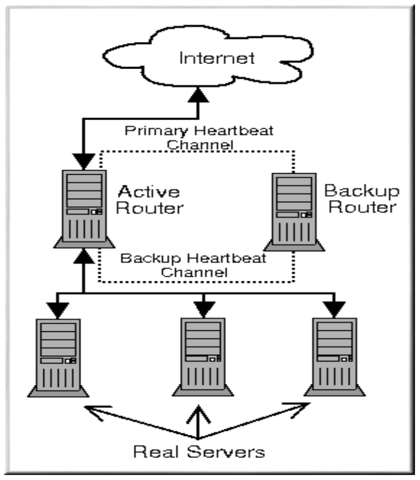


Fig. 1 : The Linux Virtual Server

We have chosen LVS as a starting tool to develop our load balancer. Indeed, this platform corresponds to the requirements of our application, particularly as it is based on Linux. However, its main problem is the failure of the load balancer. In case of such event, any connection is denied. This is why we have decided to use the Piranha system for developing our cluster. This later platform is a Redhat Inc. free software product. It is directly derived from LVS and is described at the URL: http://www.redhat.com/software/advancedserver/technical/piranha.html.

A Piranha cluster contains a pair of redundant load balancers or routers: an active one and a backup one. The active router plays two roles in the cluster: first it balances the load on the real servers. Second, it checks the integrity of the services on each of the real servers. The backup router's job is to monitor the active router and to supply in the event of failure. A Piranha cluster uses specifically the NAT IP packets



forwarding method. However, it offers all the scheduling techniques of the LVS. Figure 2 shows a simple Piranha cluster consisting of two layers.

Fig. 2: A Piranha Cluster

Service requests arriving at the cluster are addressed to a virtual IP address or VIP. This is a publicly advertised address the administrator of the site associates with a fully qualified domain name. The active router also dynamically monitors the overall health of the specific services on the real servers through simple send/expect scripts. To help in detecting the health of services that require dynamic data, such as HTTPS or SSL, the administrator can also call external executables. If a service on a real server malfunctions, the active router stops sending jobs to that server until it returns to a normal operation state. If a real server fails, the active router removes it from the pool of active servers. Piranha also offers a web-based interface for configuring and administrating a cluster. Moreover, as it is based on modular software architecture, it allows to easily including a new code or exchanging some parts of the existing code. Indeed, there are mainly three daemons that run in parallel. These later related Piranha issues are very important for our project and do not exist in LVS. This motivates our choice of Piranha.

In the next section, we briefly present the general software architecture of the voice enabled interface application. In section 3, we discuss a summary mapping of the software architecture onto the Piranha cluster. The design of a new Piranha scheduling technique is presented in section 4. Finally, we conclude in section 5.

2. Software Architecture.

In this section, we detail the design of speech technologies; namely, the Text-To-Speech (TTS), Automatic Speech Recognition (ASR), and dialog management processes, in the mobile environment. We also detail their integration in a 3-tiers architecture; namely, the data browser, the data presentation server, and the information management system (see figure 3).

2.1. Data browser

2.1.1. Embedded audio software. The minimal requirements for mobile audio devices are a microphone, a headset, a keypad and a wireless network adapter. At the software level, the operator interacts essentially with the application through ASR and TTS systems but the possibilities to record and play audio memos must also be available. Due to ergonomic and economic reasons, the embedded CPU and memory resources could be very limited in the mobile device. Then, the embedded software set cannot include all the components required by the browser instance.

A solution is to split the browser algorithms between mobile and fixed devices. In a first approach, we consider that only keypad handler, audio player, audio recorder, ASR front-end and TTS back-end are implemented in mobile devices. Then, we can consider that such mobile devices act as audio servers. The counterparts of ASR, TTS and the dialogue document browser are implemented on fixed devices. Each mobile runs a single instance of the embedded TTS/ASR software and this instance is connected, through socket mechanisms over the wireless network, to a single instance of a VXML [10] client. Audio data have real time aspects and they are transmitted with the RTP+RTCP/UDP/IP protocol. We consider also more powerful mobile audio devices. Different types of devices can carry more TTS/ASR blocks and then reduce the required resources in terms of network bandwidth and CPU/memory in fixed servers. The most powerful terminals can embed all voice components (ASR/TTS and dialogue VXML client). The voice-enabled interface must deal with every type of mobile audio devices.

The wireless network introduces new constraints. According to the current radio conditions (distance,

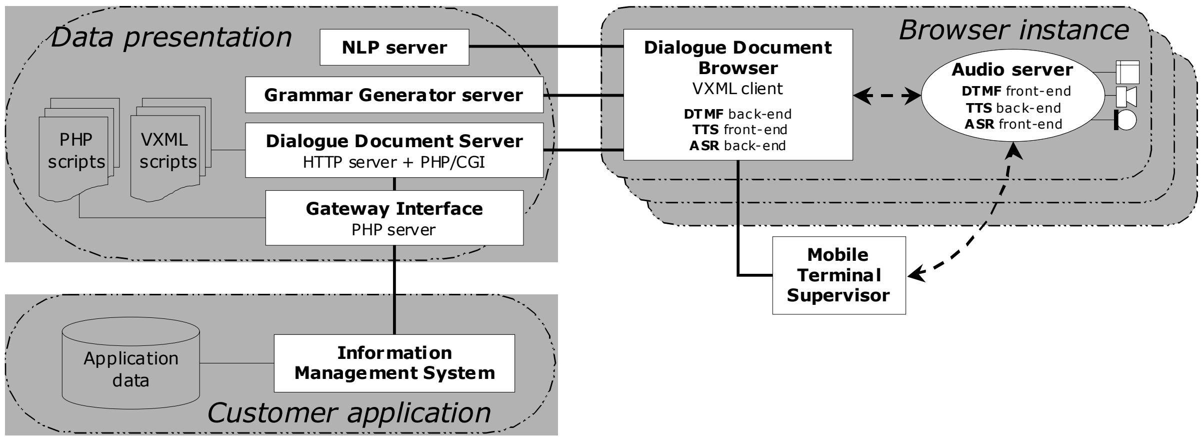


Fig. 3: Software architecture (dotted arrows represents data stream over wireless liaison)

obstacles, reverberation, etc.), the available bandwidth can severely degrade. As concrete applications can require up to several hundreds of simultaneous communications, the use of efficient compression algorithms (in terms of bandwidth an CPU requirement) is essential.

2.1.2. Dialogue document browser. The dialogue document browser parses VXML documents. According to the dialogue description, the browser can start the audio output dataflow (audio playing or TTS) and the audio input dataflow (audio recording, ASR or keypad event). The input result is analyzed to start a new action: jump to another point of current dialogue, run JavaScript action, load a new dialogue document, etc. We consider two types of audio output: audio memos and sentences. Audio memos are directly sent to the mobile device but in case of sentences, a TTS system is required.

It is possible to process sentences in a full TTS process and transmit the result as audio memos but it is more efficient in terms of bandwidth and CPU to split the TTS algorithms and transmit intermediate data over the wireless network. The TTS front-end acts as a layer using a translator between sentence and intermediate data. Once again, an optimization is possible by using the Natural Language Processing (NLP) resource shared by all TTS front-end algorithms, and, with this point of view, the NLP resource can be considered as part of data presentation (NLP server).

In the case of user input, before performing speech recognition, the ASR process must be initialized with a recognition grammar. As the NLP server, all ASR back-end algorithms can share a grammar generator resource, so the grammar generator can also be considered as part of data presentation.

2.1.3. Mobile Terminal Supervisor. When a mobile device becomes operational (switch on) and when it enters in the radio coverage (reaching wireless network dedicated to the vocal application), the embedded audio server requests the Mobile Terminal Supervisor to join the application. Then, the new mobile is registered and it can describe its characteristics, availability, and hints as to which dataflow flows can be served. Besides useful actions (login/logout and authorization management), the Mobile Terminal Supervisor must periodically check the current status of each mobile and detect connection losses. That status information is required to handle suspended dialogue mechanisms. An implementation of the Network Element Control Protocol [11] is used to perform this management task.

2.2. Data presentation

The vocal scenarios of the application are described in a set of dialogue documents, which respect the VXML formalism. The goal of the dialogue document server is to diffuse dialogue documents towards the VXML clients. Usual HTTP servers can be used for this task. Dialogue documents could be stored in a file system or could also be generated from information management system data. In this last case, a gateway interface (PHP server) should be used to generate queries and format the result inside a VXML template.

As said before, the Grammar Generator and the NLP server are considered as presentation servers and are included in the data presentation.

2.3. Customer information system

One of the main advantages of this architecture is the possibility to add a vocal interface without re-designing the customer Information Management System (IMS). All communications with the IMS are made through the dialogue document server that can support well-known gateway interfaces (PHP, Perl CGI, batch scripts,…).

3. Mapping the software architecture

Given the software architecture of the application, our objective is to distribute the charge fairly among on the one hand the mobile and fixed devices, and on the other hand within the server cluster.

The first aspect means the possibility to divide an application execution between the requesting mobile device and the server cluster. Namely, to define an execution boundary between mobile/fixed components. A first raised question is how to split the ASR and TTS processes between a back-end and a front-end devices. This boundary will depend on the mobile device's capabilities, in terms of vocal recognition and vocal synthesis. The second aspect means the possibility to distribute the load of several applications execution fairly among the servers of the cluster. We have to load balance the servers of the Data Presentation layer, the ASR back-ends and the TTS front–ends.

As the main software component of Piranha is executed on the router, we have included in its front the mobile terminal supervisor of figure 3 as a module. This module will allow establishing and managing a dialogue between an audio server and its associated VXML client. This dialogue uses the NECP protocol [11], and ensures some functions related to the mobile devices, such as:

- Detecting the arrival of a new mobile device;
- Detecting a mobile device exit.

As soon as a VXML client starts, it requests to the audio server to describe which dataflow (ASR front-end and TTS back-end) it supports; the boundary between fixed and mobile devices will then be well defined.

Once the decision regarding the execution boundary has been made, the part of the application that is executed on the virtual server is scheduled onto the real servers according to a new scheduling technique.

4. A new scheduling technique, DWLLS

Voice enabled interface applications needs a new scheduling technique. Indeed, as all the available Piranha techniques we have presented either lead to load imbalance in some corner cases or require to modify the client applications. We have thus designed a new Piranha dynamic scheduling technique that takes into account the charge of the nodes of the cluster in terms of their CPU and memory load. It also takes into account the number of active TCP/IP connections of each node. This information is collected with the NECP protocol, as each NECP_KEEPALIVE messages can contain one or more measure of performance queries.

As for the existing Piranha scheduling techniques, the Dynamic Weighted Least Load Scheduling (DWLLS) is based on the principle of weights assigned periodically to the nodes. These weights indicate the relative dynamic power of each cluster node. Indeed, periodically, the software module responsible for the nodes, which is executed on the active router, collects the state of each node in terms of its CPU load, memory load, and its number of active IP connections. This is done by using standard Linux monitoring tools. Then, for each node i, the active router computes a new weight w_i according to a given function of the above parameters, and assigns it to the real server. The function to evaluate w_i can be for instance a simple linear combination of CPU load cpu_i and memory load m_i, weighted by the node i nominal power p_i.

$$w_i = p_i\ (\lambda_1\ cpu_i + \lambda_2\ m_i),$$
$$\text{with } \lambda_1 + \lambda_2 = 1,\quad 0 < \lambda_1, \lambda_2 < 1$$

Supposing there are n nodes, each node i has weight w_i and alive connections c_i (i=1...n), the next network connection will be directed to the node j, such that

$$c_j/w_j = \min \{c_i/w_i\ , i=1...n\}.$$

We are currently implementing and validating this scheduling technique on a small configuration of fixed and mobile devices.

5. Conclusion

We have presented an approach for scheduling voice enabled interface applications using the Piranha system. In the current stage, we are working with a prototyped configuration. We are planning to install the developed architecture and the final product in a real environment.

The Piranha system overcomes the major problem of the proposed physical architecture; namely the potential failure of the active router. It also offers a web based interface and a cluster monitoring features. It was hence a well-founded starting point for our objectives.

We are including a protocol dialogue between the fixed virtual server and the mobile devices in order to define a boundary execution and to distribute applications execution between fixed parts and mobile ones. We have also sketched a new scheduling technique for the server cluster. This work has yet to be validated on real configurations and applications, but the preliminary tests encourage us in the choices of software architecture, tools and scheduling techniques.

6. References

[1] Chad Yoshikawa, Brent Chun, Paul Eastharn, Armin Vahdat, Thomas Anderson, and David Culler, "Using Smart Clients to Build Scalable Services", USENIX'97, 1997.

[2] Robert L. Carter, Mark E. Crovella, "Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks", Boston University Technical Report, 1996, http://www.ncstrl.org/.

[3] Eric Dean Katz, Michelle Butler, and Robert McGrath, "A Scalable HTTP Server: The NCSA Prototype", Computer Networks and ISDN Systems, pp155-163, 1994.

[4] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed, "NCSA's World Wide Web Server: Design and Performance", IEEE Computer, pp68-74, November 1995.

[5] T. Brisco, "DNS Support for Load Balancing", RFC 1794, http://www.internic.net/ds/

[6] Ralf S.Engelschall, "Load Balancing Your Web Site: Practical Approaches for Distributing HTTP Traffic", Web Techniques Magazine, Volume 3, Issue 5, May 1998.

[7] Edward Walker, "pWEB - A Parallel Web ServerHarness", http://www.ihpc.nus.edu.sg/STAFF/edward/pweb.html,Apr 1997.

[8] Eric Anderson, Dave Patterson, and Eric Brewer, "The Magicrouter: an Application of Fast Packet Interposing", http://www.cs.berkeley.edu/~eanders/magicrouter/, May 1996.

[9] A. Dahlin, M. Froberg, J. Walerud and P. Winroth, "EDDIE: A Robust and Scalable Internet Server", http://www.eddieware.org/, May 1998.

[10] “VXML: Voice eXtended Markup Language” - http://www.w3.org/TR/2003/CR-voicexml20-20030220/.

[11] A. Cerpa, J. Elson, H. Beheshti, A. Chankhunthod, P. Danzig, R. Jalan, C. Neerdaels, T. Shroeder and G. Tomlinson, "NECP: The Network Element Control Protocol", Work in Progress, http://www.circlemud.org/~jelson/writings/.