UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Sistema Baseado em Software Livre Para Reconhecimento de Fala em Nuvem em Português Brasileiro com Alta Disponibilidade

Bruno Gomes Haick

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém - Pará - Brasil
2013

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Sistema Baseado em Software Livre Para Reconhecimento de Fala em Nuvem em Português Brasileiro com Alta Disponibilidade

Autor: Bruno Gomes Haick
Orientador: Aldebaro Barreto da Rocha Klautau Júnior

Dissertação submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Pará para obtenção do Grau de Mestre em Engenharia Elétrica. Área de concentração: **Computação Aplicada**.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém, PA
2013

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Sistema Baseado em Software Livre Para Reconhecimento de Fala em Nuvem em Português Brasileiro com Alta Disponibilidade

AUTOR(A): BRUNO GOMES HAICK

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM 23/12/2013

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
(Orientador - UFPA / ITEC)

Prof. Dr. Jefferson Magalhães de Morais
(Membro - UFPA / ICEN)

Profa. Dra. Yomara Pinheiro Pires
(Membro - UFPA / Castanhal)

VISTO:

.....

Prof. Dr. Evaldo Gonçalves Pelaes

COORDENADOR DO PPGEE/ITEC/UFPA

Agradecimentos

Agradeço primeiramente à minha família, por todo o carinho e amor. Principalmente ao meu Pai Roberto Haick e minha mãe Cristina Haick, por dedicarem suas vidas à me ensinarem como ser uma pessoa de bons valores morais. Por serem capazes de priorizar meus objetivos ao invés dos seus e por sempre me incentivarem a lutar por tudo tudo que desejo na vida. E por fim agradeço por terem me guiado até este momento de minha vida.

Agradeço ao meu irmão, que mesmo com as brigas de moleques, esteve sempre por perto para ajudar no que fosse preciso.

À minha avó e tias por todo o apoio e carinho.

Aos meus colegas e amigos do Laboratório de processamento de sinais, por sempre estarem disponíveis a ajudar, por compartilharem conhecimento, ajudarem nas muitas dúvidas. Por tornarem o LaPS um ambiente colaborativo. E não poderia deixar de dizer que o LaPS é uma família. Obrigado.

Queria então, fazer um agradecimento especial, ao meu orientador Aldebaro Klautau, que além de um grande orientador, é "uma mãe", um grande amigo, alguém que sempre respeitou, ajudou, aconselhou e se preocupou comigo tanto pelo lado profissional quanto pelo lado pessoal, então eu agradeço por todas as oportunidades e ensinamentos que me deu.

Aos amigos da graduação e pós, que estiveram juntos, que direta ou indiretamente participaram desta conquista junto comigo, dando força ou mesmo contribuindo para este trabalho realizado.

Agradeço aos alunos do LaPS, Cássio Batista e Thiago Coelho, e ao professor Nelson Neto, pela contribuição para a realização deste trabalho.

Agradeço aos meus amigos Rafael Oliveira e Pedro Batista, por toda a ajuda, incentivo, torcida, para que hoje eu pudesse estar aqui, por estarem presentes mesmo que um pouco de longe neste momento em que estou concluindo mais uma conquista da minha vida.

Bruno Gomes Haick

Dedico este trabalho aos meus pais

Resumo

Este trabalho visa propor uma solução contendo um sistema de reconhecimento de fala automático em nuvem. Dessa forma, não há necessidade de um reconhecedor sendo executado na própria máquina cliente pois o mesmo estará disponível através da Internet. Além do reconhecimento automático de voz em nuvem, uma outra vertente deste trabalho é alta disponibilidade. A importância desse tópico se dá porque o ambiente servidor onde se planeja executar o reconhecimento em nuvem não pode ficar indisponível ao usuário. Dos vários aspectos que requerem robustez, tal como a própria conexão de Internet, o escopo desse trabalho foi definido como os softwares livres que permitem a empresas aumentarem a disponibilidade de seus serviços. Dentre os resultados alcançados e para as condições simuladas, mostrou-se que o reconhecedor de voz em nuvem desenvolvido pelo grupo atingiu um desempenho próximo ao do Google.

PALAVRAS-CHAVES: Alta Disponibilidade, Nuvem, Reconhecimento Automático de Fala

Abstract

This work aims to propose a solution containing an automatic speech recognition system in cloud. Thus, there is no need for a recognizer running on client machine itself since the same will be available via the Internet. In addition to cloud-based automatic speech recognition another aspect of this work is high availability. The importance of this topic is because the server environment where the recognition cloud will run must be available to the user. The various aspects that require robustness, such as Internet connection itself, the scope of this work was defined as free softwares that enable companies can use to increase the availability of their services. Among the results achieved under the simulated conditions, it was shown that the speech recognizer cloud developed by the group achieved a performance close to Google's.

KEYWORDS: High Availability, Cloud, Automatic Speech Recognize

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ASR - *Automatic Speech Recognition*

CETUC - *Centro de Estudos em Telecomunicações*

DRBD - *Distributed Replicated Block Device*

HA - *High Availability*

HMM *Hidden Markov Model*

HTTP - *Hypertext Transfer Protocol*

ICEN *Instituto de Ciências Exatas e Naturais*

ICMP - *Internet Control Message Protocol*

IP - *Internet Protocol*

MFCC *Mel-Frequency Cepstral Coefficients*

PB *Português Brasileiro*

RAID - *Redundant Array of Independent Disks*

SO - *Sistema Operacional*

TCP - *Tramission Control Protocol*

TIC - *Tecnologia de Informação e Comunicação*

TI - *Tecnologia de Informação*

UDP - *User Datagram Protocol*

UFPA *Universidade Federal do Pará*

WER *Word Error Rate*

xRT *Real-time factor*

Capítulo 1

Introdução

O uso de TIC (Tecnologia da Informação e Comunicação) está se intensificando na vida das pessoas e, principalmente, no contexto empresarial. As redes de computadores crescem de forma surpreendente, tornando-se parte do cotidiano das pessoas e empresas, aumentando significativamente sua influência direta na produção e desenvolvimento.

Todos os dias surgem soluções de grandes negócios, crescendo, resolvendo grandes problemas, de forma que este trabalho pretende abordar uma solução que ganhou muita força nos últimos anos, que é reconhecimento automático de voz.

Porém, torna-se cada vez mais necessário, manter muitos sistemas ou serviços capazes de atender as necessidades de um ambiente de produção, e com essa necessidade vem a dificuldade de manter este ambiente com seus serviços sempre disponíveis.

De forma que é notável que, no mundo em que vivemos onde a tecnologia se torna cada vez mais indispensável, em que a disponibilidade de recursos tecnológicos se torna cada vez mais crítica, o conceito de Alta Disponibilidade ganha bastante força, e se torna também indispensável neste contexto.

Desta forma, este trabalho visa propôr uma solução contendo um sistema de reconhecimento de fala automático em nuvem, ou seja, sem a necessidade de um reconhecedor estar sendo executado na própria máquina cliente, uma

vez que estará disponível através da Internet. Para garantir a disponibilidade desta solução, deve-se atentar para o conceito já mencionado de Alta Disponibilidade. Assim sendo, este trabalho será dividido em duas vertentes:

- Alta Disponibilidade
- Reconhecimento Automático de Voz em Nuvem

1.1 Objetivo

A proposta aqui apresentada consiste em configurar um sistema em que os serviços devem ser mantidos o máximo de tempo disponível. Sendo que em uma situação onde um determinado serviço, considerado crítico, esteja sendo monitorado em um servidor venha a ficar indisponível por algum motivo, um segundo servidor deveria assumir a responsabilidade de responder as requisições de clientes no lugar do servidor onde o serviço indisponível se encontra. Este procedimento deve ser realizado de forma automática, ou seja sem que seja necessária a intervenção humana.

1.2 Alta Disponibilidade

Alta disponibilidade (*High Availability*), não é um conceito novo para administradores de sistemas, principalmente quando trata-se de ambientes computacionais onde a disponibilidade é uma característica crítica. Esse conceito não surgiu recentemente, e está sendo amplamente disseminado e cada vez mais importante, assim como a influência que os sistemas computacionais exercem no mundo, na era da tecnologia.

Um sistema de alta disponibilidade tem como principal característica utilizar mecanismos de detecção, recuperação e mascaramento de falhas, objetivando suprir possíveis falhas de servidores em sistemas em produção, *on line*, de uma forma automatizada, e manter o sistema o máximo de tempo possível

funcionando, a fim de evitar que o cliente perceba ou sofra com alguma instabilidade no sistema, minimizando assim o tempo de indisponibilidade dos serviços fornecidos por este sistema, incluindo as situações de manutenção do sistema.

Muitos projetos *open-source* foram iniciados por desenvolvedores ou grupos de desenvolvedores, com intuito de encontrar soluções para problemas como o descrito nesse trabalho, dentre os quais os programas **Heartbeat** [1], o **Mon** [2] e o **Drbd** [3] se destacam. Estes têm o objetivo de resolver problemas de indisponibilidade de serviços, monitoramento de serviços e replicação de dados.

O sistema será considerado indisponível, caso um cliente não tenha acesso ao sistema ou a parte fundamental do mesmo. O tempo total de indisponibilidade do sistema é chamado de *downtime*.

Em qualquer cálculo de disponibilidade, não se contabiliza paradas programadas para manutenção, instalação ou atualização do sistema, ou parte do mesmo. Porém, é importante calcular o valor absoluto desta indisponibilidade, pois mesmo que pensando em um valor de 99 % pareça ser um bom nível de disponibilidade, isso significa um sistema 87,5 horas indisponível em 1 ano. Esse tempo não é aceitável em empresas de grande porte. Dessa forma, baseado no *downtime* mensal e anual, pode-se definir os níveis de disponibilidade apresentados na Tabela 1.1:

Tabela 1.1: Tabela de Níveis de Alta Disponibilidade.

Disponibilidade (%)	**Downtime / Mensal**	**Downtime / Anual**
95 %	1d 12h	18d 6h
96 %	1d 4h 48m	14d 14h 24m
97 %	0d 21h 36m	10d 22h 48m
98 %	0d 14h 24m	7d 7h 12m
99 %	0d 7h 12m	3d 15h 36m
99,9 %	0d 0h 43m 11.99s	0d 8h 45m 35.99s
99,99 %	0d 0h 4m 19.20s	0d 0h 52m 33.60s
99,999 %	0d 0h 0m 25.92s	0d 0h 05m 15.36s

1.3 Reconhecimento de Fala

Um dos maiores desafios dos pesquisadores no mundo, é criar tecnologias que auxiliem à melhoria entre o homem e a máquina, com intuito de que a utilização deste recurso seja natural. Desta forma, os sistemas de reconhecimento de voz vem ganhando espaço, já que por parte das pessoas a utilização da fala é muito natural, diminuindo assim a dificuldade destes se relacionarem com a máquina. Com o aumento da tecnologia, há também o aumento da capacidade de processamento, o que viabiliza cada vez mais a utilização deste recurso em várias aplicações, tais como: atendimento automatizado em centrais de atendimento por telefone, aplicativos em computadores, celulares, dentre outros.

Um grande avanço como este, contribui para a resolução de um grande problema que seria disponibilizar a utilização de recursos tecnológicos como computadores a deficientes físicos. Assim estes não precisam mais deixar de ter acesso a informação, ou a própria tecnologia.

1.4 Metodologia

A alta disponibilidade será implementada tanto em hardware como em software. Em hardware, consiste em aplicar redundância de servidores, onde para cada servidor contendo serviços disponibilizados na rede, haverá um servidor de backup, com dados replicados e serviços configurados igualmente. Em software, consiste em implementar e configurar os softwares **Heartbeat**, que tem como objetivo tomar decisões no sistema, **Mon**, que tem como função monitorar o estado dos serviços escolhidos como críticos, e **Drbd**, que tem como função a replicação de dados entre os servidores através de um enlace de rede.

Por exemplo, o **Heartbeat** foi utilizado nesse para controlar os recursos disponibilizados, ou seja, ele tem o poder de iniciar e parar um ou mais recursos, e comunicar a instância em outro servidor para que deve iniciar

seus serviços. Já o **Mon**, foi configurado para monitorar os mesmos recursos, sendo que se um serviço gera algum evento o mesmo é tratado acionando o Heartbeat. Por sua vez o **Drbd** tem a função de realizar o espelhamento dos dados entre os dois servidores através da rede, para que quando o segundo servidor venha assumir a responsabilidade pelas requisições dos clientes não haja inconsistência nos dados.

1.5 Organização do trabalho

Este trabalho é, portanto, assim dividido:

- **Capítulo 1 - Introdução**: Apresenta o assunto e o escopo do trabalho.
- **Capítulo 2 - Clusters de Alta Disponibilidade, Utilizando Ferramentas de Software Livre**: Explica os conceitos básicos necessários para melhor entendimento do trabalho, assim como indica outras soluções existentes no mercado.
- **Capítulo 3 - Reconhecimento de Fala em Nuvem**:
- **Capítulo 4 - Resultados** : Apresenta os resultados deste trabalho.
- **Capítulo 5 - Considerações finais**: Apresenta as considerações finais do autor.

Capítulo 2

Ferramentas de Software Livre para Alta Disponibilidade

Este capítulo tem o intuito de apresentar ao leitor as ferramentas utilizadas para a construção ou configuração da componente de Alta disponibilidade da solução proposta no trabalho. Assim como, será mostrado o funcionamento de cada ferramenta, e por qual motivo foram escolhidas, sendo que existem outras que podem solucionar o problema descrito neste trabalho.

A forma como foi configurado o sistema, como instalação, arquivos de configuração, configurações no Sistema Operacional, etc, podem ser encontradas no trabalho de conclusão de curso (TCC) [4] do mesmo autor deste trabalho.

2.1 Introdução

Os sistemas computacionais baseados em clusters seguem duas diferentes linhas da computação um pouco diferentes: **Alto Desempenho** (HP - *High Performance*) e **Alta disponibilidade** (HA - *High Availability*).

Clusters em geral são definidos como Sistemas Paralelos ou Distribuídos, constituídos por um conjunto de computadores interconectados, em que os recursos são unificados, utilizados como um único computador. Clusters HP são referenciados por seu Alto Desempenho e Escalabilidade, conhecidos

como Clusters de Estação de Trabalho (WSCs - *Workstation Clusters*) ou Rede de Estação de Trabalho (NOWs - *Network of Workstations*); Clusters HA possuem a perspectiva da Alta Disponibilidade.

Clusters de Alta Disponibilidade, podem ser compostos de duas ou mais máquinas (nós), dependendo do nível de desempenho exigido pelo ambiente computacional onde o mesmo está inserido.

Até alguns anos atrás, um nível alto de disponibilidade exigia geralmente o uso de hardware e software proprietários, afastando os usuários comuns desta tecnologia. Porém, com o avanço da comunidade de software livre, surgiram soluções alternativas. A solução é combinar computação altamente escalável e de confiança com os componentes *open source* disponíveis. Isso pode ser feito através dos Clusters. Para que isso seja possível, deve-se integrar dois sistemas idênticos conectados através de uma rede de computadores (Nós **primário e secundário**). O nó secundário possui uma "cópia"dos dados e processos do servidor primário, podendo então substituí-lo em caso de falhas.

Estes Clusters possuem as seguintes características marcantes:

- **Disponibilidade**: Capacidade de um sistema de manter-se disponível mesmo em caso de falhas.
- **Escalabilidade**: Capacidade de ampliação do número de nós conforme a necessidade do sistema.
- **Gerenciabilidade**: Capacidade de ser gerenciado como um sistema único.

2.2 Conceitos mais utilizados

Abaixo são explicitados alguns termos e conceitos que serão utilizados várias vezes neste trabalho, para que se possa facilitar o entendimento do trabalho.

Recurso: é o próprio objeto de trabalho dos mecanismos de alta disponibilidade, algo que se pretende manter disponível continuamente. Pode ser um serviço da rede, um sistema de arquivos, discos, processador entre outros.

Failover: quando um recurso de uma máquina apresenta uma falha, outra máquina do sistema deve assumi-lo, de forma transparente para os usuários. A máquina não precisa necessariamente parar por completo, basta que o recurso protegido pare.

Failback: é o oposto do *failover*, o elemento que falhou retoma seu estado normal e é colocado (manual ou automaticamente) de volta para trabalhar.

High Availability: Alta disponibilidade em inglês (também usado como HA).

Transferência de estado: quando uma máquina executa o *failback*, ela pode estar desatualizada em relação à que a substituiu. Ela então realiza uma transferência de estado para sincronizar os recursos de ambas as máquinas, mantendo a consistência.

Single point of failure (SPOF) ou ponto único de falha: é usado para identificar um recurso que não possui alta disponibilidade.

Uma falha em um SPOF comprometerá todo o sistema independentemente de quão protegidos estejam outros recursos.

Disponibilidade contínua: implica em serviço "*non-stop*", sem intervalos. Representa um estado ideal, e geralmente é um termo usado para indicar um altíssimo grau de disponibilidade, no qual apenas uma pequena quantidade de "*downtime*" é permitida. Alta disponibilidade não implica em disponibilidade contínua.

Tolerância a falhas: significa altíssima disponibilidade. Um sistema que tolera falhas tem a habilidade de continuar servindo independentemente de uma falha de hardware ou de software, e é caracterizado por redundância em hardware, incluindo CPU, memória e subsistema de I/O. Alta disponibilidade não implica em tolerância a falhas.

Cluster: agrupamento de recursos que fornecem ao sistema a ilusão de

um recurso único. Exige um mecanismo de gerência e permite alta disponibilidade, escalabilidade e alto desempenho. Por definição, implica em dois ou mais computadores juntos fazendo o serviço que normalmente uma máquina faria sozinha. Existem basicamente cinco tipos de clusters: HA (Alta Disponibilidade), FT (*Fault Tolerance*), MPP (*Massive Parallel Processing*), SMP (*Simetric Multiprocessing*) e NUMA *(Non-uniform Memory Access*). Essas categorias não são mutuamente exclusivas e devem ser consideradas características individuais que definem um cluster. De qualquer forma, uma delas se sobressairá.

Falha (fault): um comportamento inesperado do sistema em uma situação aparentemente normal, um desvio da especificação do sistema. Ocorre no universo físico.

Erro (error): manifestação visível da falha, que pode ser percebida pelo usuário/programador, porém, sem permitir a identificação da causa ou a localização da falha em si. Ocorre no universo lógico.

Defeito (fail): desvio da especificação, causado pelo erro. O sistema fornece respostas / reações incorretas. Ocorre no universo do usuário. A relação causal é: falha - erro defeito. Um erro é, portanto, a interpretação lógica de uma falha, e um defeito é uma manifestação de um erro percebida pelo usuário.

Parada planejada: quando o sistema é paralisado intencionalmente para manutenção, upgrade, ou para qualquer outra tarefa que exija o sistema fora de funcionamento.

Parada não planejada: quando o sistema sai do ar inesperadamente. São as paradas não planejadas que a alta disponibilidade tenta evitar.

2.3 Watchdog - Software

Um software *watchdog* é um processo que tem como objetivo monitorar processos em busca de erros. Esta tarefa é realizada geralmente com exatidão diferente, já que é necessário que o processo monitorado esteja ciente da

monitoração do *watchdog* e coopere com sua tarefa.

O *Watchdog* monitora a aplicação verificando se a mesma deixa de funcionar, se isso acontecer, pode determinar uma ação. Se o processo monitorado está configurado para a cooperar com o *watchdog*, a eficiência do mesmo aumenta. A cooperação com o *watchdog* pode ser realizada através de vários mecanismos, dentre elas o uso de *Heartbeats* e notificações de inatividade.

2.4 Heartbeats

O *heartbeat* é uma notificação dada de forma periódica e enviada pela aplicação para o *watchdog* com intuito de assegurar que esta permanece ativa. Isto pode consistir em uma aplicação inicializar uma mensagem "Eu estou vivo."- "I'm Alive.", ou um esquema de requisição / resposta no qual o *watchdog* faz uma requisição a aplicação - "Você está viva?"- "*Are you Alive?*", para confirmar se esta permanece ativa e aguarda por um reconhecimento. Em ambos os casos, quando o *timeout* especificado expira pelo lado do *watchdog*, este assume que a aplicação esteja indisponível. Então, toma como ação fechar o processo e reiniciar a aplicação. *Watchdogs* podem coexistir no mesmo sistema das aplicações protegidas ou em outro sistema. No primeiro caso, o *watchdog* fica sem utilidade se o próprio sistema falhar. Porém, se o *watchdog* estiver em outro sistema o mesmo irá detectar o problema.

2.5 Notificações de Inatividade

A ideia das notificações de inatividade é informar ao *watchdog* os períodos em que a aplicação fica inativa. O *watchdog* pode então tomar ações preventivas como simplesmente reiniciar a aplicação. Esta ação que parece sem sentido no primeiro momento, entretanto, decorrido um longo tempo da inicialização de um software, o seu desempenho pode degradar e apresentar ineficiência na utilização dos recursos computacionais que o suportam, tais como, estouro de memória, bloqueio indevido de arquivos, fragmentação de

dados havendo perda de integridade, etc. Assim então faz sentido a ação de reiniciar um serviço quando este apresenta algum problema em sua execução.

2.6 Soluções

Como citado anteriormente, a evolução de ambientes de Alta disponibilidade se torna cada vez mais rápida, com isso, surgem várias grandes soluções no mundo corporativo, tanto no âmbito de soluções **proprietárias**, como soluções baseadas em **software livre** e **open source**. A seguir, para que se possa entender os motivos da escolha da solução adotada neste trabalho, iremos citar e explicar as vantagens e desvantagens exemplificando com algumas soluções existentes no mundo do ambiente computacional.

2.6.1 Soluções proprietárias

Dentre as soluções proprietárias, podemos citar:

- **ARTICA**: Uma solução da empresa ARTICA Technology [5].
- **GalaxyVisions**: Uma solução da empresa GalaxyVisions [6].
- **Lifekeeper**: Uma solução da empresa [7]

A principal vantagem de se optar por uma solução proprietária, é que a manutenção do sistema fica por conta da empresa que fornece a aplicação, ou seja, quando se contrata um sistema proprietário, inclui-se também a manutenção e responsabilidade deste sistema.

Porém, a grande desvantagem é que esta aplicação só é fornecida mediante pagamento, ou seja, uma quantia de dinheiro é investida em soluções com esta desta natureza. E claro, devido a natureza deste ambiente, o custo é muito alto, ou seja, estas soluções geralmente são adotadas por grandes empresas, que possuem capital para tal adoção.

2.6.2 Soluções Livres

As soluções livres estão crescendo cada vez mais no âmbito comercial. Já que se mostraram capazes de manter os ambientes computacionais com grande desempenho, com um custo muito baixo, elevando então o lucro destas empresas de forma satisfatória.

A desvantagem encontrada em adotar sistemas livres, é que a manutenção do sistema fica a encargo da própria empresa, ou mesmo esta pode contratar uma terceira para tal. Mas de ante mão não há uma empresa responsável por possíveis *bugs* de software por exemplo. Tendo como grande exemplo de empresas que se mantém utilizando software livre com grande sucesso a empresa **Google**.

Dentre as soluções mais utilizadas no mercado podemos citar:

- **Lemuria** [8].
- **LVS - Linux Virtual Server** [9].
- **Heartbeat** [1].

Dentre as soluções citadas acima, a que mais se adequou ao problema descrito neste trabalho foi o uso do **Heartbeat**, já que o mesmo é capaz de implementar um nível de redundância maior que os outros. Sua documentação é bastante organizada, e seu desempenho mostrou-se muito bom para o ambiente proposto.

Desta forma, o leitor encontrará nas próximas seções a descrição e o funcionamento dos softwares utilizados para a configuração da solução de Alta Disponibilidade deste trabalho.

2.7 Replicação de dados

O *Drbd* é uma solução baseada em software livre, desenvolvida para formação de Clusters de alta disponibilidade. O papel deste sistema é a replicação ou espelhamento de dados de dispositivos de blocos (discos rígidos,

partições, volumes lógicos, etc.) entre servidores em uma rede de computadores, utilizando a rede para transmitir os dados.

Formas de espelhamento de dados:

- **Em tempo Real:** Enquanto as aplicações estão modificando os dados do sistema, a replicação está ocorrendo de forma contínua.
- **Transparente:** Do ponto de vista das aplicações, os dados são guardados em uma única máquina, ou seja, as aplicações não tem conhecimento de que seus dados estão armazenados em mais de um servidor.
- **Síncrono ou Assíncrono:** Utilizando espelhamento **síncrono**, quando uma aplicação que está realizando operação de escrita de dados, só será notificada que sua escrita foi realizada com sucesso, após os dados terem sido escritos em todos os nós que compõem o sistema. Este tipo de espelhamento se torna a melhor escolha quando se trata da implementação, ou aplicação de espelhamento em Clusters de alto desempenho, já que neste caso, não se permite a perda de dados, caso por algum motivo ocorra uma falha no nó ativo, ou principal. Utilizando espelhamento **assíncrono**, a aplicação é notificada que sua escrita foi realizada com sucesso, assim que a escrita tenha sido completada no dispositivo local, antes que os dados sejam enviados para os servidores. Já este tipo de espelhamento, se torna uma boa escolha em uma solução que se dê através de uma rede onde a distância entre os nós seja muito grande, ou seja, o RTT (Round trip time) seja maior que a latência de escrita de dados que a aplicação suporte.

2.7.1 Funcionamento

O *Drbd* Funciona no topo dos dispositivos de bloco, tais como partições de disco rígido, LVM [10] (*Logical Volume Manager*), etc. Desta forma, o *Drbd* referencia blocos de dispositivos, e foi desenvolvido com intuito de compôr soluções de alta disponibilidade. Isso é feito através da execução

de espelhamento dos dispositivos de blocos como um todo, através de uma determinada rede de computadores.

Na prática o mesmo funciona tal como um RAID [1] [11] 1 em rede. Todos os dados gravados em um determinado dispositivo de blocos (disco ou partição), são replicados para um outro dispositivo de blocos de forma automática, assim como acontece em um array RAID 1.

Neste caso, a diferença é que a replicação de dados não ocorre de um disco local para outra partição ou disco local, mas sim para um dispositivo de blocos que se localiza em outro nó ou servidor que compõe o cluster, em outro ponto da rede de computadores.

Na Figura 2.1, que foi retirada de [12], site oficial do projeto *Drbd* pode-se observar 2 blocos, onde cada um representa um servidor que compõe o cluster de alta disponibilidade. Cada um destes blocos é constituído por elementos ou componentes pertencentes ao kernel Linux [13], tais como: sistema de arquivos, drivers de disco, escalonador de disco, a pilha de comunicação TCP/IP [14] e placa de rede.

Pode-se observar na Figura 2.1, que a comunicação entre os servidores se dá através de dispositivos de rede dedicados a esta comunicação, e é através deste enlace que os dados dos serviços de alta disponibilidade são transmitidos do Servidor principal (nó ativo) do cluster, para o servidor secundário, que está em *standby*.

Também nota-se que o **Drbd** está localizado em uma camada acima do disco rígido, ou seja, os dados não serão escritos diretamente no disco, e sim irão acessar o dispositivo virtual criado pelo drbd, o qual irá se responsabilizar por realizar a tarefa de escrita destes dados no disco local, assim como enviar os dados para o dispositivo **Drbd** localizado no nó remoto do cluster, através do enlace de rede já citado anteriormente.

[1] Redundant Array of Independent Disks

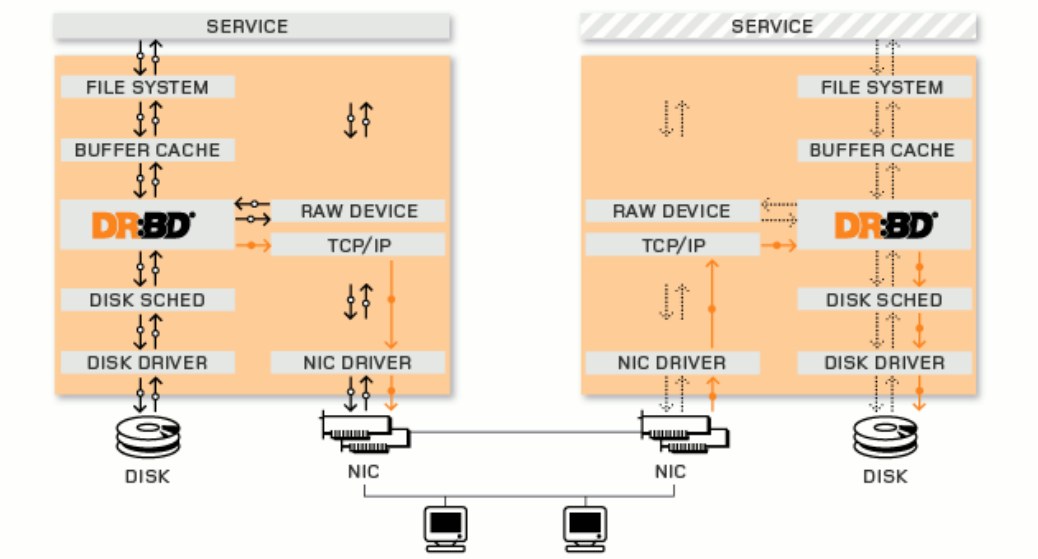

Figura 2.1: Funcionamento do Drbd. Fonte: http://www.drbd.org

2.8 Componente de Alta Disponibilidade - Heartbeat

O *Heartbeat* faz parte do projeto LINUX-HA (High-availability Linux), sendo o principal software do projeto. Ele é um programa licenciado sob a GPLv2 (GNU General Public License) [15].

Pode-se considerar este software como sendo o núcleo do ambiente de alta disponibilidade, já que possui a responsabilidade de monitorar os servidores em produção e, em casos de falha, deve automaticamente realizar os procedimentos para manter o funcionamento do sistema como um todo.

O servidor secundário do cluster, verifica a disponibilidade do servidor primário (em produção) enviando uma mensagem e exigindo uma resposta, sendo que a troca destas mensagens é feita através de um meio de comunicação, que pode ser Ethernet ou serial. A checagem se dá entre as instâncias do Heartbeat instaladas nos nós do cluster. No caso de o servidor não responder, o mesmo será considerado indisponível, desta forma o nó secundário inicia os serviços locais e assume responsabilidade pelos recursos do sistema do cluster, assim este se torna o nó primário do cluster, sendo responsável

também por responder as requisições dos clientes a partir deste momento.

Para ilustrar o funcionamento do Heartbeat, a Figura 2.2 mostra dois servidores (*Deathmask* e *Kanon*), nomes escolhidos pelo autor, com o Heartbeat instalado e configurado, além dos serviços desejados. Pode-se notar que os dois servidores são interligados por um cabo de rede dedicado, para que a comunicação entre os mesmos se dê de forma mais rápida. O *Heartbeat* cria uma interface virtual no servidor ativo do cluster, assim este endereço (10.57.2.9) é o endereço conhecido pelos clientes na rede, desta forma, quando um novo nó se tornar o nó ativo, os clientes não percebem a indisponibilidade de um dos nós. Assim, na visão dos clientes, o sistema está 100 % disponível.

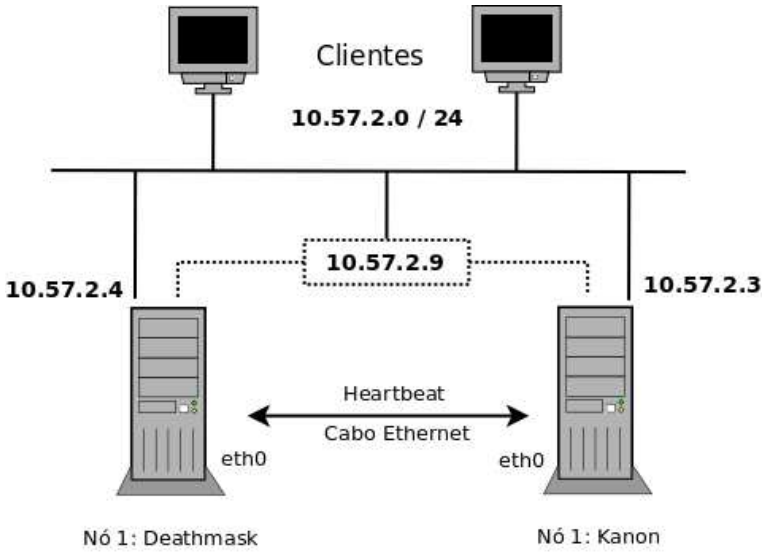

Figura 2.2: Funcionamento do Heartbeat.

Principais características:

- O número de nós do cluster não é fixo, ou seja, ele pode ser utilizado para a construção de clusters muito grandes assim como relativamente simples.
- Monitorar recursos, sendo capaz de mover ou mesmo reiniciar estes para um outro nó em caso de falha, de forma automática.

- Possui mecanismos de proteção, para remover nós do cluster em caso de falhas.
- Gerência de recursos baseado em diretivas, inter-dependência de recursos e restrições.

Um exemplo da utilidade do *Heartbeat*: anteriormente, explicamos como instalar, configurar, ativar e testar o *Drbd*. Porém, podemos perceber que indicar qual nó do cluster é reconhecido como nó primário na visão do *Drbd*, foi uma tarefa realizada manualmente, através de comandos específicos desta ferramenta.

O *Heartbeat* permite que esta tarefa seja executada de forma automática, possibilitando que a indicação de qual nó se tornará primário em um determinado momento, tendo como base a checagem da disponibilidade de um nó do cluster, ou conectividade de rede do próprio com os outros *hosts* da rede.

2.9 Monitorando serviços

O **Mon** [3], é uma ferramenta muito importante que tem como função monitorar a disponibilidade de serviços sejam estes locais ou remotos e enviar alertas referentes a eventos pré-determinados. Definimos como serviços, qualquer software ou programa sendo testado por um **"monitor"**, que poderia ser algo simples como por exemplo executar um ping em uma máquina ou sistema, ou algo mais complexo, como por exemplo analisar os resultados gerados por uma aplicação.

Os alertas podem ser configurados de acordo com a necessidade do administrador, dentre as possibilidades de alerta estão as seguintes:

- Alertas através do envio de emails.
- Alertas através da adição de uma tarefa ou ticket em um sistema gerenciador de tickets, ou tarefas. Tendo como exemplo destes sistemas o software Bugzilla.

- Alertas através da chamada do Heartbeat com intuito de gerenciar falhas ("failover") no cluster ativando o outro nó do cluster.
- Alertas utilizando mensagens SMS para um número de celular.

Os "monitores" e alertas, podem ser escritos em muitas linguagens, como por exemplo Java, Python, Bash, e facilmente incorporados ao **Mon**.

O **Mon** é um software *open Source* [16], e distribuído sob a GPLv2.

Como já visto no capítulo anterior, o *Heartbeat* não é capaz de iniciar o processo de **"failover"** dos recursos do cluster, ou seja, recuperação de falhas, por sí só. Desta forma surge a importância da utilização do **Mon**, já que o mesmo é capaz de verificar o estado atual de um determinado serviço, e caso o mesmo gere algum evento, tal como finalizar com erro, pode-se ativar o *Heartbeat* e assim iniciar o processo de passagem de recursos de um nó para o outro.

Tendo em vista que o **Mon** é utilizado na hora de monitorar eventos, ele se torna o único software que compõe a solução proposta, que não tem de ser obrigatoriamente implementado. Porém como nossa solução necessita monitorar os recursos oferecidos pelo cluster, então temos a necessidade de implementar o mesmo.

O **Mon** toma como base os conceitos de **"monitor"** e **"alert"**. Define-se **"monitor"** como sendo um checagem a ser realizada, isto é, um script de checagem, que como foi dito anteriormente, pode ser desenvolvido em qualquer linguagem, tanto que consiga retornar 0 (zero) no caso de finalizar com sucesso, e um valor diferente de 0 (zero) caso finalize com um erro qualquer.

E define-se **"alert"** como um script que será executado quando existir uma situação em que o **"monitor"** retorne um valor diferente de 0 (zero), ou seja, uma situação em que a checagem feita pelo **"monitor"** finalize com erro.

É importante atentar para o fato de que se o valor de retorno das checagens diferentes de 0 serão considerados como erros, deve-se então criar scripts

de checagem de forma organizada e inteligente para que os mesmos não gerem erros desnecessários, e sim somente quando o recurso estiver realmente indisponível.

Capítulo 3

Reconhecimento de Fala em Nuvem

Este capítulo tem o intuito de apresentar ao leitor o que são e para que servem os sistemas para reconhecimento automático de fala (ASR) [17]. Também serão apresentadas as formas de avaliação e as ferramentas para desenvolvimento utilizadas para melhoria do processo de reconhecimento de voz. Sem esquecer é claro, de apresentar o cenário utilizado para realizar o reconhecimento de fala em nuvem.

3.1 Breve Histórico

De acordo com [18], os sistemas ASR têm sido estudados desde os anos 50 [17–19]. Ainda por volta dos anos 50, havia outros estudos ou projetos sendo desenvolvidos. Então surgiu o primeiro reconhecedor de dígitos isolados com suporte a um único locutor, desenvolvido nos laboratórios Bell. Assim como também surgiu a ideia de utilização do conceito de redes neurais em sistemas ASR, o que não foi em frente devido problemas práticos. Já por volta do 70, ganhou muita força a utilização da técnica *dynamic time-warping* (DTW) [20] (alinhamento temporal dinÂmico), que mede a similaridade entre duas sequências após alinhá-las ao longo do tempo.

A princípio, como pode ser observado em [21], os sistemas ASR tinham

como foco principal a aplicação de regras gramaticais e sintáticas à fala. Desta forma, os sistemas conseguiriam determinar as palavras que obedecessem as regras pré-determinadas. Porém estes sistemas não obtiveram grande sucesso, já que existem algumas características que se tornaram obstáculos, tais como sotaques e regionalismos.

Já na década de 80, surge a utilização de métodos estatísticos para resolver problemas de reconhecimento de palavras conectadas, tendo como mais utilizados os modelos ocultos de Markov (HMMs) [22, 23].

Atualmente,tem-se como estado da arte a representação dos sistemas ASR através da utilização de HMMs. Aumentando sua eficiência têm-se técnicas como aprendizado discriminativo [24], seleção de misturas de gaussianas [25], dentre outras. Existem além dos já citados, sistemas híbridos, utilizados para o reconhecimento de fala [26–29], sendo estes baseados na utilização de HMMs e redes neurais artificiais.

3.2 Reconhecimento automático de voz (ASR) - Fundamentação básica

Sistemas ASR utilizam métodos estatísticos baseados em modelos de Markov (HMMs), sendo compostos por cinco blocos, mostrados na Figura 3.1.

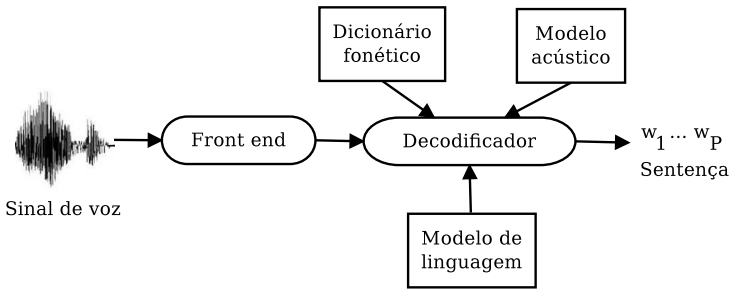

Figura 3.1: Principais blocos de um tipico sistema ASR.

Entende-se que os sistemas ASR possuem dois principais modos de funcionamento: comando e controle e ditado. No primeiro caso, têm-se um sistema que possui um modelo de linguagem com uma gramática livre-do-contexto, a

qual possui um número restrito de palavras aceitas. Já na segunda aplicação, o modelo de linguagem é probabilístico (N-gramas) e tipicamente pode aceitar dezenas de milhares de palavras.

3.2.1 Front End

Esta é a primeira fase do processamento de voz, cuja responsabilidade é parametrizar as amostras de voz oriundas da conversão analógico/digital.

Neste processo, ocorre a segmentação do sinal em curtos segmentos, chamados de janelas ou "*frames*", que podem variar entre 20 e 30 milissegundos, com um deslocamento da janela de análise geralmente de 10 milissegundos. Desta forma, para cada *frame*, calcula-se em um vetor de parâmetros $\mathbf{x}$ de dimensão L (tipicamente, $L = 39$). Assume-se que T *frames* estão organizados em uma matriz $\mathbf{X}$ de dimensão $L \times T$, o que representa uma sentença completa.

Há algumas alternativas para a parametrização do sinal de voz [17,30,31]. Porém a análise dos *Mel-Frequency Cepstral Coefficients* (MFCCs) prova ser suficiente [32] em muitos casos, e geralmente é aplicada na construção de sistemas ASR.

3.2.2 Modelo acústico

O modelo acústico tem por função representar o sinal de voz (sinal acústico) utilizando um modelo matemático criado através de *features* extraídas do sinal. Desta forma, no reconhecimento de palavras isoladas ou fonemas, segmentos desconhecidos podem ser mapeados de maneira correta em palavras.

Uma cadeia de Markov [22] consiste em uma máquina de estados finita, que pode modificar seu estado a cada unidade de tempo. Resumidamente, pode-se dizer que uma cadeia oculta de Markov, consiste em um modelo matemático, sendo este modelo formado por uma cadeia de estados conectados entre si, sendo que a cada transição entre estados existe uma probabilidade de

ocorrência associada. Existe também um processo estocástico ligado a cada estado, este processo é conhecido como processo de observação de saída, e pode ser de duas formas: discreto ou contínuo. Este processo de observação de saída, representa uma ocorrência do fenômeno sendo modelado. Nos sistemas ASR, é adotada uma topologia chamada "left-right" para estruturar as HMMs, na qual somente são permitidos dois tipos de transição de estado: De um estado para ele próprio, ou para o estado seguinte.

Apesar de possível, a abordagem que define uma HMM para cada fonema (monofone) representado no modelo acústico não representa uma boa modelagem para fala. Devido ao fato dos articuladores do trato vocal não se moverem de uma posição para outra imediatamente na maioria das transições de fones, se faz necessário o uso de estratégias que levem em consideração essa característica do trato vocal. Nesse sentido, durante o processo de criação de sistemas que modelam a fala fluente, busca-se um meio de modelar os efeitos contextuais causados pelas diferentes maneiras que alguns fones podem ser pronunciados em sequência [33]. A solução encontrada é o uso de HMMs dependentes de contexto (trifones), que modelam o fato de um fone sofrer influência dos fones vizinhos. Por exemplo, supondo a notação do trifone a-b+c, temos que b representa o fone central ocorrendo após o fone a e antes do fone c. Essa é a notação tipicamente utilizada no HTK.

Segundo [17], existem basicamente duas estratégias para a criação de modelos trifones: *internal-word* e *cross-word*. As diferenças entre as mesmas consiste em que, no caso da *internal-word* as coarticulações que extrapolam a duração das palavras não são consideradas, sendo assim, menos modelos são necessários. Já no caso do *cross-word*, a modelagem é mais precisa, visto que a mesma considera a coarticulação entre o final de uma palavra e o início da seguinte. Contudo, com a utilização dessa estratégia o número de modelos trifones gerados cresce muito, o que dificulta o trabalho do decodificador e gera uma necessidade de mais dados para treino. Alguns exemplos de transcrição podem ser conferidos abaixo:.

Os exemplos utilizam a frase: **arroz com bife**.

- Monofones:
  sil a R o s k o∼ b i f i sil

- Internal-Word:
  sil a+R a-R+o R-o+s o-s k+o∼ k-o∼ b+i b-i+f i-f+i f-i sil

- Cross-Word:
  sil sil-a+R a-R+o R-o+s o-s+k s-k+o∼ k-o∼+b o∼-b+i b-i+f i-f+i f-i+sil sil

Quando se trabalha com modelos acústicos baseados em trifones [17], o número de HMMs cresce bastante, o que gera um grande problema bastante conhecido de modelagem, que é a insuficiência de dados para estimar os modelos. Existe o método de compartilhamento, utilizado para combater este problema, e melhorar a qualidade dos modelos. Existem algumas técnicas para compartilhamento de estados, sendo uma delas conhecida como *data-driven* [17],na qual ocorre a clonagem dos monofones seguida pela conversão para trifones, por fim todos os estados centrais dos trifones derivados do mesmo monofone são vinculados conforme a Figura 3.2. Nesta abordagem, assume-se que o contexto do trifone não afeta o estado central do modelo.

Outra técnica para compartilhamento de estados, descrita em [34], consiste no uso de uma árvore de decisão fonética para unir os estados que são acusticamente semelhantes. Esse método envolve a construção de uma árvore binária utilizando um procedimento de otimização sequencial *top-down*, onde questionamentos são anexados a cada nó. As perguntas são relacionadas ao contexto fonético dos fones vizinhos ao fone analisado, como ilustra a Figura 3.3. Por exemplo, as questões são do tipo “o fonema à esquerda é nasal?”, e dependendo da resposta (sim/não), uma das possíveis direções é seguida. Ao final do processo, todos os estados no mesmo nó folha são agrupados.

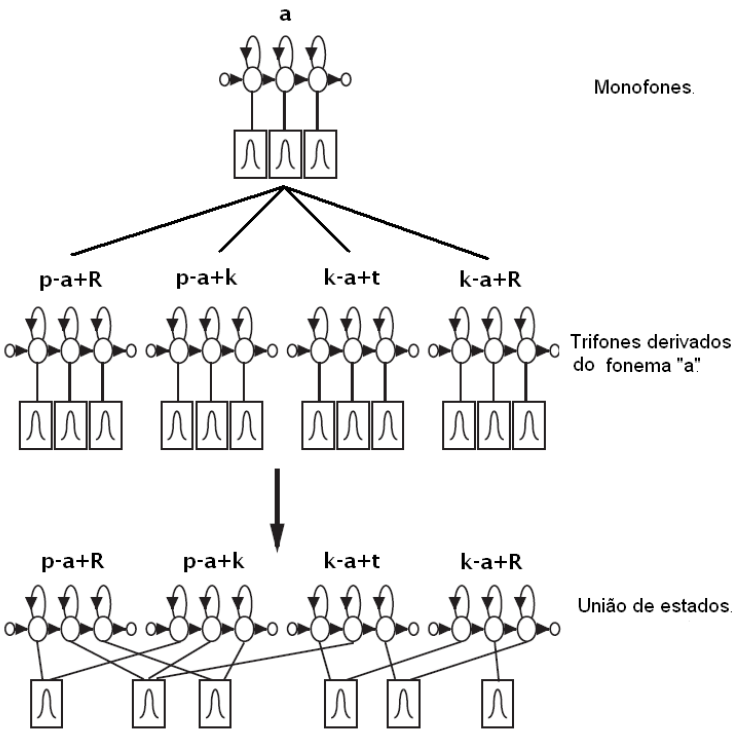

Figura 3.2: Compartilhamento de estados para fins de aumento da robustez da estimativa dos modelos HMM.

3.2.3 Modelagem da língua

Para que um sistema ASR tenha condições de se comportar de forma satisfatória, ou seja, com um bom desempenho, é necessário que este possua informações sobre a estrutura do idioma utilizado, sendo que tais informações podem ser fornecidas utilizando gramáticas livres de contexto ou modelos de linguagem.

Gramáticas livre de contexto

A gramática cria as regras para delimitar o que o sistema pode reconhecer, sendo esta gramática composta por variáveis e expressões regulares, criando assim o universo de palavras que podem ser reconhecidas, e até mesmo em qual ordem. O sistema reconhecedor por sua vez deve procurar o padrão

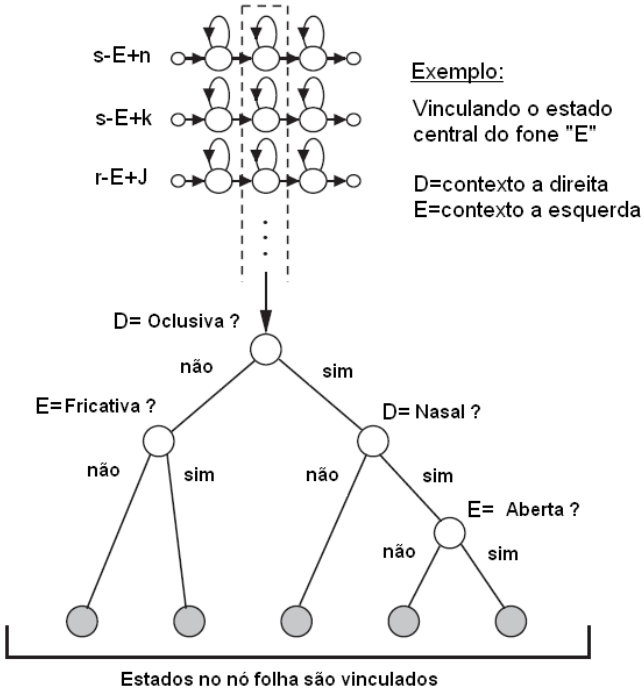

Figura 3.3: Compartilhamento de estados utilizando árvore de decisão fonética.

que mais se aproxima da entrada desejada, ou seja, este procura dentro da gramática fornecida, a regra criada que mais se assemelha com a entrada fornecida. As gramáticas tornam o sistema mais simples, já que o universo de palavras aceitas é restrito.

Modelos de linguagem

Existem casos em que é necessário se trabalhar com um grande volume de palavras no vocabulário, e neste caso trabalhar utilizando gramáticas livres de contexto se torna inviável, já que seria uma tarefa que exigiria um esforço muito grande e um longo tempo, tornando a tarefa impraticável. Desta forma, segue então o uso dos modelos de linguagem. Os modelos têm como função caracterizar a língua, para isso, tenta obter as regularidades, verificando de acordo com as regras gramaticais da linguagem, do idioma, a

validade das frases. De acordo com [35], utilizar modelos de linguagem gera um ganho de desempenho considerável.

De modo geral, um modelo de linguagem estima, de uma forma confiável, a probabilidade de ocorrência de uma determinada sequência de palavras $W = w_1, w_2, ...w_k$, onde k é o número de palavras na sentença. A probabilidade $P(W_1^k)$ é definida na equação 3.2

$$P(W_1^k) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_k|w_1, ..., w_k - 1) \quad (3.1)$$

$$P(W_1^k) = P(w_1)\prod_{i=2}^{k} P(w_i|w_1, ..., w_i - 1) \quad (3.2)$$

3.2.4 Decodificador

A decodificação é a ultima etapa do processo de reconhecimento de voz, sendo que esta consiste em transcrever as amostras da fala (voz), que são desconhecidas, para a forma de texto. O processo de decodificação é controlado por uma rede de palavras, sendo esta construída a partir do modelo de linguagem. A rede é composta por um conjunto de nós conectados, sendo que estes nós são as palavras, e cada conexão possui uma probabilidade de ocorrência, sendo estas as transições. Após se obter a rede construída, o dicionário de fonemas e o modelo acústico, pode-se então determinar a probabilidade de se percorrer qualquer um dos possíveis caminhos da rede, dada uma amostra de voz desconhecida. Então a tarefa do decodificador é encontrar os caminhos mais prováveis dentro dos possíveis.

3.2.5 Métricas de avaliação

Após a etapa de decodificação vem a análise dos resultados. A medida de desempenho utilizada na maioria das aplicações que usam reconhecimento de voz é a taxa de erro por palavra (WER). Como tipicamente as transcrições correta e reconhecida possuem um número diferente de palavras, as mesmas

são alinhadas através de programação dinâmica [36]. Dessa forma, de acordo com [17], a WER pode ser definida como:

$$WER = \frac{S + D + I}{N} \tag{3.3}$$

onde N é o número de palavras na sentença de entrada, S e D e I são o número de erros de substituição, deleção e inserção na sentença reconhecida, respectivamente, quando comparada com a transcrição correta.

Outra métrica de avaliação é o fator de tempo-real (xRT). O fator xRT é calculado dividindo o tempo que o sistema despende para reconhecer uma sentença pela sua duração. Assim, quanto menor for o fator xRT, mais rápido será o reconhecimento.

3.3 Cenário para Reconhecimento de voz em Nuvem

Após descrever um pouco o funcionamento do reconhecimento de voz, aborda-se o sistema desenvolvido. Este trabalho tem por objetivo criar um cenário para que possamos utilizar esses conceitos de outros lugares fora do contexto local, utilizando vários dispositivos, tais como, computadores, celulares, tablets, sendo que estes devem ter acesso à internet, dado que essa tecnologia é necessária para acessar o sistema remotamente.

Utilizando-se dos conceitos de Alta Disponibilidades apresentados anteriormente neste trabalho, foi criado um cenário, no qual tem-se como grande foco a disponibilização de um sistema de reconhecimento de fala, sendo que este serviço deve estar sempre disponível. Para tal cenário que é apresentado através da Figura 3.4 utilizou-se a estrutura do laboratório de Processamento de Sinais (LaPS) da Universidade Federal do Pará, o qual possui um domínio **www.laps.ufpa.br** que dará acesso ao servidor de reconhecimento.

Ao longo deste trabalho foram utilizadas várias ferramentas afim de compor a solução para o problema proposto. Todas as ferramentas utilizadas são

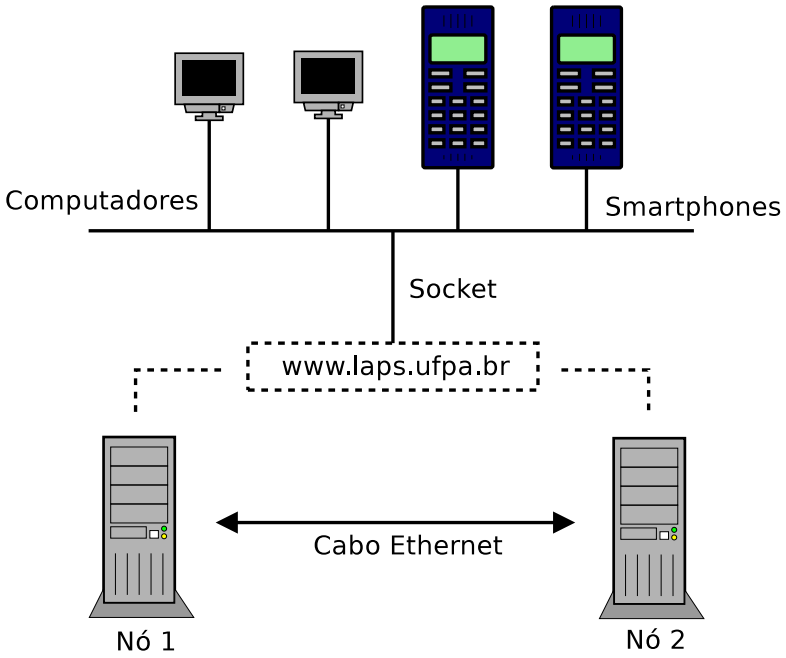

Figura 3.4: Cenário utilizado para reconhecimento de voz em Nuvem

baseadas em software livre e *opensource*.

As ferramentas utilizadas aqui podem ser divididas em dois lados deste cenário, o lado da construção e composição da nuvem, e o lado da solução que compõem a vertente do reconhecimento de fala.

3.3.1 Alta disponibilidade

O **Drbd** foi utilizado para sincronização dos dados entre os servidores internamente, já o **Heartbeat** foi utilizado para gerenciar as ações do cluster de alta disponibilidade, também houve a utilização da ferramenta chamada **Mon**, que tem como intuito monitorar os recursos que compõem a solução afim de saber quando algum recurso está com problemas.

3.3.2 Reconhecimento de voz

O software Julius [37] foi utilizado para prover reconhecimento de fala em modo distribuído. Para que o reconhecimento em Português Brasileiro

(PB) seja possível, o Julius utiliza algumas ferramentas disponibilizadas pelo grupo de pesquisa *FalaBrasil* da Universidade Federal do Pará, como modelo acústico [38] e dicionário fonético [39].

O modelo acústico foi treinado com 3 bases de áudio contendo arquivos no formato RIFF WAV da Microsoft. No total, há 111.078 arquivos com codificação PCM linear, 16 bits por amostra, *little-endian* e 8 kHz de taxa de amostragem, totalizando 166 horas e 10 minutos de gravação. O número de gaussianas por misturas usado foi 8 e o *software* utilizado para o treinamento foi o HTK [40].

O corpus CETUC

O Centro de Estudos em Telecomunicações (CETUC) [41], através do Professor Doutor Abraham Alcaim, gentilmente cedeu ao LaPS esse corpus de áudio para PB, sendo este composto por 100.998 arquivos de 101 locutores, totalizando aproximadamente 144 horas e 39 minutos de áudio.

O corpus LaPSStory

Desenvolvida em [38], a LaPSStory é uma base de áudio para PB composta por 2.099 arquivos de áudio extraídos de *audiobooks*. O corpus totaliza aproximadamente 15 horas e 41 minutos de áudio.

O corpus Spoltech

Base construída pela Universidade Federal do Rio Grande do Sul, Universidade de Caxias do Sul e Oregon Graduate Institute (USA), formada por áudios oriundos de leitura e resposta à perguntas de 477 locutores. O corpus contém 7.199 arquivos, totalizando 4 horas e 20 minutos de gravação.

O Julius foi configurado para operar em modo servidor através de seu próprio módulo de comunicação tcp/ip (adinnet), e assim está apto a aceitar requisições através de uma rede de computadores utilizando *sockets* [42] numa dada porta para tal comunicação.

Uma gramática de testes com 235 sentenças também foi construída no formato do Julius, contendo nomes de ruas e de alguns pontos da cidade de Belém do Pará.

Capítulo 4

Resultados

Este capítulo abordará os resultados obtidos neste trabalho. Para melhor entendimento este será dividido em duas seções:

- Alta Disponibilidade
- Reconhecimento de Fala

4.1 Alta Disponibilidade

Esta seção tem por objetivo mostrar os resultados obtidos utilizando um único serviço executando nos servidores que compõem a nuvem utilizada para fornecimento dos serviços necessários para a realização deste trabalho. O serviço utilizado foi o **Apache**, que é um servidor Web, desta forma, podemos demonstrar a disponibilidade do servidor, sempre que necessário.

Definimos uma simulação, onde o servidor de nome **kanon** inicia como servidor primário e o **deathmask** como secundário. Como já informado anteriormente, o **kanon** possui endereço IP 10.57.2.3, já o **deathmask** possui IP 10.57.2.4. O IP do cluster, é 10.57.2.9, sendo que os clientes na rede, só conhecem este endereço.

Para monitorar os serviços do cluster, foi utilizado o software Zabbix [43]. Este software, é um software Livre e *opensource*, que tem como intuito moni-

torar serviços, gerar alertas, etc. Desta forma, a partir deste foram gerados gráficos que demonstram a disponibilidade de cada serviço com o passar do tempo. Porém para esta demonstração, foram escolhidos apenas dois serviços: **Apache** (Servidor Web) e Julius (Reconhecedor de Fala disponível em nossa nuvem), e este serviço fica "ouvindo" requisições na porta 5530.

Realizou-se a seguinte simulação: o serviço **Apache** foi propositalmente parado, para simular uma queda de serviço no servidor primário **kanon**. Apresentamos o *log* do sistema dos dois servidores. O *log* do servidor **deathmask**, é apresentado na Figura 4.1, e nota-se que o mesmo recebe um sinal do servidor **kanon**, informando que este último entrará em modo *standby*, ou seja, irá desativar todos os seus serviços. Já no log do servidor **kanon**, apresentado através da Figura 4.2, pode ser visualizado que o mesmo consegue desativar seus serviços com sucesso.

```
16:47:14 deathmask heartbeat: [13428]: info: kanon wants to go standby [all]
16:47:15 deathmask kernel: [529886.751352] block drbd0: peer( Primary -> Secondary )
16:47:15 deathmask kernel: [529886.753088] block drbd1: peer( Primary -> Secondary )
16:47:16 deathmask heartbeat: [13428]: info: standby: acquire [all] resources from kanon
16:47:16 deathmask heartbeat: [13494]: info: acquire all HA resources (standby).
```

Figura 4.1: Log do servidor Deathmask.

```
16:47:04 kanon heartbeat: [1247]: info: standby: deathmask can take our all resources
16:47:04 kanon heartbeat: [2134]: info: give up all HA resources (standby).
16:47:04 kanon ResourceManager[2148]: info: Releasing resource group: deathmask 10.57.2.9 drbddisk::r0 Filesystem::/de
nfs-kernel-server slapd drbddisk
16:47:04 kanon ResourceManager[2148]: info: Running /etc/ha.d/resource.d/drbddisk  stop
16:47:04 kanon kernel: [366051.268162] block drbd0: role( Primary -> Secondary )
16:47:04 kanon kernel: [366051.270045] block drbd1: role( Primary -> Secondary )
16:47:04 kanon ResourceManager[2148]: info: Running /etc/ha.d/resource.d/slapd  stop
16:47:04 kanon ResourceManager[2148]: info: Running /etc/init.d/nfs-kernel-server  stop
16:47:04 kanon kernel: [366051.322111] nfsd: last server has exited, flushing export cache
16:47:04 kanon ResourceManager[2148]: info: Running /etc/init.d/mysql  stop
16:47:04 kanon ResourceManager[2148]: info: Running /etc/ha.d/resource.d/apache2  stop
16:47:04 kanon ResourceManager[2148]: info: Running /etc/ha.d/resource.d/Filesystem /dev/data/RECURSOS /srv/recursos/
16:47:04 kanon Filesystem[2305]: INFO: Running stop for /dev/data/RECURSOS on /srv/recursos
16:47:04 kanon Filesystem[2305]: INFO: Trying to unmount /srv/recursos
16:47:05 kanon Filesystem[2305]: INFO: unmounted /srv/recursos successfully
16:47:05 kanon Filesystem[2299]: INFO:  Success
16:47:05 kanon ResourceManager[2148]: info: Running /etc/ha.d/resource.d/drbddisk r0 stop
16:47:05 kanon ResourceManager[2148]: info: Running /etc/ha.d/resource.d/IPaddr 10.57.2.9 stop
16:47:05 kanon IPaddr[2437]: INFO: ifconfig eth0:0 down
16:47:05 kanon IPaddr[2425]: INFO:  Success
```

Figura 4.2: Log do servidor Kanon.

4.1.1 Relatórios

Para gerar os relatórios que serão apresentados nesta seção do trabalho, foi utilizado o software Zabbix [43].

Estado dos serviços antes da simulação iniciar.

Antes de chegarmos ao passo em que apresentam-se os gráficos da troca de serviços entre os servidores, mostraremos primeiramente o estado inicial dos serviços em cada servidor. Deve-se assumir que em todos os gráficos que serão apresentados a seguir o eixo Y representa o número de instâncias que o serviço possui no sistema, e o eixo X representa o tempo, que é amostrado de uma em uma hora.

A Figura 4.3 indica o estado (disponibilidade) do serviço **Apache** no servidor **kanon** ao longo do tempo. Nota-se observando a parte final do gráfico, que este serviço está executando de forma correta.

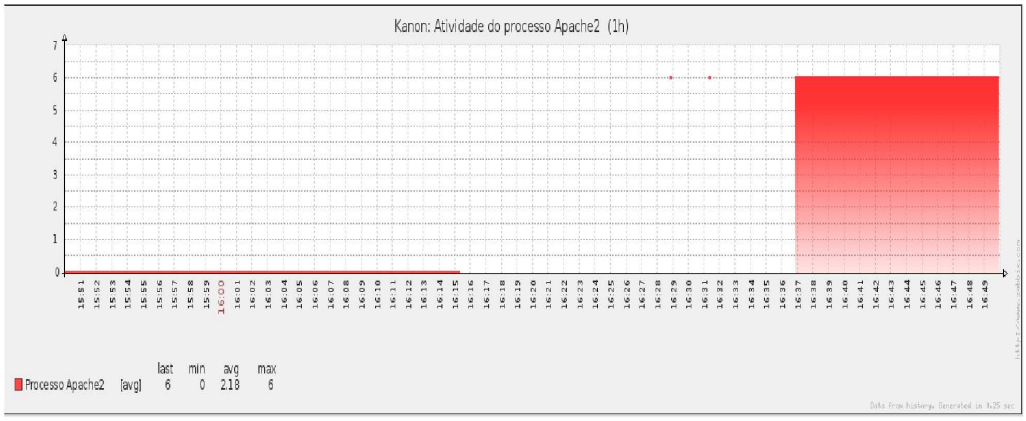

Figura 4.3: Gráfico de disponibilidade do Apache no servidor kanon.

No caso do servidor **Deathmask**, os serviços não estão executando, já que o mesmo está, neste momento, definido como servidor secundário. E para mostrar ao leitor que os serviços realmente estão parados, apresenta-se a Figura 4.4, com o estado do serviço **Apache** neste servidor.

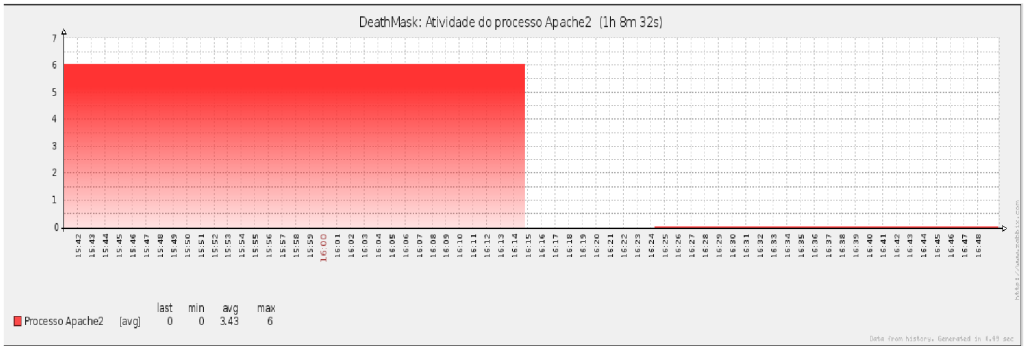

Figura 4.4: Gráfico de disponibilidade do Apache no servidor Deathmask.

Monitorar os mesmos serviços utilizando o endereço IP 10.57.2.9 (endereço IP do cluster), pode parecer redundante, já que os serviços no mesmo terão o mesmos estados do servidor **kanon**. Porém, a ideia é que o leitor perceba que quando este servidor entrar no modo *standby*, o IP passará a apontar para o servidor **Deathmask**. E o que espera-se desta solução é que o **zabbix** não perceba instabilidade quando estiver monitorando este IP. Assim pode-se constatar que os clientes também não irão notar tal acontecido. Desta forma, as Figuras 4.5 e apresentam gráficos de disponibilidade dos serviços monitorando o IP do cluster antes do servidor **kanon** entrar em modo *standby*.

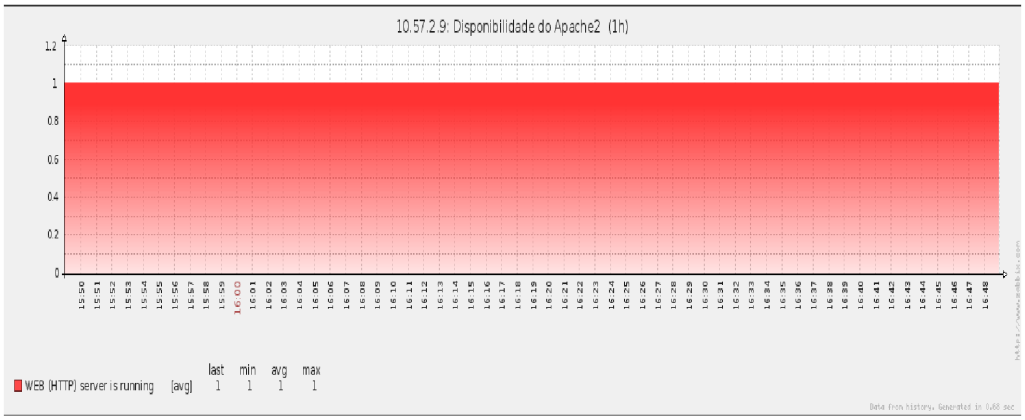

Figura 4.5: Gráfico de disponibilidade do Apache2 monitorando o IP do cluster.

4.1.2 Estado dos serviços após o início da simulação.

Após ter sido apresentado o estado dos serviços escolhidos antes do início do processo de troca de responsabilidade entre os servidores, será mostrado‘ os estados dos serviços no momento em que o servidor **kanon** entra em modo *standby*. Então pode-se observar na Figura 4.6, que neste servidor o serviço **Apache2** com 6 instâncias, passa a não‘ter instâncias executando, ou seja, ficam “fora do ar”.

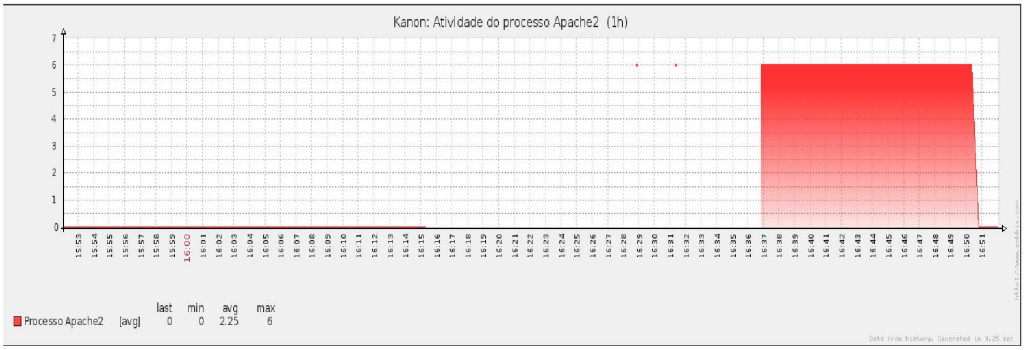

Figura 4.6: Gráfico de disponibilidade do Apache2 no servidor kanon.

Agora, o servidor **Deathmask**, se encontra como nó primário do cluster, e assume também o endereço IP 10.57.2.9, ou seja, este servidor tem a‘ responsabilidade de responder as requisições dos clientes. Assim sendo, nota-se na Figura 4.7 que o serviço **Apache2** mudou seu estado‘ de desligado para ligado, e assim passou a funcionar corretamente.

Após a simulação descrita acima, chega o momento em que será apresentado o resultado que prova a solução descrita no decorrer deste trabalho.

A Figuras 4.8, mostra ao leitor que um dos serviços (**Apache2**) não sofreu indisponibilidade, o que‘ comprova que o cliente não perceba a mudança dos servidores, e que o segundo serviço monitorado (**nfsd**), sofreu uma indisponibilidade em um tempo‘ mínimo, mas logo em seguida volta a funcionar. O que no pior caso pode gerar uma queda momentânea de conexão, mas em um segundo a conexão se restabelece.

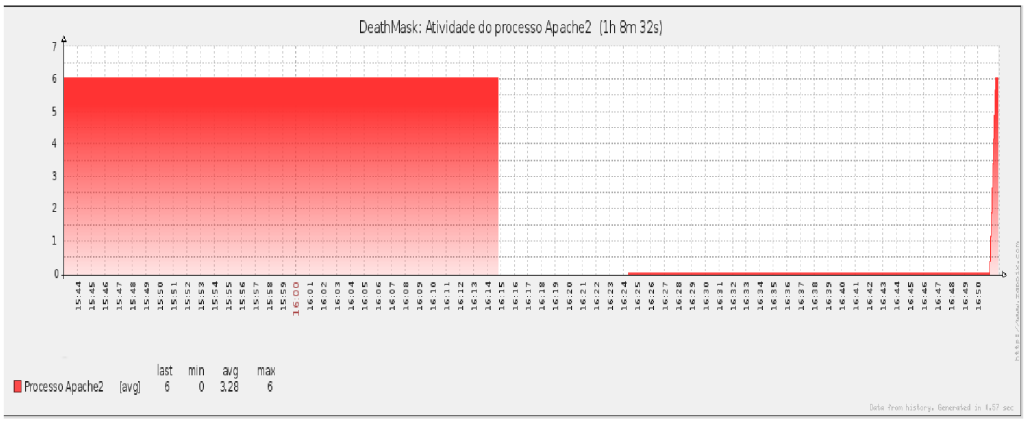

Figura 4.7: Gráfico de disponibilidade do Apache2 no servidor Deathmask.

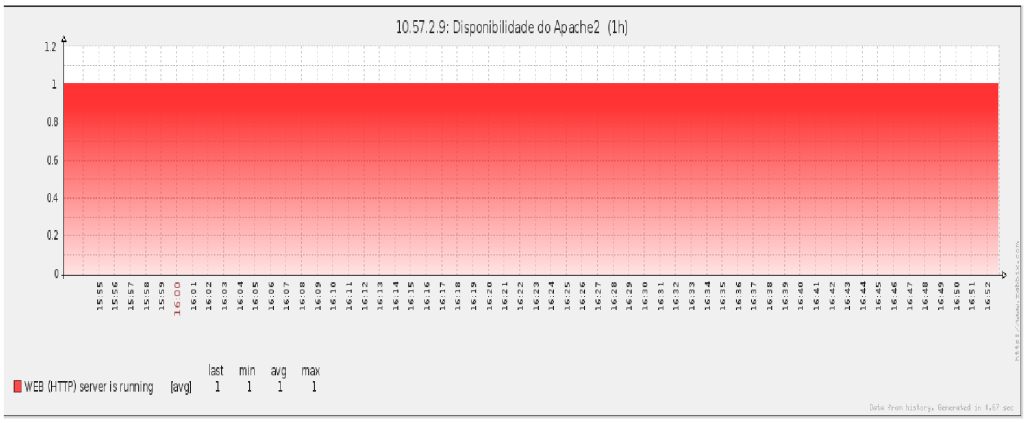

Figura 4.8: Gráfico de disponibilidade do Apache2 monitorando o IP do cluster.

4.2 Reconhecimento de Fala

Esta seção do trabalho apresenta os resultados obtidos a partir da utilização de um software desenvolvido sobre a plataforma *Android 2.2* com suporte a reconhecimento de voz em Português Brasileiro através do decodificador Julius [37], que opera em modo distribuído; e da API *SpeechRecognizer* do Google [44] que, também operando em modo distribuído, foi utilizada como base para os testes comparativos.

Nesta parte do trabalho será apresentada uma comparação entre as ferramentas de reconhecimento de fala e desenvolvimentos futuros.

4.2.1 Aplicativo e Plataforma de testes

Para a realização de testes que possam demonstrar o desempenho do serviço de reconhecimento de voz proposto neste trabalho, foi criado um software simples que pode ser usado em qualquer *smartphone* com suporte ao sistema *Android*.

Quando o usuário pressiona o botão *start* e fala o local da cidade, o aplicativo envia a voz via *stream* de áudio ao servidor (nuvem) de reconhecimento de voz (Julius), e logo após o aplicativo realizar detecção de silêncio, o envio de dados ao servidor é interrompido. Este, então, realiza reconhecimento de voz através do Julius e em seguida devolve, através da mesma conexão, uma string contendo o resultado mais aproximado que encontrou.

Para melhor entendimento, pode-se observar a Figura 4.9.

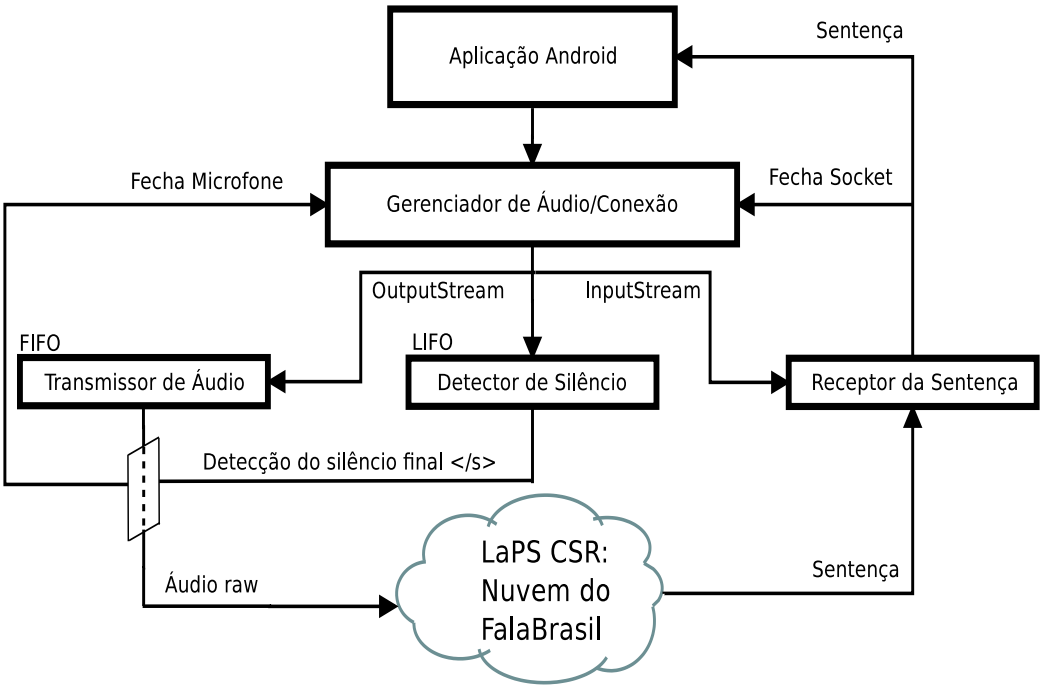

Figura 4.9: Funcionamento do sistema de Reconhecimento de Fala.

O dispositivo móvel utilizado para os testes foi o Samsung Galaxy Y GT-S6102B, com processador ARMv6 BCM21553 e 290 MB de RAM.

Para verificar a viabilidade (taxa de acerto e tempo) das ferramentas consideradas, 50 sentenças (wav) pré-gravadas com uma frequência de amostragem de 8 kHz foram reproduzidas para cada reconhecedor utilizando a mesma caixa de som e o mesmo ambiente acústico. Para o Julius, a transcrição de cada arquivo se configura como uma possível sentença na gramática livre de contexto, enquanto que a ferramenta do Google não nos permite utilizar uma gramática, já que então utiliza um modelo de linguagem próprio.

Neste trabalho, o *average real time factor* ($\overline{xRT}$) foi definido como a média entre o tempo total do processo T_{ret} (desde a detecção do silêncio até o retorno da sentença) dividido pela duração do áudio T_{aud} de cada arquivo:

$$\overline{xRT} = \frac{1}{N_s} \times \sum_{i=1}^{N_s} \frac{T_{ret_i}}{T_{aud_i}}, \tag{4.1}$$

onde o número de sentenças é $N_s = 50$. Essa métrica foi adotada por conta da impossibilidade de se obter alguns dados, como o tempo de decodificação do Google, tempo de envio do áudio remanescente após a detecção do silêncio, etc.

Para melhor entendimento de como o $\overline{xRT}$ apresenta-se neste sistema, pode-se observar a Figura 4.10

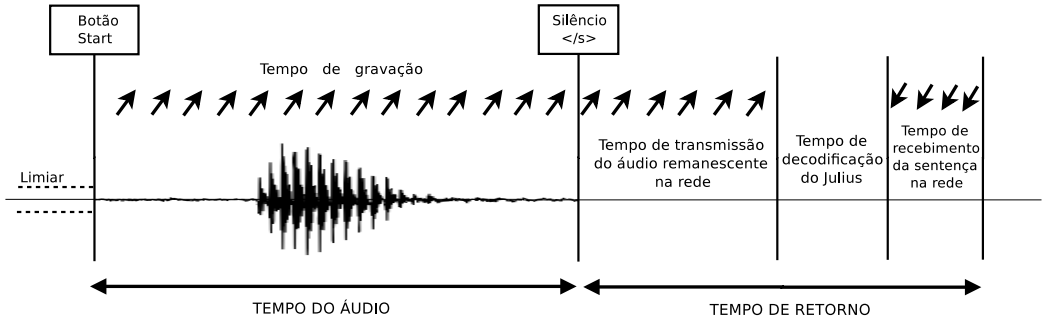

Figura 4.10: $\overline{XRT}$.

A taxa de acerto (acurácia) foi calculada a partir da taxa de erro por sentença reconhecida (SER) e taxa de erro por palavra (WER) dada por:

$$SER = \frac{\text{número de sentenças incorretas}}{N_s} \times 100\% \tag{4.2}$$

$$WER = \frac{D + S + I}{N_p}, \tag{4.3}$$

onde N_p é o número de palavras na sentença de entrada, D, S e I são, respectivamente, os números de erros de deleção, substituição e inserção na sentença reconhecida.

4.2.2 Resultados

A comparação entre os reconhecedores prova que o sistema de reconhecimento distribuído apresentado se mostrou bastante satisfatório da maneira que foi implementado.

A Tabela 4.1 mostra uma comparação entre os valores de $\overline{xRT}$ para as ferramentas. Os dois sistemas distribuídos foram testados com acesso via rede celular 3G e Wifi.

Tabela 4.1: Comparação do tempo de retorno entre as ferramentas.

Ferramenta	$\mu_{T_{ret}}$ (s)		$\overline{xRT}$	
	3G	Wifi	3G	Wifi
Julius	10,5	0,95	2,7231	0,2279
Google	7,5	0,70	2,4086	0,2063

Em termos de taxa de acerto, o Julius, utilizando uma gramática com 482 palavras e 235 sentenças possíveis, obteve uma SER de 16% contra 24% do Google.

Já com relação à *WER*, Julius novamente obteve uma taxa melhor do que a do Google: 10,16% contra 13,33%.

Alguns erros gramaticais (não fonéticos) foram observados nos resultados do Google e incluídos nos cálculos da acurácia. Por exemplo, a frase “bernardo sayão” foi escrita incorretamente como “bernardo saião”. Já o Julius

não apresentou esse tipo de erro por conta do uso de gramáticas personalizadas.

A Tabela 4.2 mostra uma comparação entre a taxa de erro por sentença e por palavra para os dois reconhecedores.

Tabela 4.2: Comparação de acurácia entre os sistemas.

Ferramenta	SER (%)	WER (%)
Google	24	13,33
Julius	16	10,16

As Figuras 4.11, 4.13, 4.12 e 4.14 apresentam um histograma dos tempos de retorno de sentença, para cada servidor (Julius e Google), obtidos durante os testes realizados, utilizando redes wireless e 3G. Deve-se Observar que, para os gráficos apresentados a seguir, tem-se no eixo Y o número de áudios e no eixo X, o tempo. Para os testes foram variados os servidores (Julius e Google), assim como a tecnologia de acesso a internet, entre Wireless e 3G.

A Figura 4.11, apresenta um histograma do tempo de retorno do áudio, utilizando rede *Wireless*, e utilizando o servidor do Google. Para este cenário, pode-se observar que no pior caso, ou seja, o pior atraso, ocorreu com apenas um áudio, onde o tempo de retorno chegou a 1,6 segundos, e que há uma maior concentração com tempo menor que 1 segundo.

A Figura 4.12, apresenta um histograma do tempo de retorno do áudio, utilizando rede *Wireless*, e utilizando o servidor Julius (LaPS Cloud). Para este cenário, pode-se observar que no pior caso, ou seja, o pior atraso, ocorreu com apenas um áudio, onde o tempo de retorno chegou a 1,4 segundos, e que há uma maior concentração com tempo menor que 1 segundo.

A Figura 4.13, apresenta um histograma do tempo de retorno do áudio, utilizando rede *3G*, e utilizando o servidor do Google. Para este cenário, pode-se observar que no pior caso, ou seja, o pior atraso, ocorreu com apenas um áudio, onde o tempo de retorno chegou a 18 segundos, e que há uma maior concentração com tempo entre 6 e 10 segundos. O caso em que mais áudios repetiram o mesmo tempo, foram 8 áudios com aproximadamente 8

segundos.

A Figura 4.14, apresenta um histograma do tempo de retorno do áudio, utilizando rede *3G*, e utilizando o servidor Julius (LaPS Cloud). Para este cenário, pode-se observar que no pior caso, ou seja, o pior atraso, ocorreu com apenas um áudio, onde o tempo de retorno chegou a 12 segundos, e que há uma maior concentração com tempo entre 10 e 11 segundos. O caso em que mais áudios repetiram o mesmo tempo, foram 8 áudios com aproximadamente 11 segundos.

Ao se utilizar a rede 3G para realizar os testes necessários, foi observado que em alguns momentos a lentidão para resposta em ambos os servidores se tornava expressiva. Alguns áudios tiveram um tempo de resposta acima de 10 segundos. Já para os resultados com utilização da rede wireless não foi observado intermitência no serviço. Sendo assim os resultados mostram que o Julius obteve um resultado próximo ao do Google no quesito tempo.

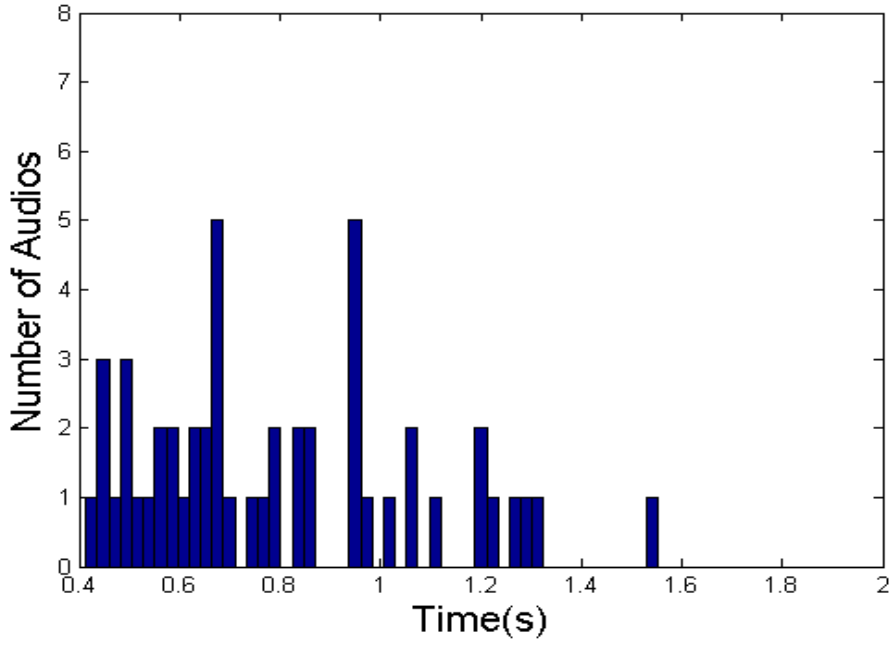

Figura 4.11: Histograma do tempo de retorno para o Google em rede Wireless.

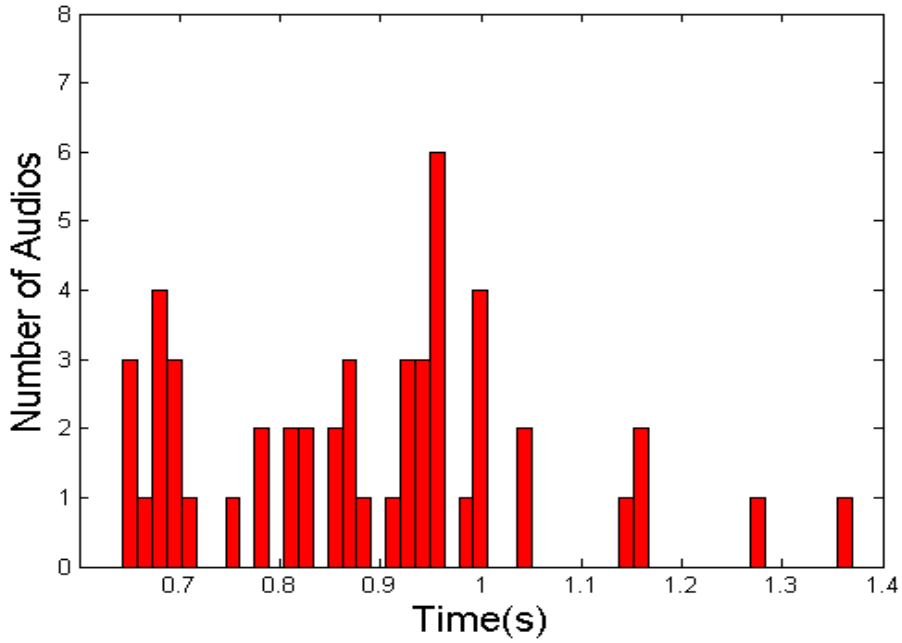

Figura 4.12: Histograma do tempo de retorno para o Julius em rede Wireless.

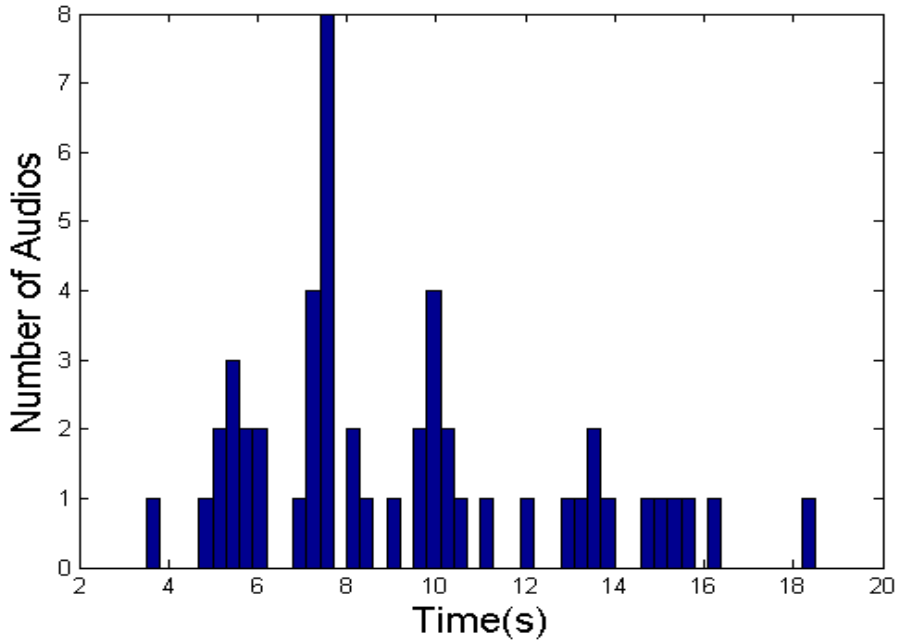

Figura 4.13: Histograma do tempo de retorno para o Google em rede 3G.

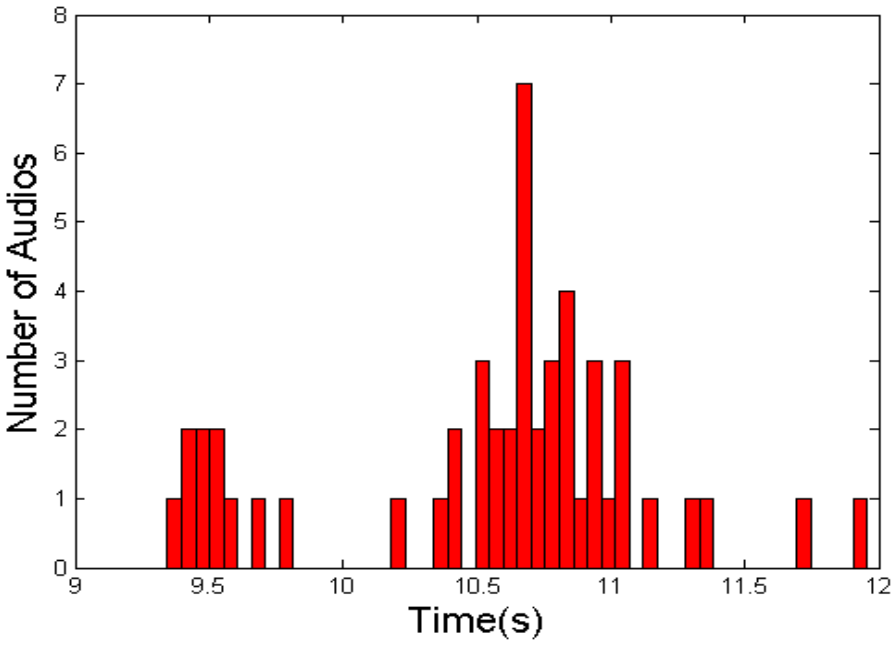

Figura 4.14: Histograma do tempo de retorno para o Julius em rede 3G.

Capítulo 5

Considerações Finais

Em tempos onde a tecnologia está cada dia mais fazendo parte do cotidiano do ser humano, onde é notável o crescente número de empresas investindo em soluções inovadoras, os sistemas ASR, ganham grande força, sem esquecer que não se pode pensar ou planejar grandes soluções, com milhares ou milhões de usuários, acessos, requisições, sem prover estes com alto grau de disponibilidade. Empresas como Google, Microsoft, dentre outras, estão entre as maiores empresas do mundo, e entre as que mais investem nestes recursos tecnológicos, de forma que não se torna difícil visualizar que estes sistemas se tornam cada vez mais fortes, e cada vez mais, estas soluções são aceitas por parte dos usuários no mundo.

É importante ressaltar que a solução apresentada neste trabalho, é composta por recursos livres, sendo uma solução de baixo custo, e com resultados expressivos.

O sistema ASR, se equipara ao sistema de uma gigante empresa do ramo tecnológico no mundo, acompanhando o avanço tecnológico, nota-se que os *smartphones* são extensamente utilizados pela população mundial, e a solução proposta aqui, está disponível para para estes dispositivos assim como para Desktops, tablets, dentre outros.

5.1 Trabalhos Futuros

A nuvem de reconhecimento ainda é composta por um único computador respondendo a requisições dos clientes, desta forma pode-se então aumentar o número de computadores, ou seja, a configuração desta solução em ambiente com utilização de um *cluster* para reconhecimento de Fala, ou seja, vários computadores trabalhando paralelamente com intuito de aumentar a velocidade de resposta ao cliente, assim como contribui para o aumento da disponibilidade do serviço.

Outra opção de trabalhos futuros é realizar comparação entre sistemas de reconhecimento locais, ou seja sem a utilização de um servidor, por exemplo utilizar o Sphinx, ou mesmo uma versão para *smartphones*, tal como o *pocketsphinx*.

Referências Bibliográficas

[1] "Documentação do projeto Heartbeat," 2011, Acesso em 28/11/2013.

[2] "Documentação do Mon," 2011, Acesso em 30/11/2013.

[3] Lars Ellenberg, "Drbd 9 & device-mapper," *Linux-Kongress*, 2008.

[4] Bruno Gomes Haick, *Soluções livres para computação de Alta Disponibilidade: Estudo de caso usando DRBD, HEARTBEAT E MON*, 2011.

[5] "Artica," 2011, Acesso em 02/12/2013.

[6] "GalaxyVisions," 2011, Acesso em 02/12/2013.

[7] "LifeKeeper," 2011, Acesso em 02/12/2013.

[8] "Projeto Lemuria," 2011, Acesso em 02/12/2013.

[9] "Documentação do projeto Linux Virtual Server," 2011, Acesso em 01/12/2013.

[10] "Linux LVM Guide," 2011, Acesso em 26/11/2013.

[11] Derek Vadala, *Managing RAID on Linux*, O'Reilly, 2003.

[12] "Site oficial Drbd," 2011, Acesso em 22/11/2013.

[13] Daniel P. Bovet and Marco Cesati, *Understanding the Linux Kernel*, O'Reilly, 2005.

[14] James F. Kurose and Keith W. Ross, *Rede de computadores e a Internet, Uma abordagem top-down*, chapter 3, Pearson Addison Wesley, 2006.

[15] "GNU General Public License," 2010, Acesso em 29/11/2013.

[16] "Iniciativa Open Source," 2011, Acesso em 28/11/2013.

[17] X. Huang, A. Acero, and H. Hon, *Spoken Language Processing*, Prentice-Hall, 2001.

[18] A. M. da Cunha and L. Velho, "Métodos probabilísticos para reconhecimento de voz," Tech. Rep., Laboratório VISGRAF - Instituto de Matemática Pura e Aplicada, 2003.

[19] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*, PTR Prentice Hall, Englewood Cliffs, N.J., 1993.

[20] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. on ASSP*, vol. 26, no. 1, pp. 43–49, 1978.

[21] Jelinke and Frederick, "Métodos estatísticos para reconhecimento de voz," *The MIT Press*, 1998.

[22] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–86, Feb. 1989.

[23] H. Juang and R. Rabiner, "Hidden Markov models for speech recognition," *Technometrics*, vol. 33, no. 3, pp. 251–272, 1991.

[24] P. Woodland and D. Povey, "Large scale discriminative training of hidden Markov models for speech recognition," *Computer Speech and Language*, vol. 16, pp. 25–47, 2002.

[25] Akinobu Lee, Tatsuya Kawahara, and Kiyoshiro Shikano, "Gaussian mixture selection using context-independent HMM," *In Proceedings IEEE-ICASSP*, 2001.

[26] M. Cohen, H. Franco, N. Morgan, D. Rumelhart, and V. Abrash, "Hybrid neural network/hidden markov model continuous speech recognition," *Proceedings of the International Conference on Spoken Language Processing*, 1992.

[27] H Schwenk, "Using boosting to improve a hybrid HMM/neural network speech recognizer," in *ICASSP*, 1999, pp. 1009–12.

[28] Andrew Senior, "An empirical study of learning rates in deep neural networks for speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[29] Geoffrey Hinton, "Deep neural networks for acoustic modeling in speech recognition," in *IEEE Signal Processing Magazine*, 2012.

[30] J. Picone, "Signal modeling techniques in speech recognition," *Proceedings of the IEEE*, vol. 81, no. 9, pp. 1215–47, Sep. 1993.

[31] J.-C. Junqua and J.-P. Haton, *Robustness in Automatic Speech Recognition*, Kluwer, 1996.

[32] S. Davis and P. Merlmestein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Trans. on ASSP*, vol. 28, pp. 357–366, Aug. 1980.

[33] P. Ladefoged, *A Course in Phonetics*, Harcourt Brace, 4 edition, 2001.

[34] L.R. Bahl, P.V. Souza, P.S. Gopalakrishnan, D. Nahamoo, and M.A. Picheny, "Context dependent modeling of phones in continuous speech using decision trees," in *DARPA Speech and Natural Language Processing Workshop*, 1994, pp. 264–270.

[35] R.P. Lippmann, "Speech recognition by machines and humans," *Speech Communication*, vol. 22, pp. 1–15, 1997.

[36] Richard Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[37] Akinobu Lee, Tatsuya Kawahara, and Kiyoshiro Shikano, "Julius - an open source real-time large vocabulary recognition engine," *Proc. European Conference on Speech Communication and Technology*, pp. 1691–1694, 2001.

[38] Nelson Neto, Carlos Patrick, Aldebaro Klautau, and Isabel Trancoso, "Free tools and resources for Brazilian Portuguese speech recognition," *Journal of the Brazilian Computer Society*, vol. 17, pp. 53–68, 2011.

[39] Ana Siravenha, Nelson Neto, Valquíria Macedo, and Aldebaro Klautau, "Uso de regras fonológicas com determinação de vogal tônica para conversão grafema-fone em Português Brasileiro," *7th International Information and Telecommunication Technologies Symposium*, 2008.

[40] "http://htk.eng.ac.uk," Visitado em dezembro de 2013.

[41] "Centro de Estudos em Telecomunicações (CETUC)," Visited in August, 2012.

[42] Maicon Alves Alves, *Sockets Linux*, BRASPORT, 2008.

[43] Rihards Olups, *Zabbix 1.8 Network Monitoring*, Packt Publishing, 2010.

[44] "Android Developers," Acessado em dezembro de 2013, `http://developer.android.com/`.