Deaf Accessibility as a Service: uma Arquitetura Elástica e Tolerante a Falhas para o Sistema de Tradução VLIBRAS

Eduardo Falcão, Tiago Maritan e Alexandre Duarte

[1]Digital Video Applications Lab - LAVID
Federal University of Paraiba - UFPB
João Pessoa, Paraíba, Brazil
{eduardolf, tiagomaritan, alexandre}@lavid.ufpb.br

Abstract. *When designed, Information and Communication Technologies rarely take into account the barriers that deaf people face. Currently, there are tools for automatic translation from spoken languages to sign languages, but, unfortunately, they are not available to third parties. To reduce these problems, it would be interesting if any automatic translation service could be publicly available. This is the general goal of this work: use a preconceived machine translation from portuguese language to Brazilian Sign Language (LIBRAS), named VLIBRAS, and provide Deaf Accessibility as a Service publicly. The idea is to abstract inherent problems in the translation process between the portuguese language and LIBRAS by providing a service that performs the automatic translation of multimedia content to LIBRAS. VLIBRAS was primarily deployed as a centralized system, and this conventional architecture has some disadvantages when compared to distributed architectures. In this paper we propose a distributed architecture in order to provide an elastic service and achieve fault tolerance.*

1. Introdução

A língua que um indivíduo usa para se comunicar depende de sua natureza além do grupo de indivíduos com o qual ele convive. Os ouvintes, por exemplo, comunicam-se por intermédio de línguas oralizadas. Já os surdos, por outro lado, encontram na linguagem gestual-corporal um meio eficaz de comunicação como alternativa à falta de capacidade auditiva. Portanto, a língua na qual o surdo consegue perceber e produzir de maneira natural é a língua de sinais (LS), ao passo que as línguas orais, utilizadas cotidianamente pela maioria das pessoas, representam apenas "uma segunda língua" [de Góes 1996].

Segundo o censo demográfico realizado em 2010 pelo Instituto Brasileiro de Geografia e Estatística (IBGE), 5,1% da população brasileira possui deficiência auditiva. Deste total, 1,7 milhões têm grande dificuldade para ouvir e 344,2 mil são surdos [Instituto Brasileiro de Geografia e Estatística 2010]. Normalmente, nos diferentes contextos da sociedade atual brasileira, a informação é transmitida através da língua portuguesa. Mas o nível de proficiência dos surdos na língua portuguesa pode tornar a leitura uma tarefa árdua e limitada [Gomes and Góes 2011], fazendo deste fator uma barreira a mais na inclusão digital.

A LIBRAS é dotada de uma gramática própria, diferente da gramática da língua portuguesa. Com relação à ordem das palavras, por exemplo, existem diferenças entre as duas línguas [de Araújo 2012]. Na maioria dos casos, a língua portuguesa utiliza

sentenças no formato sujeito-verbo-objeto (SVO), enquanto que a LIBRAS geralmente utiliza sentenças no formato tópico-comentário [Brito 1995]. Araújo (2012) utilizou os seguintes exemplos para explicar esta diferença:

- O urso (S) matou (V) o leão (O).
- Eu (S) não vi (V) o acidente na rua (O).

As sentenças supracitadas seriam representadas em LIBRAS da seguinte forma:

- URSO (Tópico), LEÃO MATAR (Comentário).
- RUA ACIDENTE (Tópico) NÃO ENXERGAR (Comentário).

É de extrema importância a existência da tradução de LIBRAS para a língua portuguesa, com objetivo de "reorganizar" as frases alterando a ordem das palavras, as palavras em si (e.g., verbos em LIBRAS são sempre no infinitivo) e eliminando partes desnecessárias (e.g., artigos). Caso contrário, a sinalização realizada resultaria apenas em "português sinalizado", de dificuldade ainda elevada por não constituir a LIBRAS.

A acessibilidade para surdos requer primordialmente a tradução para LIBRAS. Esta tradução é de extrema importância mas traz consigo alguns problemas:

1. Alto custo do serviço: segundo SINTRA (2010) , a tradução de 60 minutos de um áudio em língua portuguesa para a LIBRAS custa 585 reais;
2. Grande dinamismo de conteúdos nas tecnologias de informação e comunicação (TICs).

Para sistemas de *Video on Demand* - VoD, prover acessibilidade para surdos também é uma tarefa complicada. Ao analisar o *Youtube*, por exemplo, é possível notar a inviabilidade da tradução de seus vídeos por intérpretes, devido aos milhões de usuários do serviço, e à grande quantidade de vídeos que são enviadas ao sistema.

Uma das soluções para sistemas de VoD seria a utilização de um serviço de tradução automática da língua portuguesa para a LIBRAS. Para tanto, este trabalho utiliza o sistema VLIBRAS [de Araújo 2012] para oferecer este serviço. Atualmente, o VLIBRAS é implantado em uma arquitetura centralizada, e, portanto, pode ser aprimorado utilizando uma arquitetura distribuída. Para uma noção inicial da performance do VLIBRAS implantado em uma arquitetura centralizada[1], foi medido o tempo de processamento para 1, 10, 20, e 40 requisições simultâneas, apresentado na Tabela 1. Tais requisições foram compostas por textos com 324 palavras.

Tabela 1. Média de tempo de processamento para requisições concorrentes

Nº de requisições	Tempo médio	Pior tempo	Desvio padrão	Falhas
1	21.369s	21.369s	0	0
10	109.618s	118.023s	21.682	0
20	206.45106	223.023s	49.490	0
40	279.612s	281.016s	2.153	7

[1]Instância da Amazon c3.large, Ubuntu 12.04 - 64 bits, 2 vCPU (CPUs virtuais) e 7 ECUs (unidade de processamento elástico), 3.75GB RAM

Com esses testes, foi possível constatar que o sistema falha ao atender um número de requisições igual ou superior a 40. Deste modo, fica mais claro entender a necessidade de uma arquitetura distribuída e dinâmica. Como o serviço será disponibilizado de forma pública, em momentos de pico o sistema poderá receber cargas muito superiores a 40 requisições, porém, em momentos de baixa demanda, também poderá receber um número de requisições inferior a 40. Uma das principais características da arquitetura distribuída proposta é a capacidade de provisionamento dinâmico de recursos. Com ela é possível economizar financeiramente em momentos de baixa demanda, mas também prover um serviço de qualidade mesmo quando submetidos a grandes cargas de requisições.

Uma vez demonstrada a problemática que os surdos enfrentam nas TICs, este trabalho tem como objetivo geral a oferta de *Deaf Accessibility as a Service* (DAaaS) - um serviço de acessibilidade para surdos. Para tanto, será disponibilizado publicamente um serviço automatizado de tradução de conteúdos multimídia para a LIBRAS. Essa oferta pública demanda do VLIBRAS elasticidade para atender grandes cargas de requisições. Portanto, o principal objetivo deste trabalho é projetar um arquitetura distribuída, elástica e tolerante a falhas. A ideia é utilizar a nuvem para prover o DAaaS à terceiros de forma transparente, e utilizar os recursos de modo eficiente. Apesar deste trabalho utilizar um sistema de tradução previamente concebido, avaliar o nível de acerto na tradução realizada pelo sistema não entra no escopo dos objetivos.

2. Acessibilidade para Surdos Brasileiros nas TICs

Para conceber o DAaaS foi preciso analisar as ferramentas de tradução automática mais abrangente, considerando variedade de entradas e plataformas de disponibilização.

Diante dos sistemas de tradução português -> LIBRAS pesquisados, foram comparadas as principais características dos mesmos, com objetivo de justificar a escolha do VLIBRAS como núcleo da API DAaaS. As características avaliadas são:

1. formatos de entrada aceitos: texto, áudio, vídeo, legenda;
2. plataformas de execução: PC, *Web*, *mobile*, televisão digital;
3. forma de tradução: se a representação visual dos sinais é feita utilizando português sinalizado, ou utilizando a glosa (representação textual em LIBRAS);
4. Dicionário expansível: se o projeto disponibiliza alguma ferramenta que permita a expansão do dicionário de sinais por especialistas em LIBRAS.

A tabela 2 lista os trabalhos com suas respectivas características.

A característica mais importante a ser avaliada é a forma de tradução. Os trabalhos que convertem o texto em língua portuguesa para a glosa devem ser priorizados, uma vez que o português sinalizado é insuficiente para a compreensão completa por parte dos surdos. Normalmente, o dicionário de sinais é expandido por uma equipe de especialistas em LIBRAS e TI pertencentes ao projeto, porém, sabe-se que a quantidade de sinais a ser confeccionados é muito grande. Desse modo, pode-se perceber o quanto a funcionalidade de "Dicionário expansível" é importante, pois permite que pessoas externas ao projeto possam contribuir com a criação de novos sinais. Por fim, quanto mais opções de formatos de entrada e plataformas de execução o projeto oferecer, mais formatos de

[2]T = texto, A = áudio, V = vídeo, L = legenda

[3]W = *Web*, M = *mobile*

Tabela 2. Sistemas de Tradução Automática Português -> LIBRAS. Adaptado de: [Pivetta et al. 2011]

Tradutor	Entradas [2]				Plataformas [3]				Tradução	Código	Dic. expansível
	T	A	V	L	PC	W	M	TV			
F-LIBRAS	X				X				glosa	fechado	X
FALIBRAS	X	X			X	X	X		glosa	fechado	
POLI-LIBRAS	X				X	X			glosa	aberto	X
ProDeaf	X	X				X	X		glosa	fechado	
Rybená	X					X	X	X	port. sinalizado	fechado	
TLIBRAS	X	X			X			X	glosa	fechado	
VLIBRAS	X	X	X	X	X	X		X	glosa	fechado	X
VE-LIBRAS	X	X			X				port. sinalizado	fechado	

entrada e plataforma de execução o serviço DAaaS poderá oferecer. Portanto, a partir das informações colhidas e projetadas na tabela 2, foi concluído que para o presente trabalho a melhor opção é o sistema de tradução automática VLIBRAS. Nesse sentido, o código do VLIBRAS foi disponibilizado para a concepção do DAaaS.

3. *Deaf Accessibility as a Service*

Sistemas centralizados apresentam várias desvantagens quando comparados a sistemas distribuídos. A solução atual para o sistema de tradução automática VLIBRAS é baseada em um sistema centralizado, e, portanto, possui capacidade de tolerância a falhas limitada além de não ter capacidade de provisionamento dinâmico de recursos. Adicionalmente, a nuvem permite pagamento de recursos exclusivamente perante uso, não havendo necessidade de manter máquinas ociosas, e fácil implantação e configuração do serviço em diferentes lugares do mundo, não impondo a necessidade de espaço físico, energia, refrigeração, e manutenção da infraestrutura.

A capacidade de provisionamento dinâmico permitirá o sistema alocar recursos para atender grandes cargas de requisição e readaptar-se quando a demanda diminuir, economizando recursos [Radhakrishnan 2012]. Essa elasticidade automática é uma das características que atenua a probabilidade de ocorrência de falhas, através da alta disponibilidade dos recursos [Bala and Chana 2012]. Além disso, o trabalho utiliza uma técnica de tolerância a falhas reativa (resubmissão de tarefas) e uma proativa (*self-healing*).

3.1. Arquitetura Distribuída Proposta

A arquitetura proposta para o DAaaS foi projetada para ser implantada em uma infraestrutura de nuvem. O objetivo desta arquitetura é incorporar as seguintes funcionalidades:

> **Descentralização do fluxo de rede:** a existência de vários nós de processamento implica em vários fluxos de rede entre os clientes e as instâncias de processamento;

Provisionamento dinâmico de recursos: quando em momentos de sobrecarga, a escalabilidade sob demanda permite que o DAaaS tenha recursos disponíveis para processar as requisições através da inclusão de novas instâncias de processamento;
Tolerância a falhas:
Alta disponibilidade dos recursos: os recursos são sempre disponibilizados com redundância e em diferentes zonas de disponibilidade;
Em nível de *software*: tolerância a falhas reativa por resubmissão de tarefas, e proativa por meio de *self-healing*;

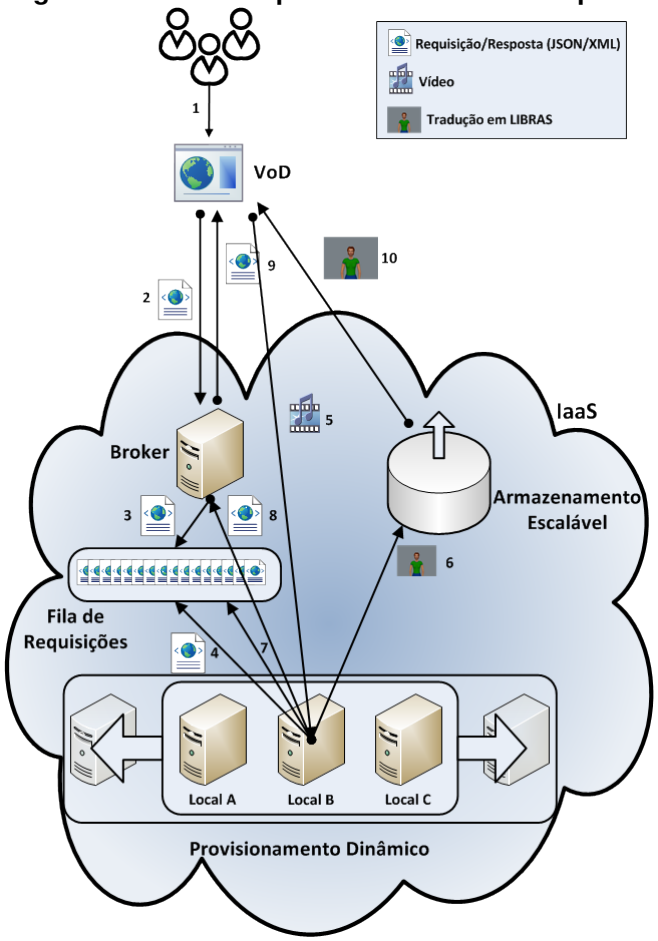
Para tal, pretende-se utilizar o *Broker* e a Fila de Requisições. O *Broker* funciona como um gerenciador de requisições, adicionando as novas requisições na Fila de Requisições, e notificando os nós de processamento. Como será explicado na seção 3.2, a Fila de Requisições tem como principal objetivo a recuperação de falhas utilizando um mecanismo de resubmissão.

Na arquitetura centralizada existe um único canal para fluxo de rede para que os conteúdos de entrada (vídeo, legenda, ou texto) e de saída (vídeo de tradução) trafeguem. Tal fato converge para elevar a probabilidade de haver um gargalo de rede, aumentando o tempo de resposta para as requisições. A arquitetura proposta (Figura 1) busca diminuir o gargalo de rede uma vez que existirá um canal de rede para cada instância de processamento, e, portanto, vários canais de tráfego de vídeos para o sistema como um todo. Ao invés do sistema de VoD, por exemplo, enviar primariamente o arquivo de vídeo para tradução (como acontece na arquitetura centralizada), envia apenas uma requisição (arquivo XML/JSON) que contém o endereço daquele vídeo, e então a instância de processamento estabelece uma conexão direta para realizar o *download* do vídeo. Esta característica do sistema evita a sobrecarga no canal de rede em que se insere o *Broker*, uma vez que os arquivos XML/JSON possuem carga extremamente inferior a arquivos de vídeo.

Na Figura 1, é ilustrado o funcionamento da arquitetura proposta. Para explicar o funcionamento desta arquitetura será utilizado o exemplo em que o sistema de VoD é o cliente do DAaaS. O processo completo para que um sistema de VoD requisite a tradução de algum conteúdo obedece a sequência de passos ordenada e identificada pelos números na Figura 1.

1. O usuário escolhe um vídeo do sistema de VoD para assistir e requisita a opção de acessibilidade;
2. O sistema de VoD envia uma requisição (arquivo JSON ou XML em conformidade com a API do DAaaS) de tradução ao *Broker* do sistema DAaaS;
3. O *Broker* adiciona essa requisição a uma fila de requisições, e notifica as instâncias de processamento para que elas saibam que existe uma nova requisição na fila;
4. As instâncias que possuírem recursos livres, irão competir para processar as requisições na fila. Cada requisição é alocada de forma atômica por uma única instância, nesse caso, a instância B;
5. A instância B realiza o *download* do vídeo e dá início à tradução automática;
6. Uma vez que a tradução para LIBRAS seja concluída, o vídeo de tradução é transferido para o serviço de armazenamento de arquivos escalável;
7. A instância envia um sinal para a Fila de Requisições remover aquela requisição específica, uma vez que tenha sido processada com sucesso;
8. Uma mensagem contendo o endereço do vídeo de tradução em LIBRAS é enviado para o *Broker*;

Figura 1. DAaaS - Arquitetura Distribuída Proposta



9. Essa mensagem é enviada para o sistema de VoD;
10. O sistema de VoD realiza o *download* do vídeo de tradução em LIBRAS e exibe para o usuário final de maneira sincronizada com o vídeo original.

A seguir, serão demonstradas as estratégias de tolerância a falhas (3.2) e provisionamento dinâmico de recursos (3.3), que visa prover resiliência e elasticidade ao DAaaS.

3.2. Tolerância a Falhas

3.2.1. Alta Disponibilidade dos Recursos

A existência de redundância de recursos em localizações geográficas diferentes provê tolerância a falhas por meio de alta disponibilidade. De acordo com a Figura 1, as instâncias de processamento estão situadas em diferentes locais físicos. É possível configurar o provisionamento dinâmico para criar novas instâncias em diferentes localizações geográficas, visando equilibrar a quantidade de instância nestes locais (e.g., 2 instâncias no "local A", 2 instâncias no "local B", e 2 instância no "local C"). Se porventura alguma unidade de processamento falhar, o DAaaS não ficará indisponível, uma vez que outros nós de processamento podem assumir a carga do nó defeituoso até que o problema seja resolvido. Deste modo, evita-se que problemas com rede ou energia em determinada localização, ou até mesmo falhas de *hardware*, interfiram na disponibilidade do DAaaS.

3.2.2. Em Nível de *Software*

Na antiga arquitetura centralizada, as falhas eram irreversíveis, ou seja, se ocorressem não eram passíveis de recuperação. A tolerância a falhas em nível de *software* permite a recuperação de uma falha ocorrida, além de sua prevenção por antecipação.

Os componentes responsáveis pela tolerância a falha em nível de *software* são o *Broker* e a Fila de Requisições. Com relação a tolerância a falha, as principais atividades do *Broker* são:

> **Gerenciamento de requisições:** toda requisição chega primeiro ao *Broker*, que é encarregado de enviá-la para a Fila de Requisições. Se não houvesse o gerenciamento as requisições seriam escalonadas automaticamente para um nó de processamento, tornando a recuperação de uma eventual requisição falha restrita àquela instância de processamento;
> **Notificação:** assim que o *Broker* adiciona uma nova requisição à Fila de Requisições, seu próximo passo é notificar todos os nós de processamento para que eles saibam o momento correto de consumir a Fila de Requisições;
> **Provisionamento dinâmico de recursos:** baseado na quantidade de requisições na Fila de Requisições, o *Broker* adiciona ou remove novos nós de processamento. O *Broker* também faz checagens periódicas para garantir que todos os nós de processamento estejam em pleno funcionamento, removendo os defeituosos e adicionando novos quando necessário.

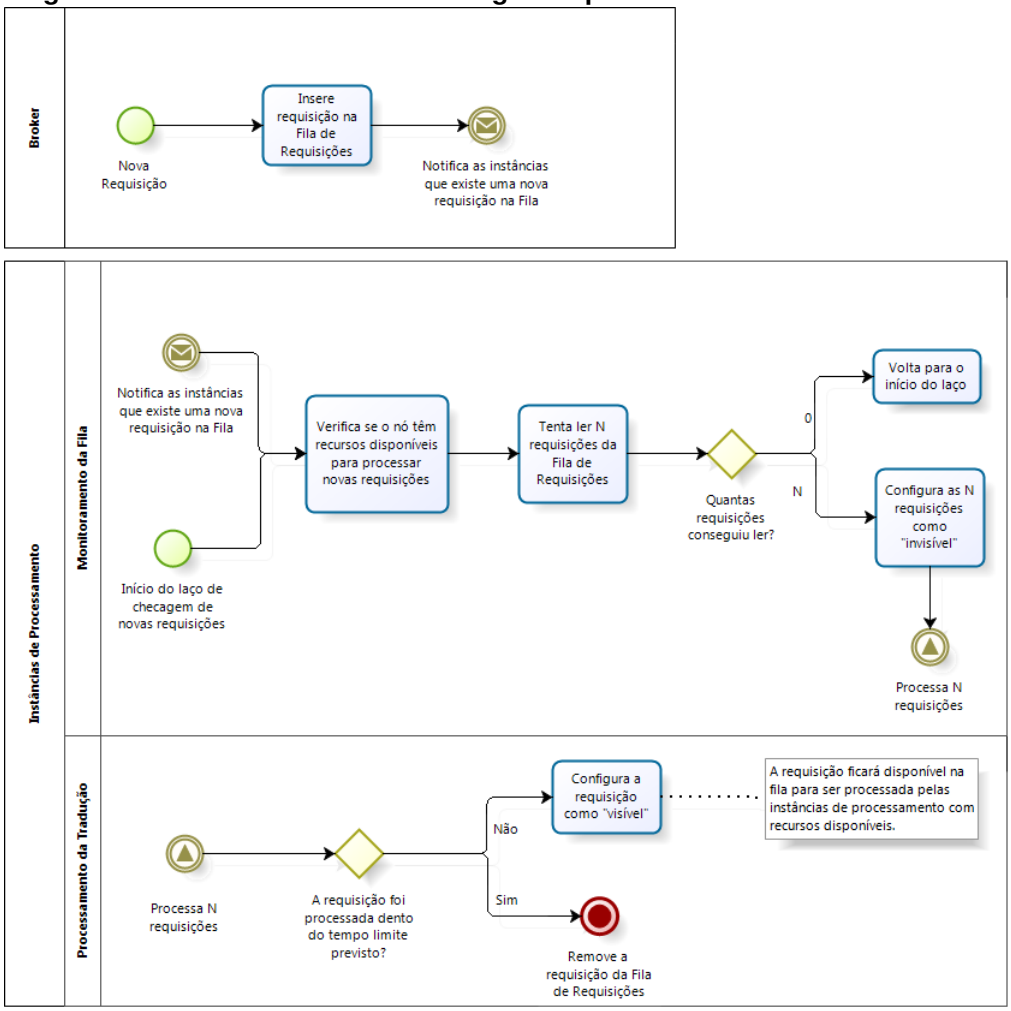
O serviço DAaaS usa uma técnica de tolerância a falha reativa por resubmissão de tarefas. A técnica reativa proposta consiste em resubmeter as requisições que falharem à Fila de Requisições. Para tal, todas as requisições desta Fila de Requisições são marcadas com uma *tag* "visível" ou "invisível". Todas as requisições inseridas nesta fila são marcadas por padrão como visível, para que todas as instâncias possam acessá-las. Quando o *Broker* adiciona novas requisições a essa Fila de Requisições, sua próxima tarefa é notificar as instâncias de processamento para avisar que a fila possui novos itens a serem consumidos. Assim que uma instância começa a processar uma requisição da fila, a mesma é marcada como "invisível" até que essa instância termine seu processamento e explicitamente a remova da fila.

Testes preliminares devem ser realizados para definir quantas requisições simultâneas cada instância consegue processar com sucesso, e qual o pior tempo para processamento em situações de . Consideramos como falha o caso em que uma instância não conseguir processar a requisição dentro desse pior tempo estipulado. Este pior tempo é utilizado como tempo de invisibilidade padrão, ou seja, se o processamento de alguma requisição for maior do que o pior tempo pré-calculado a requisição será automaticamente marcada como visível e será reprocessada por outra instância ou até pela mesma. Portanto, há a possibilidade de acontecer a "falha real", onde o processamento da requisição realmente falhou, mas também há a possibilidade da "falha falsa", quando uma instância demorar mais tempo para processar uma requisição do que o "pior tempo" estipulado. Quando ocorrer a "falha falsa" a requisição será processada duas ou mais vezes, onde o primeiro processamento pode até ser concluído com sucesso, mas por demorar muito é desconsiderado pelo DAaaS, e o último processamento terminará antes do pior tempo.

Quando uma requisição falhar e tornar-se visível novamente, não haverá mais nen-

huma notificação exclusiva do *Broker* às instâncias avisando que aquela requisição específica está mais uma vez visível. Contudo, as instâncias podem consumir a requisição que falhou através de outras notificações do *Broker* para outras requisições, ou através de um laço de checagem de requisições (15 segundos) que as instâncias de processamento implementam para consumir novas requisições independente da notificação do *Broker*. O modelo de processo de negócios para tolerância a falhas reativa por meio de resubmissão de tarefas é detalho na Figura 2, através de um diagrama BPMN (*Business Process Modeling Notation*).

Figura 2. Modelo de Processos de Negócios para Tolerância a Falhas Reativa



A técnica proativa também é implementada no *Broker*, através do provisionador dinâmico de recursos, que é encarregado de alocar ou desalocar recursos diante das métricas especificadas. Uma das funções deste provisionador é verificar se alguma das instâncias não está mais em funcionamento, através de requisições HTTP às máquinas supostamente ativas, e as repor caso elas aparentem estar defeituosas.

3.3. Provisionamento Dinâmico de Recursos

A estratégia utilizada para provisionamento dinâmico de recursos baseia-se na Fila de Requisições. Deste modo, testes preliminares devem ser realizados para definir quantas requisições simultâneas cada tipo de instância utilizada consegue processar, e qual

o pior tempo para processamento em situações de sobrecarga. Baseado na quantidade máxima de requisições simultâneas, no tamanho da Fila de Requisições, e na quantidade de instâncias de processamento ativas, é possível calcular se o DAaaS está sobrecarregado ou subutilizado, para criar novas instâncias ou remover algumas.

A tendência é que as instâncias de processamento estejam sempre sobrecarregadas (caso haja requisições na Fila), uma vez que o *Broker* sempre notifica todas as instâncias quando novas requisições são inseridas na fila, além da existência do laço de checagem de novas requisições nas instâncias de processamento, que independe do *Broker*. Portanto, as instâncias de processamento estarão sempre "absorvendo" a Fila de Requisições se tiverem recursos disponíveis, ou seja, sempre que não estiverem processando a quantidade máxima de requisições delimitadas.

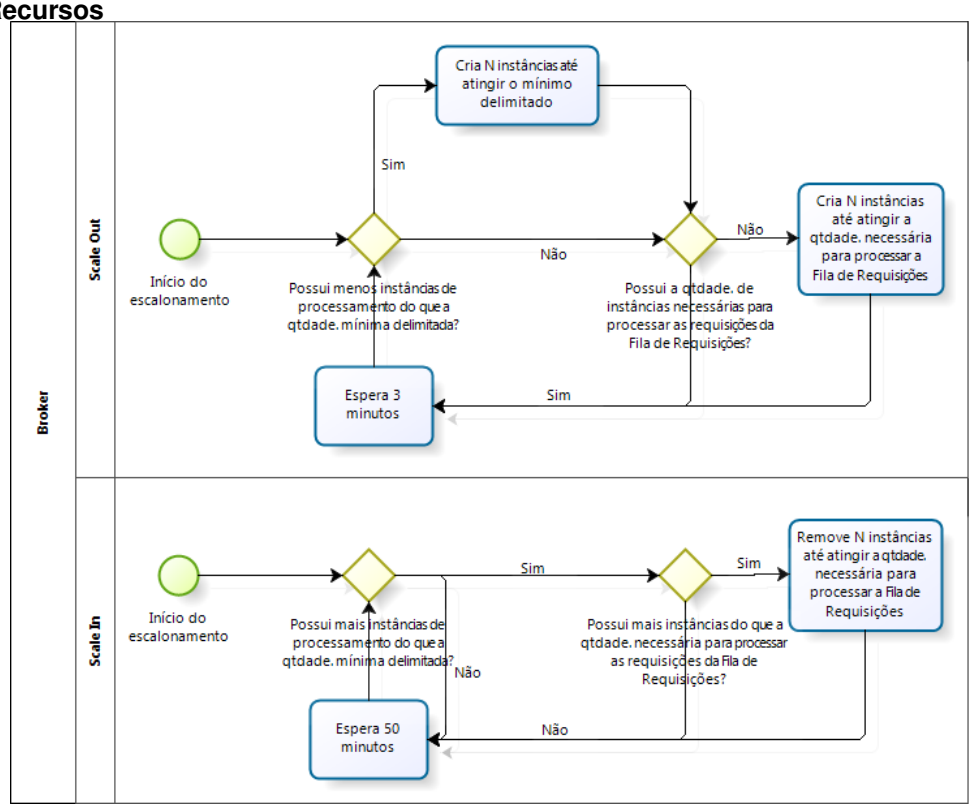
O diagrama ilustrado na Figura 3, detalha os passos do algoritmo de provisionamento dinâmico para criar ou terminar as instâncias de processamento VLIBRAS. Seguindo esse modelo de processo de negócios, é possível simular o que aconteceria se, por exemplo, a Fila de Requisições possuísse 1435 requisições, a instância pudesse processar no máximo 15 requisições, e o pior tempo para processamento em situações de sobrecarga fosse 30 minutos. Lembrando que o DAaaS inicia com 2 instâncias de processamento, que é a quantidade mínima, o algoritmo de escalonamento executaria os seguintes passos:

1. "O DAaaS possui menos instâncias de processamento do que a quantidade mínima delimitada?" **Não**, o DAaaS tem 2 instâncias disponíveis.
2. "O DAaaS possui a quantidade de instâncias necessárias para processar a Fila de Requisições?" **Não**, o DAaaS possui apenas 2 instâncias, que não têm capacidade para atender todas as requisições em tempo hábil.
3. "O DAaaS cria N instâncias até atingir a quantidade necessária para processar a Fila de Requisições." Abaixo vamos calcular N a partir do código do DAaaS:
   (a) int instancesRequired = (int) Math.ceil(requestsOnQueue/maxRequestsPerInstance);
   (b) int instancesRequired = (int) Math.ceil(1435/15);
   (c) int instancesRequired = (int) Math.ceil(95.66);
   (d) int instancesRequired = 96;
4. Sabendo que uma instância leva de 90 a 120 segundos para iniciar, 3 minutos é um tempo com folga para que a instância inicie e absorva a quantidade máxima de requisições, nesse caso, 15.
5. "O DAaaS possui menos instâncias de processamento do que a quantidade mínima delimitada?" **Não**, o DAaaS tem 96 instâncias disponíveis.
6. Quando o algoritmo de escalonamento chegar neste ponto, "O DAaaS possui a quantidade de instâncias necessárias para processar a Fila de Requisições?", a resposta será **SIM**, já que as 96 instâncias (94 recém criadas) absorveram toda a Fila de Requisições. A menos que nesse meio tempo novas requisições tenham chegado na fila, e que as instâncias ativas não as tenham processado.

Para redução da quantidade de instâncias, o chamado *scaling in*, o DAaaS esperaria o pior tempo para processamento em situações de sobrecarga. No caso do exemplo anterior, o DAaaS esperaria 50 minutos para calcular quantas instâncias deveriam ser eliminadas a partir do número de requisições na Fila, e da quantidade de instâncias ativas. Já que o pagamento na AWS é por hora, não faz sentido terminar instâncias já pagas por

um tempo distante de 60 minutos. Quando o *Broker* notifica uma instância para ser removida ela não é imediatamente excluída, porém, a partir desta notificação ela se prepara pra encerrar, ignorando as novas requisições na Fila de Requisições notificadas pelo *Broker* ou pelo laço de checagem. Uma *thread* é ativada para checar continuamente se as requisições em andamento já finalizaram, e em caso positivo, a *thread* encerra a instância atual. Um detalhe que pode ser melhorado nesse algoritmo é o monitoramento exato de quantas requisições cada instância ativa possui, para ao notificar a remoção de algumas instâncias, fazê-lo de forma crescente, notificando primeiramente as instâncias com menor número de requisições.

Figura 3. Modelo de Processos de Negócios para Provisionamento Dinâmico de Recursos



4. Cenários de Uso

A API está sendo disponibilizada na AWS, recebendo requisições na *url* http://184.169.148.166/Broker/requests. O DAaaS está implantado utilizando instâncias do tipo *m1.small* e é financiado por meio de créditos da AWS para pesquisas científicas. Atualmente, a API possui dois usuários: 1 - sítio do SENAPES; 2 - aplicativo VLIBRAS *mobile*.

O SENAPES é um evento que será aberto ao público e tem como objetivo principal a apresentação e discussão de experiências no desenvolvimento e implantação de tecnologias assistivas para promoção da acessibilidade para pessoas surdas nas TICs. Portanto, haverá grande acesso de pessoas surdas ao sítio e o mesmo será acessível a elas por meio do DAaaS.

Para obter a tradução de um texto no sítio do SENAPES o usuário deve selecionar o texto, e clicar no botão “Traduzir”. Então uma janela de carregamento aparecerá, e quando a tradução estiver disponível será prontamente exibida ao usuário, como pode ser observado na 4. O *plugin* de tradução para o sítio SENAPES foi desenvolvido pelo LAVID utilizando a API DAaaS.

Figura 4. Avatar traduzindo o texto selecionado

O aplicativo VLIBRAS *mobile* é disponibilizado para plataforma *android*, o que comprova que o DAaaS pode ser utilizado de forma heterogênea em diferentes meios de disponibilização. O aplicativo permite tradução a partir de texto, legenda, e também a partir do áudio, aplicando um passo adicional de conversão para texto. As Figuras 5 e 6 ilustram o processo de tradução do VLIBRAS *mobile*.

5. Considerações Finais

O presente trabalho propõe a disponibilização de um serviço de tradução automática, o DAaaS, a ser implantada em uma arquitetura distribuída, elástica e tolerante a falhas. Para este fim, utilizamos o paradigma de computação em nuvem e implantamos a arquitetura no provedor de infraestrutura AWS.

Figura 5. Opções

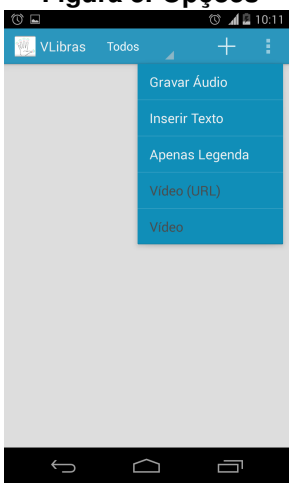


Figura 6. Avatar realizando tradução

Trabalhos futuros envolvem a validação da arquitetura proposta quanto à capacidade de provisionamento dinâmico de recursos e de tolerância a falhas com um projeto de experimentos. Também é necessário a implementação de um mecanismo de autenticação para o serviço. Atualmente, o DAaaS possui dois clientes como estudo de caso, o sítio do SENAPES e o aplicativo VLIBRAS *mobile*.

O serviço ainda apresenta um ponto crítico passível de falhas que futuramente deve ser corrigido, o *Broker*. Se, porventura, o *Broker* falhar, o DAaaS para de funcionar. Um modo rápido e fácil de consertar essa vulnerabilidade no provedor de IaaS AWS, é utilizar o *AutoScaling* aliado ao *Elastic Load Balancing*. Para tal, basta configurar o *AutoScaling* com um "gatilho" para detectar sempre que não houver instância com a *Amazon Machine Image - AMI* do *Broker* ativa, e configurar uma política de escalonamento para criar uma nova instância com a AMI do *Broker* cada vez que o gatilho for disparado. O ELB seria utilizado para redirecionar as requisições para a instância recém criada. Outra forma de utilização seria utilizar o *AutoScaling* e o ELB para manter sempre as duas instâncias do *Broker* ativas, o que aumentaria a disponibilidade do *Broker* mas também elevaria os gastos.

Por fim, agradecemos à *Amazon Web Services* pelos créditos para pesquisa que possibilitaram a realização deste trabalho.

6. Referências

References

Bala, A. and Chana, I. (2012). Fault tolerance-challenges, techniques and implementation in cloud computing. *International Journal of Computer Science Issues*, 9(1):288–293.

Brito, L. (1995). *Por uma gramática de línguas de sinais*. Tempo Brasileiro, Rio de Janeiro, Brasil.

de Araújo, T. M. U. (2012). *Uma Solução para Geração Automática de Trilhas em Língua Brasileira de Sinais em Conteúdos Multimídia*. PhD thesis, Universidade Federal do Rio Grande do Norte, Natal, Brasil.

de Góes, M. (1996). *Linguagem, surdez e educação*. Coleção Educação contemporânea. Autores Associados.

Gomes, R. C. and Góes, A. R. S. (2011). E-acessibilidade para Surdos. *Revista Brasileira de Tradução Visual*, 7(7).

Instituto Brasileiro de Geografia e Estatística (2010). Censo demográfico 2010: Características gerais da população, religião e pessoas com deficiência. Technical report.

Pivetta, E. M., Ulbricht, V., and Savi, R. (2011). Tradutores automáticos da linguagem português oral e escrita para uma linguagem visual-espacial da língua brasileira de sinais. *V CONAHPA - Congresso Nacional de Ambientes Hipermídia para Aprendizagem*.

Radhakrishnan, G. (2012). Adaptive application scaling for improving fault-tolerance and availability in the cloud. *Bell Labs Technical Journal*, 17(2):5–14.

Sindicato Nacional dos Tradutores (2013). Valores de Referência Praticados a Partir de Janeiro de 2013.