Usando a Elasticidade de Recursos em Nuvem para Aumentar o Desempenho de Aplicações Pipeline

Rodrigo da Rosa Righi[1], **Mateus Aubin**[1], **Cristiano A. da Costa**[1], **Guilherme Galante**[2]

[1]Programa Pós-Graduação em Computação Aplicada – Unisinos – Brasil
{rrrighi, maubin, cac}@unisinos.br

[2]Centro de Ciências Exatas e Tecnológicas – Unioeste – Brasil
guilherme.galante@unioeste.br

Abstract. *Although offering clear benefits for Web and business-critical demands, the use of cloud elasticity still imposes challenges when trying to reap its benefits over complex applications such as those modeled as pipelines. This often happens because replication, the standard technique for resource reorganization, by default, doesn't address both task-level parallelism and communication between working VMs. Taking into account this background, here we propose a model named Elastipipe to provide automatic elasticity over pipeline-structured applications. Elastipipe's contribution consists in a framework designed to provide the notion of flexible superscalar pipelines, in which scaling operations and load balancing take place over different elasticity units. An elasticity unit refers to a set of one or more stages of a pipeline that will be grouped together under the same elasticity rules. Using a real workload from a Brazilian company, the Elastipipe prototype presented performance gains of up to 60% when confronted with a non-elastic approach.*

Resumo. *Embora ofereça benefícios claros para aplicações Web, o uso da elasticidade em nuvem ainda impõe desafios ao tentar usá-la em aplicações complexas como aquelas modeladas em pipeline. Isso acontece porque a técnica de replicação, que é padrão na reorganização de recursos, normalmente não trata o paralelismo em nível de tarefas e comunicação entre VMs replicadas. Neste contexto propomos um modelo chamado Elastipipe para fornecer elasticidade automática sobre aplicações baseadas em linhas de montagem, ou pipelines. A contribuição de Elastipipe consiste num arcabouço projetado para oferecer o conceito de pipeline superescalar flexível, no qual operações de elasticidade e balanceamento de carga são realizadas sobre diferentes unidades de elasticidade. Uma unidade de elasticidade se refere a um conjunto de um ou mais estágios, os quais são agrupados sob as mesmas regras de elasticidade. Usando a carga de trabalho de uma empresa brasileira, o protótipo apresentou ganhos de desempenho de até 60% quando comparado com uma execução não elástica.*

1. Introdução

Vários sistemas de computação vêm sendo portas para execução em nuvem graças a capacidade de auto-escala, ou elasticidade, desse ambiente de computação distribuída [Herbst et al. 2015]. O serviço de elasticidade é melhor representado em sistemas de comércio e transações Web críticos que devem satisfazer a um determinado acordo de nível de serviço (SLA), como por exemplo, limites máximos de

tempo de resposta para diferentes tipos de requisição de entrada [Ali-Eldin et al. 2012, Islam et al. 2012, Jamshidi et al. 2014]. Além de atuar em questões de desempenho e economia de energia elétrica, a elasticidade horizontal e/ou vertical é especialmente pertinente para ambientes dinâmicos, onde a intervenção humana é difícil ou ainda, impossível [Farokhi et al. 2015]. A técnica padrão está baseada no uso de um balanceador de carga que recebe requisições e realiza operações de aumento ou redução da infraestrutura usando réplicas de máquinas virtuais (VMs). Essa interação apresenta uma arquitetura fracamente acoplada, onde réplicas não estabelecem comunicação entre si, somente cooperando através de mensagens de requisição e resposta com o balanceador de carga.

Aplicações Web como e-commerce e EFT (*Electronic Funds Transfer*) tiram proveito de uma organização de elasticidade fracamente acoplada para oferecer uma melhor experiência para o usuário [Righi et al. 2015]. Na mesma linha, essa ideia vem sendo cada vez mais explorada no contexto de sistemas de computação de alto desempenho (PAD), onde é desejável fornecer uma reorganização automática de recursos em aplicações Mestre-Escravo, Sacola de Tarefas (*Bag of Tasks*) e MapReduce [Raveendran et al. 2011]. Nesse escopo, a ideia é explorar o paralelismo de dados, uma vez que cada VM, ou réplica, executa o mesmo código sobre requisições de entrada diferentes. Entretanto, esse tipo de projeto de elasticidade não endereça aplicações complexas com comportamento de carga e comunicação entre processos imprevisíveis. Essas interações, por exemplo, podem incluir dependência de dados, causando comunicação entre processos onde origem e destino somente podem ser conhecidos em tempo de execução.

Com o intuito de mitigar esse problema, o estado-da-arte propõe como principais alternativas o uso de interfaces de programação (APIs) para extrair o paralelismo, forçando os usuários a mudarem ou inserirem diretivas no próprio código fonte para colher os benefícios dessa capacidade da computação em nuvem [Dustdar et al. 2012, Imai et al. 2012, Raveendran et al. 2011, Zhang et al. 2010]. Embora viável, essa solução pode ser classificada como custosa em relação ao tempo de implementação (*time-consuming*) e não portável, endereçando um dueto particular composto de aplicação e infraestrutura. Nesse contexto, planeja-se investigar como eficientemente integrar aplicações complexas e a capacidade de elasticidade em nuvem; mais precisamente, pretende-se focar em aplicações estruturadas em pipelines. Um pipeline é um conjunto de elementos processadores, ou estágios, conectados em série onde a saída de um elemento é a entrada do próximo. Somado ao emprego nas indústrias de CPUs e GPUs, pipelines estão presentes em workflows de PAD, bem como em procedimentos de empresas que exploram o paralelismo em nível de tarefas [Jahn et al. 2013].

Considerando o contexto exposto previamente, esse artigo apresenta o Elastipipe: um modelo de elasticidade que fornece reorganização dinâmica de recursos para aplicações estruturadas em pipeline a nível de SaaS (Software como Serviço). O usuário simplesmente adiciona tarefas para serem computadas no pipeline, o qual é gerenciado pela nuvem com operações de expansão ou consolidação de VMs. A reorganização de recursos é relevante para evitar execuções nos estados com baixa ou alta carga de CPU, enquanto mantém igual ou melhora o tempo de execução de um conjunto de tarefas quando comparado com pipelines não elásticos. A contribuição científica de Elastipipe consiste em um arcabouço para explorar o conceito de Pipeline Superescalar Elástico (PSE) no qual o administrador da aplicação pode projetar unidades de elasticidade para um con-

junto arbitrário de estágios. Por exemplo, suponha que os estágios e_1 e e_2 sejam classificados como CPU-bound, assim neste caso é possível mapeá-los para a mesma unidade de elasticidade: regras e operações de elasticidade atuam sobre VMs otimizadas para uso de CPU, as quais incluem ambos os estágios citados. Baseado no modelo proposto também foi desenvolvido um protótipo que foi avaliado com uma carga de trabalho real de uma empresa brasileira, tanto em termos de definição de estágios do pipeline quanto das tarefas que serão executadas. Os resultados são encorajadores, com ganhos de desempenho de até 60%, fazendo com que a empresa planeje migrar o seu sistema para uma nuvem na qual seja possível implementar sua aplicação conforme o modelo Elastipipe.

O restante desse artigo irá introduzir os trabalhos relacionados na Seção 2. A Seção 3 descreve o modelo Elastipipe. A Seção 4 apresenta a metodologia de avaliação, enquanto que a Seção 5 mostra uma discussão sobre os resultados. Por fim, a Seção 6 enfatiza as principais conclusões e contribuições.

2. Trabalhos Relacionados

A exploração da elasticidade em nuvem é cada vez mais perceptível em nuvens públicas e privadas (incluindo Amazon AWS, Windows Azure e RightScale para o primeiro grupo, e Eucalyptus, OpenStack e OpenNebula, para o segundo). Muitas iniciativas de pesquisa focam em utilizar esses ambientes, explorando a elasticidade através da técnica horizontal (expansão ou consolidação de VMs), onde a replicação serve para criar *templates* que servirão como unidades básicas para as operações de elasticidade [Dustdar et al. 2012, Imai et al. 2012, Loff and Garcia 2014, Marshall et al. 2010, Martin et al. 2011, Rajan et al. 2011, Raveendran et al. 2011, Righi et al. 2015, Tran et al. 2011, Zhang et al. 2010]. Para essa seção, foram selecionados quatro artigos relacionados ao tratamento de cargas de trabalho complexas [Imai et al. 2012, Marshall et al. 2010, Tran et al. 2011, Zhang et al. 2010].

Marshall, Keahey e Freeman [Marshall et al. 2010] propuseram o sistema Elastic Site, o qual eficientemente adapta os serviços oferecidos dentro de um domínio, como escalonadores em lote, armazenamento ou Web Services, para tirarem proveito de um provisionamento elástico de recursos. A ideia é promover uma extensão de clusters físicos para dentro da nuvem, onde nós trabalhadores adicionais são dinamicamente adquiridos ou desligados da nuvem baseado em mudanças na fila de trabalhos do cluster. Tran, Skhiri e Zimányi [Tran et al. 2011] desenvolveram o modelo Elastic Queue Service (EQS), o qual pode ser visto como uma arquitetura de fila de mensagens (message queue) elástica e um algoritmo elástico que pode ser adaptado para muitos sistemas de fila de mensagens com o intuito de explorar esta característica. De acordo com os autores, sistemas de filas de mensagens existentes estão somente habilitados para fornecer escalabilidade para um número reduzido de clientes e não estão aptos para tratar da dinamicidade quanto ao provisionamento de recursos. A arquitetura de EQS é realizada em camadas organizadas em componentes distribuídos, cada qual com as suas próprias regras de elasticidade. Os autores também propuseram um algoritmo de balanceamento de carga em nível de usuário, o qual é capaz de rebalancear o trabalho depois das ações de elasticidade.

Imai, Chestna e Varela [Imai et al. 2012] apresentaram o Cloud Operating System (COS), que atua como um middleware que suporta cargas de trabalho de forma elástica e escalável através da estratégia de reconfiguração baseada em migração de máquinas vir-

tuais. Enquanto outros arcabouços (por exemplo, MapReduce ou Google App Engine) forçam com que os desenvolvedores desenvolvam suas aplicações segundo uma interface de programação (API) específica, COS fornece escalabilidade através de um arcabouço de propósito geral baseado em uma linguagem de programação orientada a atores. Essa estratégia é pertinente para melhorar o tempo de migração, uma vez que somente o estado da aplicação é de fato migrado. Zhang et al. [Zhang et al. 2010] propuseram um modelo de aplicação elástico que habilita recursos da nuvem para aumentar a capacidade de computação em dispositivos móveis. A principal ideia foi a partição de uma aplicação em múltiplos componentes chamadas Weblets, os quais podem ser executados em um dispositivo móvel ou migrados para a nuvem. Atuando com conceitos similares a uma arquitetura orientada a serviço, Weblets permitem migração em nível de usuário e, assumindo que eles operam sobre HTTP, podem facilitar a interoperabilidade com provedores de nuvem. Assim, uma aplicação elástica pode aumentar as capacidades de dispositivos móveis incluindo capacidade de CPU, armazenamento e largura de banda através de recursos provenientes da nuvem.

Numa análise das quatro iniciativas citadas nessa seção, pode-se enfatizar que há uma lacuna de trabalho na exploração de um modelo de elasticidade para endereçar aplicações estruturadas em pipelines. Esse modelo de aplicação apresenta diferentes estágios de processamento, cada qual com suas próprias funcionalidades, consequentemente impondo diferentes *templates* de VMs para as operações de elasticidade. Portanto, uma solução de elasticidade deve trabalhar a reorganização de recursos em nível de estágios do pipeline para que se possa aumentar a eficiência total do sistema ao mesmo tempo que evita ignorar o fato de que cada estágio é diferente e pode exigir recursos computacionais distintos entre si.

3. Modelo Elastipipe

Essa seção apresenta o modelo Elastipipe, o qual foi desenvolvido em nível de SaaS de uma nuvem para gerenciar a elasticidade de aplicações estruturas em pipeline. Com o intuito de obter melhor desempenho, a ideia é explorar a reorganização de recursos para tornar mais rápida a execução de um conjunto de tarefas através de um mecanismo com baixo esforço por parte dos usuários, os quais não precisam mudar suas aplicações. Consequentemente, no ponto de vista do sistema, essa melhoria na perspectiva de tempo também contribui para que se obtenha um melhor índice de vazão nas tarefas. Em adição as operações de aumento de recursos que estão diretamente ligadas com o desempenho da aplicação, Elastipipe também suporta operações de consolidação de VMs, focando num melhor uso dos recursos e reduzindo o consumo energético. A Figura 1 ilustra a arquitetura de Elastipipe. O modelo pode ser visto como um arcabouço que facilita a migração de aplicações em pipeline para serem executadas na nuvem, oferecendo redução nas perspectivas de custo e riscos [Rajan et al. 2011]. Na Figura 1, os módulos Gerente de Elasticidade e Monitoramento de Recursos são comumente suportados por provedores de nuvens através de interface em linha de comando ou gráfica; enquanto que os módulos Entrada de Tarefas, Controle e Execução são de responsabilidade do modelo Elastipipe.

3.1. Arquitetura

Esta subseção descreve a arquitetura do modelo Elastipipe, detalhando seus módulos e interações. O elemento de entrada para o sistema é uma tarefa representada por T_z, as

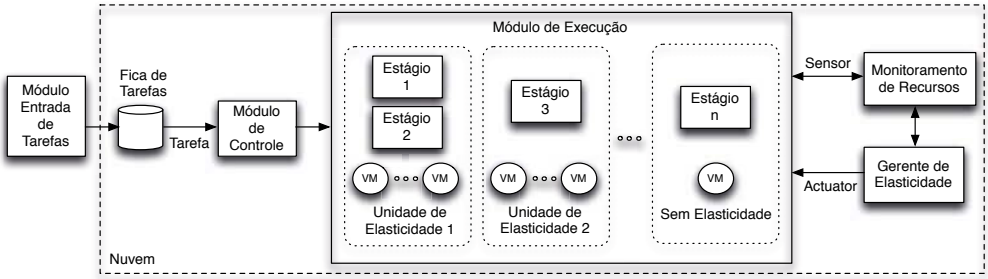


Figura 1. Arquitetura de Elastipipe. Aqui, detalha-se um exemplo no módulo de Execução: estágios podem ser agrupados sob uma mesma unidade de elasticidade, ou ainda, eles podem não apresentar a capacidade de elasticidade.

quais são numeradas da seguinte forma: $1 \leq z \leq l$. Cada tarefa possui um determinado tipo, o qual indica o pipeline com os estágios apropriados para processá-la. Cada tarefa pode apresentar um conjunto de dependências (para cada tarefa t_a, tem-se um conjunto de dependência D_a composto por tarefas arbitrárias que estão no intervalo $[1, a-1]$). Assim, a tarefa t_a irá somente entrar no pipeline quando todas as demais em D_a estiverem finalizadas. Um pipeline, representado por P_i, numerados de $1 \leq i \leq m$, é uma unidade lógica que é composto por um conjunto de estágios (representado por s_j, numerados de $1 \leq j \leq n$), podendo compartilhar estágios com outros pipelines. Uma vez que o modelo é de propósito geral, não define-se a funcionalidade de cada estágio; esse procedimento é conduzido em tempo de implementação de acordo com as necessidades da aplicação.

3.1.1. Modulo de Entrada de Tarefas e Controle

O Módulo de Entrada de Tarefas é responsável por inserir uma tarefa na fila (aqui habilitada por um sistema de fila de mensagens), de modo que o Módulo de Controle esteja habilitado para, na sequencia, ler dados dessa área compartilhada. Em particular, a fila atua como um elemento para controlar o efeito de Jitter, suavizando picos de demanda e assim habilitando a entrega regular para o Módulo de Controle. O Módulo de Controle é responsável pelas seguintes funcionalidades: (i) capturar uma tarefa da fila de tarefas; (ii) validar uma tarefa; (iii) controlar o paralelismo; (iv) tratar erros e; (v) disparar e monitorar tarefas. Após capturar uma tarefa que deve ser computada, a funcionalidade ii garante a integridade da tarefa e sua aderência ao Módulo de Execução. A ideia é reduzir a quantidade de erros na aplicação devido a mensagens desconhecidas ou mal formadas, evitando problemas que possam interferir no desempenho de outras tarefas que já estejam em execução. O controle de paralelismo é associado a definição de qualidade de serviço (QoS) para gerenciar a superescalaridade de cada pipeline. Nesse sentido é possível definir um número mínimo e máximo para tarefas que serão executadas em paralelo, o qual impacta no número de VMs usado para tratar cada unidade de elasticidade (que por sua vez pode ser composta por um único estágio ou uma coleção deles, mais detalhes na Subseção 3.2). O tratamento de erros endereça exceções, assim como a repetição da execução de tarefas que falharam devido a problemas de comunicação ou infraestrutura. A consolidação de uma VM que tenha pelo menos uma tarefa em execução é um exemplo da necessidade de relançamento de uma ou mais tarefas.

Para viabilizar a quinta funcionalidade, foi projetado um despachador de tarefas que identifica qual pipeline é o destino de uma determina tarefa, além de controlar o fluxo

de tarefas no pipeline e a dependência entre tarefas. Para cobrir tais casos, a estratégia foi a mesma aplicada em pipelines de microprocessadores, onde conflitos de dados ou controle podem prejudicar a eficiência do pipeline com pausas nas execuções. Quando situações como essa são detectadas, define-se que o estado da tarefa é em espera até que todas as dependências estejam completamente satisfeitas. A Figura 2 ilustra um exemplo de despacho onde cinco tarefas e três pipelines estão envolvidos. O fluxo de execução de cada tarefa é o seguinte: o Módulo de Controle decide o pipeline correto e despacha a tarefa para o primeiro estágio, recebendo a saída da computação na sequencia e, então, despachando novamente a tarefa (e seu conjunto de dados de saída) para o próximo estágio e assim por diante. Em outras palavras, baseado em [Rajan et al. 2011], foi projetada uma organização mestre-escravo para gerenciar a execução das tarefas, onde o Módulo de Controle representa o mestre e os estágios de processamento, os escravos. Por fim, é possível observar na Figura 2 que um estágio em particular pode aparecer em mais do que um pipeline. Isso é esperado uma vez que pipelines, na modelagem do Elastipipe, são fluxos lógicos de trabalho e a execução é tratada individualmente através dos estágios.

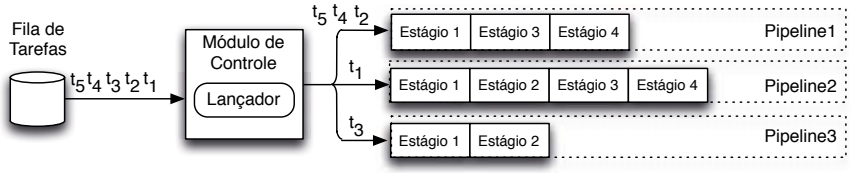


Figura 2. Execução de tarefas em diferentes pipelines.

3.1.2. Módulo de Execução

Este módulo abriga os componentes responsáveis pela execução de cada estágio do pipeline. O projeto do módulo de execução foi feito tendo em mente o conceito de microserviços [Namiot and Sneps-Sneppe 2014]: um microserviço representa um pequeno e independente serviço o qual se comunica através de protocolos simples e bem definidos. O uso de microserviços tem crescido como blocos básicos para a programação em nuvem dada as características que proporciona como melhor escalabilidade, *composability* e facilidade de mapeamento para recursos [Newman 2015]. Newman afirma que é comumente observável que transações CPU ou IO-bound são endereçadas para uma mesma unidade de elasticidade a qual pode penalizar um dos tipos mencionados ou acarretar o desperdício de recursos [Newman 2015]. Através da adoção da abordagem de microserviços, administradores podem escalar cada unidade de maneira que melhor se encaixe às suas necessidades, assim obtendo reduções de tempo de execução e custos.

Embora isolados em processos diferentes, microserviços podem se comunicar e, ainda, depender um do outro para que um objetivo comum seja alcançado. Tal afirmação facilita a construção de aplicações multi-plataforma, a emersão de novas aplicações através da combinação de microserviços, e facilita a criação de pipelines definidos em software. Através da adoção de microserviços para a implementação dos estágios do pipeline, é possível melhorar o reconhecimento de possíveis gargalos e promover um diagnóstico mais preciso em nível de aplicação e infraestrutura no momento de detecção de tarefas em falha. Por fim, microserviços são pertinentes como uma unidade de elasticidade e para implementar aplicações MPMD (*Multiple programs, multiple data streams*) como as que são organizadas em pipelines. Nessa organização, é possível explorar o pa-

ralelismo em nível funcional (ou de código) com múltiplos processadores autônomos que operam em diferentes nós, representados aqui por estágios do pipeline.

3.2. Modelo de Elasticidade

Elastipipe tira vantagem do monitoramento de recursos e gerenciamento de elasticidade já disponíveis em provedores de nuvem. Portanto, essa subseção objetiva classificar a estratégia de elasticidade de Elastipipe, enquanto também apresenta os requisitos que o provedor de nuvem deve preencher. A abordagem de elasticidade adotada é horizontal, automática e reativa. Levando em conta a palavra-chave reativa, Elastipipe considera: (i) dois limiares (*thresholds*), um mínimo e outro máximo, para guiar as ações de elasticidade quando violados; (ii) a técnica de replicação sem um único *template* de VM, mas sim uma coleção deles que varia de acordo com o número de unidades de elasticidade; (iii) um balanceador de carga por unidade de elasticidade. Essa combinação foi selecionada levando em consideração as seguintes premissas: facilidade de configuração, eficiência das ações de elasticidade e disponibilidade de implementação na maioria dos ambientes de computação em nuvem vigentes.

A abordagem de elasticidade horizontal é conhecida pelas operações de expansão ou consolidação de VMs. A abordagem vertical (também conhecida por redimensionamento de recursos) foi preterida porque é limitada às capacidades de um único nó computacional (CPU, memória, disco e rede), não sendo uma solução escalável para suportar aplicações do tipo pipeline. Do ponto de vista do usuário, a reorganização dos recursos é feita de forma automática e de acordo com a demanda de computação, sem intervenções em tempo de execução. Os únicos parâmetros necessários para o modelo são os limiares usados para guiar a elasticidade e as ações pré-configuradas pelo administrador da aplicação. Basicamente, "se" uma determinada regra for satisfeita, "então" uma ação de elasticidade é disparada para otimizar os objetivos estabelecidos pelo administrador, os quais podem ser desempenho, energia, custo ou, ainda, uma combinação deles.

Limiares, regras e ações são usados para controlar unidades de elasticidade. A Figura 3 ilustra um exemplo de quatro mapeamentos, três deles com unidades de elasticidade. Uma unidade de elasticidade pode ser vista como um agrupamento de um ou mais estágios sobre o mesmo *template* de VM; portanto, todas as operações de elasticidade irão sempre instanciar ou consolidar este elemento funcional. Cada unidade de elasticidade possui as suas próprias regras e ações, em adição a um balanceador de carga que decide a réplica alvo que esta menos carregada e um estágio em particular (quando existe mais de um estágio na mesma VM) para executar determinada tarefa. Balanceadores de carga atualizam as suas conexões com as réplicas a cada alteração na infraestrutura. Usando unidades de elasticidade como blocos básicos, Elastipipe propõe o conceito de Pipeline Superescalar Elástico, como segue:

- Definição 1. Pipeline Superescalar Elástico: O usuário ou administrador da aplicação está apto para configurar tantas unidades de elasticidade quanto requeridas para executar a sua aplicação organizada em pipeline, provendo o isolamento da elasticidade e a possibilidade de tirar proveito da decomposição funcional para agrupar estágios de acordo com a natureza de recursos exigida em cada estágio (CPU, memória, disco ou rede).

É importante enfatizar que a noção de pipeline é lógica, ou seja, uma unidade de elasticidade pode servir vários pipelines. Por exemplo, é possível acomodar os estágios

s_1, s_2 e s_3 na mesma unidade de elasticidade, mas s_1 e s_2 pertencerem ao pipeline P_1 enquanto s_2 e s_3 pertencem ao pipeline P_2. Por fim, considerando que aplicações complexas são compostas por vários estágios, cada qual com os seus requisitos de recursos, os conceitos de unidade de elasticidade e Pipeline Superescalar Elástico são úteis para balancear a duração de cada estágio. Esse equilíbrio é essencial para evitar gargalos no pipeline, uma vez que o estágio mais lento será responsável pelo acumulo de tarefas, prejudicando a eficiência da técnica de linha de montagem [Jahn et al. 2013]. Como um efeito desejável, o uso de unidades de elasticidade também contribui para aumentar a confiabilidade do sistema: o uso de réplicas favorece a tolerância a falhas. Assim, falhas sobre um único componente não irão tornar a aplicação indisponível [Zhang et al. 2010].

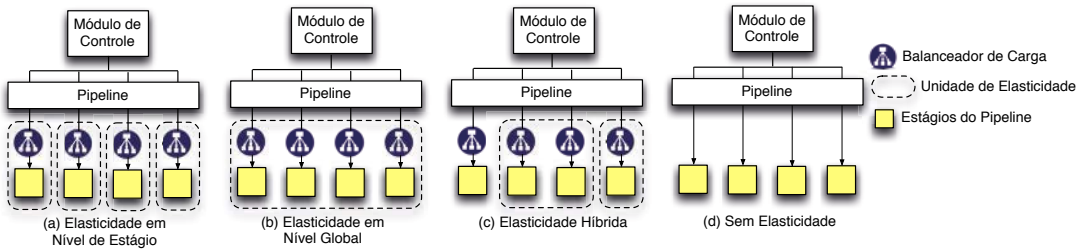


Figura 3. Unidade de elasticidade e Balanceador de Carga: Permitindo diferentes estratégias de elasticidade de acordo com o comportamento de cada estágio.

4. Metodologia de Avaliação

Essa seção descreve o protótipo Elastipipe e parâmetros envolvidos na sua avaliação.

4.1. Protótipo

Foi desenvolvido um protótipo no qual as tarefas de entrada são representadas por arquivos texto. Em adição, foi selecionado o ambiente Amazon Web Services (AWS) como provedor de nuvem e um conjunto de softwares da Microsoft para o desenvolvimento dos módulos do Elastipipe: compilador C#, arcabouço de desenvolvimento .NET e banco de dados SQL Server. Quanto à AWS, foram usados os seguintes componentes: (i) EC2 para gerenciamento de máquinas virtuais; (ii) CloudWatch para monitoramento; (iii) AutoScaling para gerenciamento e configuração da elasticidade; (iv) Elastic Load Balancing (ELB) para habilitar balanceamento de carga dentro das unidades de elasticidade; (v) SQS como serviço de mensagens distribuído e; (vi) S3 para armazenamento. O protótipo opera com o seguinte caminho de execução: (a) um componente primeiro detecta a presença de arquivos em S3 e gera uma mensagem de notificação que é inserida na fila de mensagens; (b) outro componente, responsável por capturar as mensagens dessa fila, detecta a notificação e inspeciona o arquivo para detectar qual o pipeline que a tarefa pertence; (c) levando em conta essa informação, o Módulo de Controle despacha a tarefa para o serviço apropriado para executar cada estágio do pipeline. Uma vez que os serviços operam de forma isolada e sem estado, eles podem ser seguramente instanciados quantas vezes for necessário sem comprometer a integridade da aplicação.

Foram implementados dois pipelines: P_1 e P_2. P_1 possui 5 estágios como segue: (i) validação de tarefas (caráter CPU-bound); (ii) conversão de leiaute (caráter CPU-bound); (iii) análise de consistência (caráter memory-bound); (iv) persistência em banco de dados (disk-bound); (v) estágio de distribuição, o qual é responsável por enviar um relatório para outras fontes de dados e aplicações relacionadas (network-bound). P_2 representa somente dois estágios: (i) validação de esquema, o qual é responsável pela inspeção

Tabela 1. Descrição dos cenários propostos.

Cenário	Elasticidade	Descrição da VM	Regras de Elasticidade	
1	sem elasticidade	m4.large (fixa para todos estágios)	—	
2	elasticidade global	m4.large (única unidade de elasticidade para todos estágios)	$\uparrow CPU > 75\%$	$\downarrow CPU < 35\%$
3	elasticidade funcional	c4.large (unidade otimizada para CPU)	$\uparrow CPU > 75\%$	$\downarrow CPU < 35\%$
		r3.large (unidade otimizada para Memória)	$\uparrow RAM > 75\%$	$\downarrow RAM < 35\%$
		i2.xlarge (unidade otimizada para Disco/IO)	$\uparrow IOPS > 3k$	$\downarrow IOPS < 1k$
		m4.large (unidade otimizada para Rede)	$\uparrow MBps > 40$	$\downarrow MBps < 10$

de uma tarefa e a sua validação de acordo com leiautes pré-definidos presentes em XML Schema (CPU- e memory-bound); (ii) persistência em banco de dados (disk-bound). Considerando a definição dos estágios foram definidas 5 unidades de elasticidade usando o pacote AutoScaling da AWS: (a) Otimizado para CPU, para os estágios de validação, conversão e esquema; (b) otimizado para memória, para o estágio de consistência; (c) otimizado para rede, para o estágio de distribuição; (d) otimizado para disco, para o estágio de persistência em banco de dados; (e) tratamento global onde todos os estágios são inseridos juntos em uma única unidade de elasticidade.

4.2. Definindo a Carga de Trabalho do Pipeline e Métricas de Avaliação

Para os testes, foi utilizada uma carga de trabalho capturada a partir de amostras provenientes de uma empresa parceira localizada no parque tecnológico da Unisinos (Tecnosinos). A carga se refere a traços capturados durante uma semana, com um total de 11.459 tarefas em forma de arquivos texto. Cada arquivo varia entre 3.500 e 480.000 linhas de dados, gerando um total de até 110 megabytes de dados. Em adição, foram criados três cenários para avaliar diferentes estratégias de uso da elasticidade, como pode ser visto na Tabela 1. Essa tabela também apresenta as regras de elasticidade para cada unidade de elasticidade. Três métricas são consideradas para a avaliação do protótipo Elastipipe: tempo, recurso e custo. A primeira se refere a desempenho, apresentando o tempo total de execução para todo o conjunto de dados. Através de observações periódicas feitas pela ferramenta CloudWatch, foi realizada a estimativa da métrica de recursos levando em consideração uma configuração da nuvem com i VMs e $obs(i)$ como o número de observações em que essa configuração aparece. Por fim, usa-se a noção de custo de sistemas paralelos para que seja obtida a métrica custo: multiplicação de tempo por recursos. As Equações 1 e 2 computam os valores das métricas recurso e custo. Na Equação 1, v representa o máximo de VMs que podem ser instanciados nas unidades de elasticidade.

$$recursos = \sum_{i=1}^{v} (i \times obs(i)) \tag{1}$$

$$custo = recursos \times tempo \tag{2}$$

5. Resultados

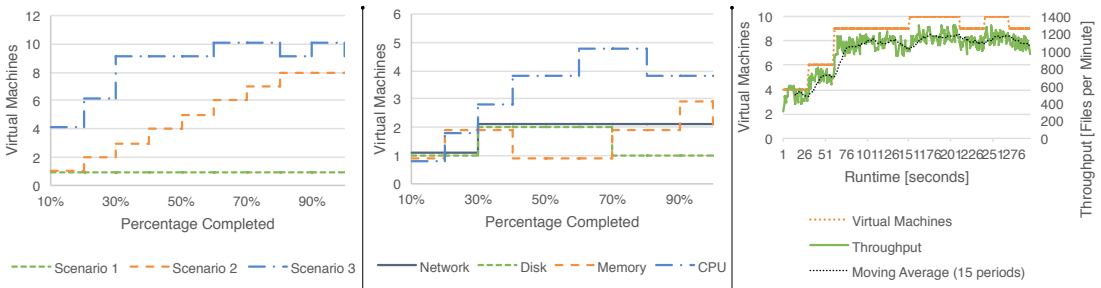
A Tabela 2 apresenta os resultados. O tempo de execução total nessa tabela confirma os benefícios de se usar a elasticidade em nuvem para tratar demandas de aplicações pipeline. A estratégia de mapeamento de serviços (estágios do pipeline) de acordo com o comportamento do seu recurso alvo (CPU-bound, network-bound, memory-bound ou disk-bound), em adição a regras particulares para endereçá-los, foi responsável pelas melhores taxas de desempenho a favor do cenário 3. Considerando o desempenho do cenário 1 como um fator de normalização, foram obtidas melhorias de 62% e 72% quando habilita-se a

Tabela 2. Resultados. Cenário 1: sem elasticidade; Cenário 2: elasticidade através de uma única unidade de elasticidade; Cenário 3: elasticidade com multiplas unidades de elasticidade

Cenário	Duração Média por Tarefa (s)	Vazão por Minuto	Duração Total	Número Médio de VMs	Observações	Uso de Recursos	Custo
1	5,52	108,70	17:34:12	1,0	1.054	1.054*1 = *1.054*	18.518
2	2,62	594,74	06:45:36	5,2	405	40*1 + 40*2 + 40*3 + 40*4 + 40*5 + 40*6 + 40*7 + 120*8 = *2.080*	14.056
3	2,13	1.007,45	04:55:12	8,5	295	29*4 + 29*6 + 145*9 + 87*10 = *2.465*	12.131

elasticidade. Comparando somente os cenários 2 e 3, o último atingiu 27% de ganho de desempenho enquanto usa 39% mais VMs para atingi-lo.

A Figura 4 (a) ilustra o comportamento de alocação de recursos ao longo da execução em cada cenário. Pode-se observar um gargalo de desempenho no cenário 2: para a maior parte do tempo de execução é utilizado um número saturado de VMs, estabilizando perto do final da execução. Isto explica o efeito escada nesse cenário: uma nova VM é alocada, mas logo se torna sobrecarregada, implicando em uma nova ação de elasticidade. Esse gargalo é minimizado no cenário 3 através da distribuição de unidades de elasticidade desde o inicio, percebe-se portanto um equilíbrio no uso de recursos mais cedo (veja a Figura 4 (b)).



(a) Visão histórica da alocação de recursos

(b) Visão detalhada da alocação de recursos no cenário 3

(c) Relação entre alocação de recursos e vazão no cenário 3

Figura 4. Observando a alocação de recursos e vazão ao longo da execução.

O cenário 1 utiliza uma única VM para acomodar os quatro conjuntos de estágios. Nesse contexto, utiliza-se a família de máquinas $m4$, a qual é classificada pela AWS como servidores de propósito geral que balanceiam requisitos de processamento, disco, memória e rede. Esse mapeamento é responsável pelo menor consumo de recursos obtidos nos testes. Entretanto, ele apresenta o pior custo uma vez que o tempo de execução foi muito maior do aquele observado nos demais cenários. O cenário 2 também começa com uma única VM, mas a elasticidade foi responsável por aumentar o consumo de recursos enquanto melhora também o tempo de execução. Em média, tem-se 26 tarefas sendo executadas concorrentemente nos pipelines quando habilita-se elasticidade sobre uma única unidade de elasticidade que abriga todos os estágios.

O cenário 3 trabalha com agrupamento de estágios em diferentes unidades de elasticidade de acordo com as características dos recursos. Nessa linha, o fato de usar mais nós neste cenário foi responsável pelo maior uso de recursos. Em particular, Baliga et al. [Baliga et al. 2011] afirmam que o número de VMs em um nó não é o maior fator de

influência para o consumo de energia; ao invés disso, eles informam que o fator mais relevante é se o nó está ligado (energizado) ou não. Sobrepondo o cenário 2, no cenário 3 foi medida uma quantidade de 36 tarefas em paralelo sendo executadas nos pipelines, quase dobrando a vazão de tarefas por minuto (de 594,74 no cenário 2 para 1.007,45 no cenário 3). A Figura 4 (c) ilustra a vazão de tarefas por minuto e sua relação com a alocação de recursos. Por fim, observa-se que o uso da decomposição funcional dos estágios do pipeline para projetar as unidades e elasticidade foi o fator que mais influenciou o desempenho, também facilitando a manutenção e a detecção de problemas.

6. Conclusão

O conceito de linhas de montagem, originado na indústria automotiva e popularizado em projetos de microprocessadores, está presente em cargas de trabalho complexas oriundas de uma variedade de áreas do conhecimento. Neste contexto, este artigo propôs um modelo chamado Elastipipe para abordar essas aplicações apresentando, ao mesmo tempo, o conceito de Pipeline Superescalar Elástico para apoiar a reorganização dinâmica de recursos. Essa reorganização é atingida através da definição de unidades de elasticidade por usuários ou administradores da aplicação, podendo contemplar um ou mais estágios do pipeline. Cada operação de expansão ou consolidação de VMs considera essa unidade como grão de recurso para aumentar ou reduzir a quantidade de recursos que melhor mapeia para a execução da aplicação em um determinado momento.

Os resultados com um protótipo revelaram a viabilidade do uso da elasticidade em nuvem, onde cenários com elasticidade apresentaram melhores taxas de custo quando comparados a uma abordagem não-elástica. Além disso, a decomposição funcional dos estágios em unidades de elasticidade de acordo com suas características (CPU-, memory-, network- ou disk-bound) foi decisiva para a obtenção dos melhores resultados de desempenho. Esse cenário de implantação alcançou ganhos de 27% quando comparado com uma situação em que todos os estágios estão concentrados em uma única unidade de elasticidade. Investigações futuras incluem o uso de cargas de trabalho sintéticas para analisar o comportamento do modelo com diferentes características de carga: ascendente, descendente e flutuante (onda).

Agradecimentos

Os autores gostariam de agradecer aos seguintes órgãos: CNPq, CAPES e FAPERGS.

Referências

Ali-Eldin, A., Tordsson, J., and Elmroth, E. (2012). An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212.

Baliga, J., Ayre, R., Hinton, K., and Tucker, R. (2011). Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, 99(1):149–167.

Dustdar, S., Guo, Y., Han, R., Satzger, B., and Truong, H.-L. (2012). Programming directives for elastic computing. *Internet Computing, IEEE*, 16(6):72–77.

Farokhi, S., Jamshidi, P., Brandic, I., and Elmroth, E. (2015). Self-adaptation challenges for cloud-based applications : A control theoretic perspective. In *10th International Workshop on Feedback Computing (Feedback Computing 2015)*. ACM.

Herbst, N. R., Kounev, S., Weber, A., and Groenda, H. (2015). Bungee: An elasticity benchmark for self-adaptive iaas cloud environments. In *Proc. Int. Symp. Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15, pages 46–56, IEEE.

Imai, S., Chestna, T., and Varela, C. A. (2012). Elastic Scalable Cloud Computing Using Application-Level Migration. In *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pages 91–98, Honolulu. IEEE.

Islam, S., Keung, J., Lee, K., and Liu, A. (2012). Empirical prediction models for adaptive resource provisioning in the cloud. *Future Gener. Comput. Syst.*, 28(1):155–162.

Jahn, J., Pagani, S., Kobbe, S., Chen, J.-J., and Henkel, J. (2013). Optimizations for configuring and mapping software pipelines in many core systems. In *Proc. Annual Design Automation Conference*, DAC '13, pages 130:1–130:8, ACM.

Jamshidi, P., Ahmad, A., and Pahl, C. (2014). Autonomic resource provisioning for cloud-based software. In *Proc. of the 9th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 95–104, ACM.

Loff, J. and Garcia, J. (2014). Vadara: Predictive Elasticity for Cloud Applications. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 541–546, Singapore. IEEE.

Marshall, P., Keahey, K., and Freeman, T. (2010). Elastic Site: Using Clouds to Elastically Extend Site Resources. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 43–52, Melbourne. IEEE.

Martin, P., Brown, A., Powley, W., and Vazquez-Poletti, J. L. (2011). Autonomic management of elastic services in the cloud. In *2011 IEEE Symposium on Computers and Communications (ISCC)*, pages 135–140, Kerkyra. IEEE.

Namiot, D. and Sneps-Sneppe, M. (2014). On Micro-services Architecture. *International Journal of Open Information Technologies*, 2(9):24–27.

Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc., Sebastopol.

Rajan, D., Canino, A., Izaguirre, J. A., and Thain, D. (2011). Converting a high performance application to an elastic cloud application. In *Proc. Int. Conf. on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 383–390, IEEE.

Raveendran, A., Bicer, T., and Agrawal, G. (2011). A framework for elastic execution of existing mpi programs. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 940–947.

Righi, R., Rodrigues, V., Andre daCosta, C., Galante, G., Bona, L., and Ferreto, T. (2015). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *Cloud Computing, IEEE Transactions on*, PP(99):1–1.

Tran, N.-L., Skhiri, S., and Zimányi, E. (2011). EQS: An Elastic and Scalable Message Queue for the Cloud. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 391–398, Athens. IEEE.

Zhang, X., Jeong, S., Kunjithapatham, A., and Gibbs, S. (2010). Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms. In *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, volume 48, pages 161–174. Springer, Heidelberg.