DARLAN SEGALIN

DETECÇÃO INTELIGENTE DE SURTOS DE PROCESSAMENTO NA COMPUTAÇÃO EM NUVEM

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

CURITIBA

2013

DARLAN SEGALIN

DETECÇÃO INTELIGENTE DE SURTOS DE PROCESSAMENTO NA COMPUTAÇÃO EM NUVEM

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Altair O. Santin

CURITIBA

2013

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

Segalin, Darlan
S454d Detecção inteligente de surtos de processamento na computação em
2013 nuvem / Darlan Segalin ; orientador, Altair O. Santin. – 2013.
xi, 45 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2013
Bibliografia: f. 43-45

1. Computação em nuvem. 2. Sistema de computação virtual. 3. Informática. I. Santin, Altair Olivo. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD 20. ed. – 004

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

PUCPR
GRUPO MARISTA

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 17/2013

Aos 17 dias do mês de Setembro de 2013 realizou-se a sessão pública de Defesa da Dissertação **" Detecção Inteligente de Surtos de Processamento na Computação em Nuvem"** apresentado pelo aluno **Darlan Segalin,** como requisito parcial para a obtenção do título de Mestre em Informática, perante uma Banca Examinadora composta pelos seguintes membros:

Prof. Dr. Altair O. Santin PUCPR (Orientador)	*(assinatura)*	Aprov. *(Aprov/Reprov)*
Prof. Dr. Alcides Calsavara PUCPR	*(assinatura)*	APROVADO *(Aprov/Reprov)*
Prof. Dr. Luiz Eduardo. S. Oliveira UFPR	*(assinatura)*	APROV *(Aprov/Reprov)*
Prof. Dr. Carlos Alberto Maziero UTFPR	*(assinatura)*	aprov *(Aprov/Reprov)*

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado Aprovado (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.

Prof. Dr. Mauro Sério Pereira Fonseca
Diretor do Programa de Pós-Graduação em Informática

Programa de Pós-Graduação em Informática
PPGIa
PUCPR

Dedico este trabalho aos meus familiares que sempre me apoiaram e aconselharam ao longo de minhas conquistas.

Agradecimentos

Agradeço a Deus por me dar saúde para conquistar meus objetivos, ao meu orientador, Dr. Altair O. Santin, pelo apoio necessário para elaboração deste trabalho, também agradeço ao Cleber Olivo, pela ajuda e orientação nos testes de classificação de dados, ao professor Dr. Carlos A. Maziero, a minha família e em especial, ao meu irmão, Liandro Segalin, pelo constante apoio.

Sumário

Sumário iii
Lista de Figuras v
Lista de Tabelas vii
Lista de Abreviaturas viii
Resumo ix
Abstract x
Capítulo 1 1
Introdução 1
1.1. Motivação 3
1.2. Objetivo Geral 4
1.3. Objetivos Específicos 4
1.4. Contribuições 5
1.5. Estrutura do Documento 5
Capítulo 2 6
Fundamentação Teórica 6
2.1 Modelos de Serviços 7
2.1.1 Infraestrutura como Serviço (IaaS) 8
2.1.2 Plataforma como Serviço (PaaS) 8
2.1.3 Software como Serviço (SaaS) 9
2.2 Modelos de Implementação 9
2.3 Papéis e vantagens de utilizar a computação em nuvens 10
2.4 Virtualização 12

2.5 Conceitos de aprendizado de máquina 12
Capítulo 3 15
Trabalhos Relacionados 15
3.1 Provisionamento de Recursos em Máquinas Virtuais 15
3.2 Balanceamento de Carga na Virtualização 17
3.3 Controle e Provisionamento de Recursos por políticas 18
3.4 Caracterizações de Spikes e Flash Crowds 21
3.5 Considerações 23
Capítulo 4 27
Proposta 27
4.1 Definição do ambiente e atributos de utilização de recursos da VM 29
4.2 Sintetização de cargas de trabalho 30
4.3 Aplicando aprendizado de máquina 34
4.4 Analisando os vetores do Dataset 37
4.5 Seleção de atributos e novos testes 38
Capítulo 5 41
Conclusão 41
Referências 43

Lista de Figuras

Figura 2.1– Modelo de Serviços. Adaptado de [08]. 7
Figura 2.2– Diagrama da camada de virtualização do Hypervisor. Adaptado de [14] 12
Figura 2.3 – Exemplo de vetores de atributos e classes. Adaptado de [15]. 13
Figura 3.1– Fluxos de trabalhos para recursos elásticos. Adaptado de [20] 19
Figura 3.2- Comparação de estratégias de agendamento para minimizar desperdício em um fluxo de trabalho de exemplo. Adaptado de [20].....20
Figura 3.3– Diagrama de arquitetura do framework de controle. Adaptado de [04].....21
Figura 3.4– Perfil de Carga de trabalho de Spike. Adaptado de [21].....22
Figura 3.5 – Taxa de acerto dos quatro melhores classificadores testados. Adaptado de [22]. 23
Figura 4.1– Visão geral do esquema incremental de escalabilidade e elasticidade da proposta. 27
Figura 4.2– Log para coleta do atributo *TotalLoadCPU* gerado em arquivo texto pelo *System Monitor*.....30
Figura 4.3- Código do programa que gera cargas de trabalho 31
Figura 4.4– Gráfico de consumo carga de trabalho (Normal).....32
Figura 4.5- Gráfico de consumo carga de trabalho (Spike 01). 32
Figura 4.6- Gráfico de consumo carga de trabalho (Spike 02). 32
Figura 4.7- Gráfico de consumo carga de trabalho (Spike 03) 32
Figura 4.8 - Gráfico de consumo carga de trabalho (Spike 04). 33
Figura 4.9 - Gráfico de consumo carga de trabalho (Spike 05). 33
Figura 4.10 - Gráfico de consumo carga de trabalho (Spike 06) 33
Figura 4.11- Gráfico de consumo carga de trabalho (Spike 07) 33
Figura 4.12- Gráfico de consumo carga de trabalho (Spike 08) 33
Figura 4.13-Gráfico de consumo carga de trabalho (Flash 09).....33
Figura 4.14- Gráfico de consumo carga de trabalho (Flash 10).....33
Figura 4.15-Gráfico de consumo carga de trabalho (Flash 11).....33
Figura 4.16-Gráfico de consumo carga de trabalho (Flash 12).....33

Figura 4.17- Gráfico de consumo carga de trabalho (Flash 13).....33
Figura 4.18– Fragmento dos arquivos com vetores de características.....34
Figura 4.19 – Esquema de validação cruzada aplicada na proposta.....36

Lista de Tabelas

Tabela 4.1– Tabela com o nome das métricas coletadas e sua descrição.....29
Tabela 4.2– Combinação de arquivos e resultados de treinamento e teste.....37
Tabela 4.3– Resultados de treinamento e teste todos os perfis juntos.....38
Tabela 4.4– Atributos selecionados pela técnica Filter.....39
Tabela 4.5– Nova combinação de arquivos, treinamento e testes.....40

Lista de Abreviaturas

SaaS	Software As a Service
IaaS	Infrastructure As a Service
PaaS	Platform As a Service
QoS	Quality of Service
IP	Internet Protocol
OS	Operating System
API	Application Programming Interface
EC2	Amazon Elastic Compute Cloud
SLA	Service Level Agreement
VM	Virtual Machine
NIST	National Institute of Standards and Technology
CPU	Central Processing Unit
SVM	Support Vector Machines

Resumo

Em um serviço de larga escala baseado na computação em nuvem, é comum acontecerem surtos (mudanças consideravelmente rápidas e inesperadas) nos padrões de processamento (carga de trabalho). Nesse caso, o surto de processamento pode ser momentâneo (Spike) ou mais duradouro (flash crowd), o problema é que não se sabe a real necessidade de alocação de uma nova máquina virtual (VM). Na computação em nuvem, a elasticidade provê a VM em função da demanda de processamento, porém, se for um Spike, situação em que o custo para alocação é maior que o benefício do processamento, o provedor de nuvem terá prejuízo. Em contrapartida, se a VM não for provida para tentar mitigar o prejuízo e a demanda for um Flash Crowd, a aplicação será penalizada. Assim, se faz necessária uma heurística para distinguir os padrões de processamento. Este trabalho propõe o uso do *support vector machine* (SVM) para classificar o processamento de surtos e diferenciar seus tipos, visando prover VM em uma estratégia que minimiza perdas para o provedor e para a aplicação. Os testes realizados como prova de conceito, utilizando um ambiente real, mostram que em mais de 99% dos casos o modelo obtido com o classificador SVM distingue os dois tipos de processamento.

Palavras-Chave: Computação em nuvem, Máquinas Virtuais, Surtos de Processamento, *Spikes*, *Flash Crowds*, classificadores, SVM.

Abstract

Processing surges are fast and unexpected changes in the processing demand that commonly occur in cloud computing. The cloud elasticity enables to handle processing surges, increasing and decreasing resources as required. However, a surge can be very fast, where the overhead to provide more resource is greater than the processing benefit. On the other hand, if the surge is slow and continuous and the required resources are not provided, the application performance may be impaired or interrupted. This paper presents a machine learning-based approach to detect and classify processing surges, in order to improve the cloud resource management, minimizing losses for the application and cloud provider. We use a real cloud provide data set to select features, to construct the classifier and to test our approach, that successfully detected and classified 99% of the processing surges.

Keywords: Cloud Computing, Virtual Machine, Processing Surges, Spikes, Flash Crowds, Classifiers, SVM.

Capítulo 1

Introdução

A computação em nuvem está transformando a maneira como as pessoas usam computadores e também altera a maneira como os serviços são executados em rede, pois o prestador de um serviço (da nuvem) é capaz de presta-los dinamicamente para atender a demanda dos clientes. A nuvem provedora da infraestrutura o faz de forma dinâmica, oferecendo os recursos sob demanda para os clientes. O provedor faz isso utilizando a virtualização, visando prover máquinas virtuais a múltiplos clientes que compartilham o mesmo servidor físico [01].

O provisionamento dinâmico de recursos não é uma técnica nova, pois fora amplamente estudado no passado. Essa abordagem tipicamente envolve: (1) construir um modelo de desempenho que prevê o número de instâncias requeridas para atender a demanda em cada nível particular a fim de atender os requisitos de QoS (qualidade do serviço); (2) periodicamente prever a demanda futura e requerimentos de recursos usando um modelo de performance; (3) alocar recursos automaticamente usando as necessidade de recursos previstos. O modelo de desempenho do aplicativo pode ser construído utilizando várias técnicas incluindo a teoria das filas (*Queing Theory*) [02], a teoria do controle (*Control Theory*) [03] e o aprendizado estatístico de máquina (*Statistical Machine Learning*) [04].

Quando se fala sobre picos em cargas de trabalho, caracteriza-se como *Spike* (pico momentâneo de processamento) qualquer situação em que o *overhead* não justifica o provisionamento de recursos, quando o *overhead* é maior que o processamento do usuário. Existem situações que inicialmente podem ser confundidas com *Spikes*, pois consomem importantes quantidades de recursos (*Flash Crowds*), mas essas demandas são contínuas e

duradouras, nesse caso a instanciação se justificaria.

O termo *Spike* é geralmente utilizado para se referir a um aumento inesperado no volume total da carga de trabalho. No caso de servidores *Web*, tais picos podem ser absorvidos pela adição de mais servidores ou usando uma combinação de *switches*, DNS e replicação para redireccionar a carga para novos servidores. Simplesmente instanciando novos recursos ao serem detectados *Spikes*, pode-se gerar um custo desnecessário de migração.

Em uma situação na qual não se sabe a real necessidade de alocação de uma nova VM, é necessário detectar se o evento é um *Spike* ou *Flash Crowd*. Se uma VM não é alocada e a demanda da carga de trabalho for um *Flash Crowd*, estar-se-á prejudicando uma aplicação; e se for feita a alocação de uma VM e for uma carga de trabalho do tipo *Spike*, recursos serão desperdiçados. Por outro lado, a instanciação de recursos dentro de uma mesma máquina física costuma ser relativamente barata e o custo de não alocação para o caso de *Flash Crowd* é mais grave que o de alocação de recurso oriundo de um *Spike*. Este último tem o agravante de ser dificilmente detectável, pois a aplicação teria que informar o que deseja consumir de recursos para avaliar se isso vai gerar um *Spike*. Porém, esse não é um cenário real, porque as aplicações alocam e usam recursos sem informar quanto usarão de antemão.

A melhor estratégia é alocar a VM assumindo recursos locais para atender a demanda de um *Spike*. No entanto, se ao identificar que se trata de um *Flash Crowd*, deve-se realocar a VM para atender da melhor forma a demanda de processo. Se não houverem recursos locais em um ambiente de nuvem híbrida, por exemplo, pode ser demandada a criação de novas instâncias de VM em uma nuvem pública, devido a uma carga de trabalho do tipo *Flash Crowd*. Todavia, isso não se justifica no caso de *Spikes*, pois geraria um custo alto sem ganho significativo, resultante do atendimento da demanda.

Assim, se for possível distinguir o *Spike* do *Flash Crowd* será possível atender adequadamente a cada demanda sem prejudicar a aplicação ou desperdiçar recursos. Neste caso, torna-se muito importante dispor de uma técnica para diferenciar o processamento *Flash Crowd* do *Spike*. Assim, para classificar essas diferentes cargas de trabalho é utilizada uma técnica de aprendizado de máquina já consolidada, o SVM (do inglês, *Support Vector Machine*). Essa técnica tem sido utilizada com bastante frequência e sucesso em várias áreas do conhecimento, inclusive em alocação de recursos em computação em nuvem [05].

A proposta desse trabalho é de criar um método para diferenciar os tipos de cargas de trabalho do tipo *Spike* ou *Flash Crowd*. Isso será realizado analisando-se atributos de

consumo de CPU e Memória através de um classificador baseado em aprendizado de máquina, o SVM.

1.1. Motivação

A motivação desse trabalho se concentra em solucionar problemas relativos à computação em nuvem, visando a escalabilidade e a elasticidade. A elasticidade é alcançada através da alocação de recursos sob demanda para os usuários da nuvem computacional. A escalabilidade é alcançada porque estes recursos podem ser demandados pelo usuário consecutivamente, dando-lhes a impressão de que os recursos são infinitos.

Em computação em nuvem, as estruturas são altamente escaláveis e provedores de infraestrutura podem facilmente expandir seus serviços para atender a rápida demanda de serviços. Esse modelo comumente chama-se *surge computing* (computação com oscilações repentinas, picos instantâneos). Neste ambiente, um dos desafios da elasticidade computacional é o correto provisionamento de recursos e serviços, pois os objetivos de um provedor de serviço são alocar e desalocar serviços da nuvem para satisfazer os acordos de nível de serviço, enquanto minimiza seu custo operacional. Entretanto, não está claro como um provedor de serviço atinge esse objetivo, não é fácil determinar o mapeamento de requisitos para ter alta agilidade e para responder as rápidas flutuações de demanda.

O principal problema não abordado pela literatura é sobre como identificar os tipos de processamento nos quais é necessário aplicar a técnica de elasticidade da nuvem, alocando mais ou menos recursos e identificando corretamente se a demanda é oriunda de uma carga de trabalho do tipo *Spike* ou *Flash Crowd*. Porém, há carência de modelos que permitam identificar tipos de processamento (*Spike* ou *Flash Crowd*) baseado no consumo de recursos operando em tempo real.

Como não há como prever o Spike e não se pode prejudicar a aplicação que se caracterizará como *Flash Crowd*, a estratégia em geral é sempre alocar os recursos demandados, o que não causa impactos importantes se ocorrer na mesma máquina física. Se o *Spike* exigir a alocação em outra máquina física (gerando um importante tráfego de rede) ou mesmo fora da nuvem privada (no domínio de nuvem pública, considerando uma arquitetura híbrida), os prejuízos podem ser muito elevados, o que se pretende evitar em um provedor de

nuvem computacional.

Os *Spikes* são comumente encontrados em aplicações interativas nas quais os usuários fazem solicitações que geram alta demanda de recursos de forma instantânea. Já em *Flash Crowd*, os usuários o fazem por um período de tempo contínuo e mais duradouro.

1.2. Objetivo Geral

Este trabalho tem como objetivo geral a criação de um método de identificação de consumo de recurso para ambiente computacional em nuvem no ambiente de infraestrutura como serviço (IaaS). Com base na coleta de métricas (atributos) de consumo de CPU e memória – em uma VM e no uso de um classificador de aprendizado de máquina, se objetiva diferenciar o processamento que leva a um *Spike* do que se caracteriza como *Flash Crowd*. Assim, visa-se tratar este provisionamento sob demanda adequadamente sem que o provedor seja impactado negativamente pelos *Spikes* e o consumidor pelo não provimento de elasticidade. Os objetivos específicos neste caso se desdobram no capítulo posterior.

1.3. Objetivos Específicos

a) Estabelecimento de métricas (atributos) para realizar a coleta de dados de consumo de CPU e memória numa VM;

b) Sintetização real de cargas de trabalhos que caracterizem comportamento do tipo *Spike* e *Flash Crowds* com base na observação de casos reais de consumo de CPU e memória;

c) Pré-processamento e rotulação de vetores de características para as classes Normais, *Spike* e *Flash Crowd*;

d) Treinamento, testes e validação do modelo obtido usando o SVM;

e) Avaliação dos resultados.

1.4. Contribuições

O presente trabalho contribui no estudo e criação de um método para a classificação de diferentes cargas de trabalhos em máquinas virtuais a fim de identificar o tipo de processamento: *Spike* ou *Flash Crowd*. Com a caracterização do comportamento da carga de trabalho, evita-se o desperdício de recursos provisionados sob demanda na nuvem.

O método criado faz uso da técnica de aprendizado de máquina (SVM) que, após gerar o modelo de processamento, consegue classificar as cargas de trabalhos, inferindo a classificação com base no modelo mesmo se aquela carga de trabalho não estava presente na base de treinamento. Nenhuma abordagem presente na literatura faz este tipo de classificação e trata o problema em tempo real para computação em nuvem.

1.5. Estrutura do Documento

O segundo capítulo deste documento apresenta a fundamentação teórica necessária para a compreensão do restante do trabalho. O terceiro capítulo contempla os trabalhos relacionados, apresentando o estado da arte relevante ao tema. O quarto capítulo apresenta a proposta deste trabalho e os resultados da sua avaliação. Por fim, no quinto capítulo há a conclusão deste estudo.

Capítulo 2

Fundamentação Teórica

Conforme [06], serviços básicos e essenciais são quase todos entregues de uma maneira simples e clara aos usuários. Serviços como água, gás, eletricidade, telefone e outros são indispensáveis para o nosso dia a dia e são comercializados por meio de um pagamento baseado no consumo/utilização. A utilização desses serviços é cobrada de acordo com a política de cada fornecedor de consumo e, ao final de determinado período, é enviada a cobrança com o valor relativo à utilização do usuário. Esses serviços são entregues em qualquer hora e em qualquer lugar. Assim, para os usuários, é possível simplesmente acender a luz, abrir a torneira, usar o fogão ou fazer uma chamada telefônica. Para a computação em nuvem (*Cloud Computing)*, a mesma ideia de utilidade está sendo aplicada, dessa maneira recursos computacionais são utilizados de forma dinâmica e utilitária.

De acordo com [01], a ideia da computação nas nuvens é a utilização de recursos computacionais em qualquer lugar independentemente da plataforma. Através da internet, utilizar as mais diversas aplicações como se estas estivessem executando de forma local em nossos computadores. Segundo [07], os recursos computacionais como rede, servidores, aplicações e serviços podem ser provisionados sob demanda e de forma rápida, liberados para utilização do usuário sem muito esforço de gerenciamento e da interação com fornecedores de serviços.

Aplicações instaladas em nossos próprios computadores ou o armazenamento de arquivos e dados locais eram comuns, estávamos acostumados. No ambiente corporativo é comum as aplicações estarem disponíveis em servidores que podem ser acessados via rede através de qualquer terminal.

A principal desvantagem deste modelo é o fato de que as aplicações e dados estão restritos a esses computadores ou servidores nessa rede. A vantagem é que seria possível, na maioria das vezes, utilizarmos as aplicações mesmo sem acesso à internet ou à rede. Devido ao crescimento e evolução constante da internet e dos meios de telecomunicações, o acesso está cada vez mais amplo e com o custo menor, criando um cenário importante para a computação nas nuvens.

Com a computação nas nuvens, os aplicativos, arquivos e dados relacionados dos usuários não precisam mais ser instalados ou armazenados de forma local, ficando disponíveis na Internet, ou seja, na "nuvem". O usuário pode apenas acessar e utiliza-los, não se preocupando com tarefas de gerenciamento da aplicação, armazenamento, manutenções, atualizações, escalabilidade, backup e outros.

2.1 Modelos de Serviços

Segundo [08], na computação em nuvem têm-se três diferentes modelos de serviços, os quais são importantes porque definem um padrão para a arquitetura, conforme exemplificado na figura 1 e explicado nas seções seguintes. A proposta do trabalho se encaixa no modelo de Infraestrutura com serviço (IaaS).

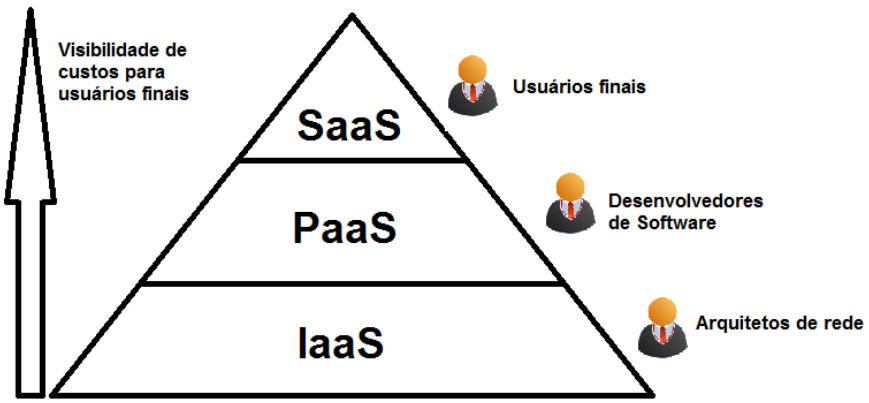


Figura 2.1– Modelo de Serviços. Adaptado de [08].

2.1.1 Infraestrutura como Serviço (IaaS)

O principal objetivo do IaaS é fazer com que o fornecimento de recursos computacionais se torne mais fácil e acessível, como servidores, rede, armazenamento e outros recursos fundamentais para construir um ambiente de aplicação sobre demanda, que podem incluir sistemas operacionais e aplicativos. No modelo de IaaS, tem-se a responsabilidade de fornecer a estrutura necessária para o Paas e o Saas. Nessa estrutura, destaca-se: suporte para adição de novos equipamentos de forma fácil; interface de administração centralizada e API para interação com os hosts; switches ou servidores são características do IaaS.

Uma estrutura de recursos computacionais baseada em técnica de virtualização define IaaS. De acordo com a necessidade da aplicação, adicionam-se mais recursos ou não, como servidores e outras de formas dinâmicas conforme a demanda necessária. Amazon EC2[09], Eucalyptus[10] e VMware[11] são exemplos de plataformas de *Cloud Computing* para IaaS.

2.1.2 Plataforma como Serviço (PaaS)

Na plataforma como serviço é disponibilizada uma infraestrutura de alto nível para implementação e desenvolvimento de aplicações criadas para computação em nuvem. A parte de rede, servidores, sistemas operacionais ou armazenamento não é controlada pelo usuário, fica de forma transparente, tem apenas controle sobre os aplicativos implantados e as configurações da aplicação hospedada. Neste modelo, são fornecidos o sistema operacional, a linguagem de programação e o ambiente necessários para o desenvolvimento, implementação e integração das aplicações.

Em geral, os ambientes são escaláveis para os desenvolvedores e devem aceitar algumas restrições sobre o tipo de software que se pode desenvolver, desde limitações que o ambiente impõe na concepção das aplicações até a utilização de bancos de dados do tipo chave-valor, ao invés de bancos de dados relacionais. Do ponto de vista do negócio, a PaaS

permitirá aos usuários utilizarem serviços de terceiros, aumentando o uso do modelo de suporte no qual os usuários se inscrevem para solicitações de serviços de TI ou de resoluções de problemas pela Web. Com isso, pode-se descentralizar certa carga de trabalho e responsabilidades nas equipes de TI nas empresas. Como exemplos de PaaS, podemos destacar o Google App Engine [12].

2.1.3 Software como Serviço (SaaS)

No modelo SaaS, os softwares são acessados e fornecidos com propósitos específicos através de navegadores de Internet. Nestes modelos, a infraestrutura subjacente, como servidores, ambientes de desenvolvimento e até mesmo características da aplicação, não são controladas ou administradas pelos usuários, exceto suas configurações específicas nas aplicações. Os softwares podem ser acessados pelos usuários de qualquer lugar e a qualquer momento através da *Web*, permitindo esta integração entre os usuários que compartilham a mesma plataforma de serviços. De forma transparente, novos recursos desenvolvidos são incorporados sem que os usuários percebam. O desenvolvedor não se preocupa com atualização da aplicação em diferentes servidores ou algo do gênero. No SaaS, consegue-se reduzir custos com aquisição de licenças, visto que essas licenças podem ser locadas juntamente com a plataforma subjacente. Como exemplos de SaaS, podem-se destacar os serviços de CRM on-line da Salesforce e o Google Docs [13].

2.2 Modelos de Implementação

Existem quatro modelos de implementação da computação em nuvem [07]. Estes modelos são diferentes baseados no negócio de cada empresa, do tipo de informação e do nível de visão desejado. Algumas empresas têm a necessidade de manter aplicativos no ambiente privado, mas podem migrar parte de outros serviços para um provedor de serviços.

Então, surge a necessidade de diferentes ambientes de computação em nuvem os quais podemos dividir em: privado, público, híbrido e comunidade.

No modelo privado, toda infraestrutura de nuvem será utilizada apenas por uma empresa ou organização, aplicando-se políticas de autenticação e autorização para acesso aos serviços. Esta nuvem pode ser administrada pela própria empresa ou por terceiros.

No modelo público, não se aplica nenhuma técnica de autenticação e autorização, sendo que toda infraestrutura pode ser acessada por qualquer usuário que conheça a localização do serviço.

No modelo híbrido, pode existir uma composição de duas ou mais nuvens, as quais podem ser privadas, públicas ou comunidades e que permitem a portabilidade de dados ou aplicativos.

No modelo comunidade, a computação nas nuvens pode ser compartilhada por um *pool* de empresas as quais compartilham dados de seu interesse. Nesta nuvem, podem-se aplicar políticas de autenticação e segurança conhecidas por todos os participantes da mesma. Esta nuvem também pode ser administrada por uma das empresas participantes da comunidade ou por uma empresa terceira.

2.3 Papéis e vantagens de utilizar a computação em nuvens

Fornecer, monitorar e gerenciar toda a infraestrutura para a solução de computação nas nuvens são papéis dos fornecedores, participando dos três níveis do modelo de serviços. O desenvolvedor utiliza os recursos de infraestrutura fornecidos e desenvolve aplicações e serviços para o usuário final na camada de Software.

Na maioria dos casos, os usuários não precisam preocupar-se com a estrutura para executar a aplicação: hardware, procedimentos de backup, controle de segurança, manutenção e outros, tudo fica a cargo do fornecedor do serviço. Existe maior facilidade no compartilhamento de dados ou trabalho colaborativo visto que todos acessam as aplicações e os dados no mesmo lugar. Também advindo de recursos de alta disponibilidade nas nuvens, se um servidor parar de funcionar ou qualquer outro dispositivo, os serviços continuam a ser providos devido à realocação em outros dispositivos.

Muitas aplicações na computação em nuvem são gratuitas e, quando for necessário pagar, os usuários somente pagam pela relação de recursos que usarem ou pelo tempo de utilização. Também quando se fala em licenciamento não será necessário pagar por uma licença de uso integral, tal como acontece no modelo tradicional. Recursos podem ser adicionados de forma dinâmica em função à determinada demanda de serviços através da escalabilidade. Vale ressaltar que esta responsabilidade de acrescer novos recursos será do fornecedor do serviço. Ao invés de comprar, instalar e operar seus próprios sistemas, por exemplo, a empresa pode deixar que o provedor da nuvem faça isso por ela. Os clientes do provedor podem pagar apenas pelos recursos que utilizarem, não precisando manter grandes conjuntos de servidores e dispositivos quando acontecerem picos de cargas. Também pode-se aproveitar a escalabilidade da nuvem fornecida pelos enormes datacenters dos provedores de nuvem.

A nuvem visa fornecer alta disponibilidade e elasticidade, sendo composta de cinco características essenciais [07]:

1: Auto-Atendimento: o consumidor utiliza os recursos conforme a sua necessidade.

2: Amplo acesso à rede: os recursos são disponibilizados na rede e acessados através de mecanismos padronizados.

3: Pool de recursos: os recursos computacionais do provedor são agrupados permitindo servir múltiplos consumidores, os recursos físicos e virtuais são distribuídos dinamicamente de acordo com a demanda.

4: Elasticidade: os recursos podem ser fornecidos rapidamente e em alguns casos automaticamente, passando para o consumidor a impressão de que a nuvem possui uma infraestrutura ilimitada.

5: Medição no uso dos serviços: a nuvem controla e otimiza o uso de recursos, fornecendo métricas de acordo com o tipo de serviço fornecido. Tanto o provedor quanto o consumidor podem monitorar e controlar a utilização dos recursos, assim o consumidor pode pagar apenas pelo o que utilizar.

2.4 Virtualização

Segundo [14], o prestador de serviços em nuvem pode compartilhar sua infraestrutura de nuvem em um datacenter para vários clientes, visto que os clientes vão dinamicamente alugar máquinas virtuais (VMs). O nível de virtualização que é oferecido depende de qual dos três modelos de serviço (SaaS, PaaS, ou IaaS) o usuário precisa. A virtualização tem como princípio a abstração de recursos computacionais utilizando máquinas virtuais, permitindo que vários sistemas operacionais sejam executados simultaneamente no mesmo servidor físico. É possível fazer um uso mais eficiente dos recursos físicos disponíveis através da migração dinâmica de máquinas virtuais.

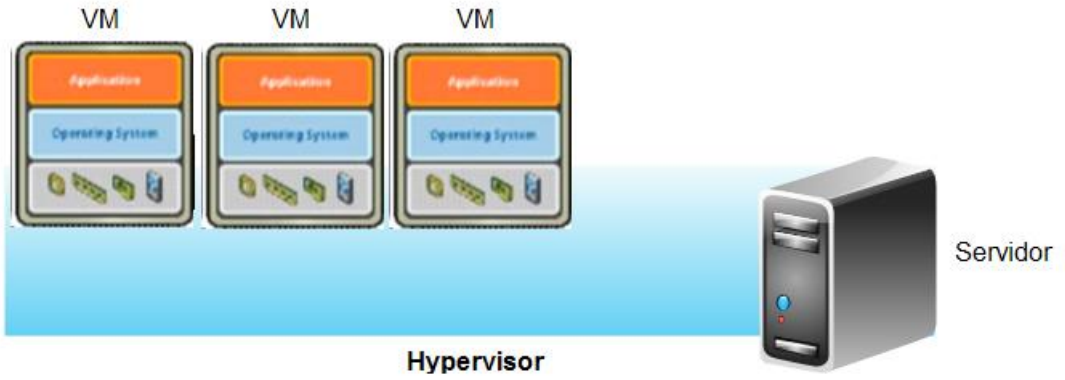


Figura 2.2– Diagrama da camada de virtualização do Hypervisor. Adaptado de [14]

2.5 Conceitos de aprendizado de máquina

Emprega-se um princípio chamado indução no aprendizado de máquina, que obtém conclusões a partir de um conjunto de exemplos. O aprendizado baseado nesse princípio pode

ser dividido em supervisionado e não supervisionado. No aprendizado supervisionado, tem-se a figura de um especialista externo, o qual apresenta o conhecimento do ambiente por conjuntos de exemplos na forma: entrada, saída desejada. No aprendizado não supervisionado não existe a presença desse especialista, não existem exemplos rotulados, ele aprende a representar as entradas através de uma medida de qualidade quando o objetivo for encontrar padrões ou tendências. As técnicas de AM são capazes de lidar com dados imperfeitos, ou seja, ruídos. [15].

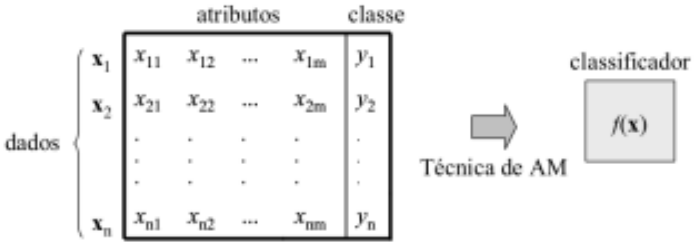


Figura 2.3 – Exemplo de vetores de atributos e classes. Adaptado de [15].

Para empregar o aprendizado de máquina necessita-se ter conhecimento de quais atributos são importantes para a classificação de dados, essa seleção de atributos pode ser feita por um especialista ou através de técnicas especificas de seleção de atributos. Na figura 2.3 exemplifica-se que se devem utilizar vetores de dados com informações dos seus atributos, ou seja, características definidas na coleta de dados, para cada vetor define-se qual classe é pertencente, após isso é realizado treinamento dessa base coletada, então se gera um modelo e partir desse modelo tenta-se classificar novas massas de dados validando assim a acurácia do modelo proposto.

Para entender melhor podemos citar um exemplo de uma fábrica que precisa separar a população de peixes que percorrem uma esteira, para isso a fábrica dispõe de uma câmera que captura fotos dos peixes, então são realizadas coletas de vários atributos como coloração, posição dos olhos, tamanho, largura, entre outros, a partir daí essas informações são reunidas a fim de classificá-las em vetores de dados, o denominado especialista determina uma massa de dados sendo de determinado peixe e outra massa de dados de outro as separando por classes, a partir dai empregam-se técnicas de aprendizado de máquina a fim de gerar um modelo que servirá para identificar e classificar novas massas de dados através de uma técnica de aprendizado de máquina.

Conforme [16], as Máquinas de Vetores de Suporte (SVMs, do inglês *Support Vector Machines*) constituem uma técnica de aprendizado que está consolidada na comunidade de Aprendizado de Máquina (AM). As SVMs são embasadas pela teoria de aprendizado estatístico[16]. Os resultados da aplicação dessa técnica são comparáveis e, muitas vezes, superiores aos obtidos por outros algoritmos de aprendizado, como as Redes Neurais Artificiais (RNAs)[16]. As SVMs têm sido utilizadas com bastante frequência e com sucesso em várias áreas do conhecimento, inclusive em alocação de recursos em computação em nuvem [05].

A partir da fundamentação teórica consegue-se dar prosseguimento para o estudo e analise de trabalhos relacionados no próximo capitulo, visando confrontá-los com a proposta deste trabalho.

Capítulo 3

Trabalhos Relacionados

Os trabalhos relacionados foram divididos em cinco sessões. Nas duas primeiras aborda-se o provisionamento de recursos e balanceamento de carga na virtualização, a terceira sessão discorre sobre o controle do provisionamento de recursos através de políticas, na quarta sessão são tratadas as abordagens de caracterização de Spike e Flash Crowds e a quinta e última sessão contém uma tabela e um resumo com a comparação entre as propostas estudadas.

3.1 Provisionamento de Recursos em Máquinas Virtuais

Os autores do artigo [17] apresentam o *SnowFlock* que propõe criar dezenas de VMs clonadas em menos de um segundo. As VMs podem executar tarefas complexas enquanto essa operação é realizada com apenas alguns segundos extras de *sobre-carga (overhead)*. Isso é relatado no artigo analisado e não tem necessidade de nenhum *hardware* especializado, a interface de programação através de APIs é simples e o ambiente proporciona escalabilidade. No artigo também são realizados *benchmarks* com alguns servidores virtuais *web*. A tecnologia utilizada na clonagem de VMs proporciona um sistema muito eficiente.

A vantagem do *SnowFlock* é a escalabilidade e a performance na clonagem de máquinas virtuais, seja através de *templates* ou VMs, criando um ambiente onde o deploy de aplicativos é extremamente rápido. Essa implementação não se preocupa com a confiabilidade dos clones e sim com o desempenho. A proposta utiliza o conceito de clone da VM via um

cluster ou uma nuvem de máquinas. O clone requer a replicação do estado da VM e o *SnowFlock* consegue isso com a paginação sob demanda. Nessa abordagem, um clone é rapidamente criado com uma pequena arquitetura de descritores da VM contendo metadados, especificações de dispositivos virtuais (discos e redes), tabelas de página e registradores de CPUs virtuais (vCPUS).

O *SnowFlock* utiliza uma técnica de migração de paginação das VMs sobre demanda para minimizar o tempo de instanciação e otimizar a utilização de recursos, diferente dos métodos convencionais de migração de VMs nos quais é feita a replicação completa do estado de paginação, que ao final gera duas VMs em execução, sendo que a replicação total coloca grandes exigências sobre a rede e resulta em tempos longos de instanciação.

Já no artigo [18], em um ambiente ideal de *Cloud*, os servidores "elásticos" aumentam e diminuem de acordo com a demanda do usuário. Atualmente um balanceador de carga ajusta o tamanho de uma máquina virtual inteira, que foi instanciada a partir do zero com base em um modelo. Infelizmente, neste mecanismo, a criação é lenta, novos servidores demoram para carregar porque essa é uma tarefa trabalhosa.

Esse artigo propõe uma nova visão chamada *Cloud Micro-elasticity*, a micro elasticidade de servidores. São realizados clones momentâneos das VMs e, para habilitar isso, é introduzido o *Color-Based Fractional VM Cloning*, uma técnica que permite o gerenciamento detalhado do estado da VM. Essa técnica, baseada em coloração de estado, permite analisar de forma mais detalhadas os estados das VMs e identificar as regiões com alta probabilidade de similaridade do conteúdo. Para avaliar o desempenho do *Color-Based Fractional VM Cloning*, foi implementado, nesse artigo, o *Kaleidoscope*, um servidor elástico, o qual reage a picos de cargas, provisionando frações de novas instâncias de VMs. Os experimentos foram realizados em cima de aplicações *Web* e OLAP. O *Kaleidoscope* instancia novos clones completos em segundos e quase corresponde à idealizada estratégia de clonagem que tem latência zero na replicação, bem como é efetivo em encontrar o estado necessário para o novo clone. As frações são incrementadas com apenas o que precisam para novas atribuições ou mantêm o estado recém-transferido. A simulação mostrou que o manuseamento refinado do estado da VM reduziu drasticamente o uso de infraestrutura, um servidor pode escalar rapidamente e precisa muito menos capacidade para lidar com o aumento de demandas.

Necessidade da elasticidade: A análise mostra que durante o mês inteiro uma camada

média de cliente demanda 15.3% do seu pico. Isso indica flutuações de longo tempo, entretanto tem demanda de elasticidade em pequenas escalas de tempo.

A micro elasticidade é alcançada por realizar o clone ao vivo da VM e aumentando a capacidade através de duas diferentes técnicas, realizando a coloração do estado da VM para prover a propagação e o compartilhamento, além da alocação fracionada de VM para minimizar a captação completa do estado da VM. Através dessas combinações, é habilitada a instanciação de frações de VM de forma rápida. Para alcançar os benefícios da propagação sob demanda (instanciação rápida de VM com curto período de preparação e alocação de recursos proporcional ao uso), sem suas respectivas deficiências, usam-se dois novos mecanismos que otimizam o desempenho da clonagem da VM: coloração de estado da VM, que discrimina o outro estado da VM em regiões semanticamente relacionadas, permitindo que o estado seja eficientemente adequado e compartilhado; e a alocação fracionária, que aloca dinamicamente memória para acomodar somente o estado que é realmente acessado.

3.2 Balanceamento de Carga na Virtualização

No artigo [19], são lançados alguns desafios centrais na construção de escalabilidade de uma nuvem. Foram baseados nos últimos anos de pesquisa e entrega de produtos para gerenciamento de recursos em clusters. São discutidas várias técnicas para lidar com esses desafios com os prós e contras de cada técnica. Também são discutidas três abordagens para desenhar um ambiente escalável. Há muitas maneiras de fornecer controles de gerenciamento de recursos em um ambiente de nuvem.

Nesse artigo, foi escolhido o Vmware Distributed Resource Scheduler (DRS) como um exemplo porque tem um rico conjunto de controles que fornecem os serviços necessários para a gestão de multi recursos ao fornecer qualidade de serviço diferenciada para grupos de máquinas virtuais, embora em uma escala pequena em comparação com implementações de nuvens típicas. Os controles básicos do gerenciamento de recursos na nuvem são baseados em: Reserva – a qual especifica um número mínimo de recursos garantidos; Limite – uma prevenção que o consumo da VM não passe aquele limite; Compartilhamentos – é garantida uma fração mínima para o compartilhamento entre os sistemas, caso isso não esteja em uso.

No Resource Pools (Grupos de recursos), os administradores e os usuários podem

especificar políticas de gestão flexíveis através da introdução do conceito que pode ser usado para especificar uma atribuição de recursos agregados para um conjunto de VMs. Um pool de recursos é um objeto nomeado com as configurações de associados para cada recurso gerenciado. As mesmas reservas, compartilhamento e limites usados para as VMs.

O DRS é projetado para aplicar políticas de gestão de recursos com precisão, oferecendo recursos físicos para cada VM com base no seu direito calculado pelo DRS. O DRS visa aproximar este comportamento através de um conjunto de hosts distribuídos, proporcionando a ilusão de que todo o cluster é um único e enorme "super-host" com a capacidade global de todos os hosts individuais.

Este artigo procura demonstrar os prós e contras de três técnicas de escalabilidade utilizadas no DRS. Escalabilidade hierárquica: nesta abordagem, os sistemas de gerenciamento de recursos são construídos uns em cima dos outros a fim de atingir a escala necessária. Escalabilidade plana: nesta abordagem, uma camada de gerenciamento completamente distribuída e descentralizada de recursos é construída, criando um único domínio de gestão de recursos. As decisões são feitas usando dados coletados e agregados sobre o grande número de hosts e máquinas virtuais. Escalabilidade estática: nesta abordagem, o gerenciamento de recursos em larga escala é obtido fazendo operações inteligentes em pequena escala. Este resultado tem sido amplamente utilizado em muitas outras áreas para balanceamento de carga.

3.3 Controle e Provisionamento de Recursos por políticas

No artigo [20] procura-se obter um custo otimizado para fluxo de trabalhos na elasticidade em Computação em Nuvem. Como observado em IaaS, a tecnologia de provisionamento de recursos torna o processo de alocação de recursos simples e direto. Os usuários só precisam identificar o tipo de recurso, período de locação e custo para as suas aplicações. Os tipos de recursos tendem a depender de características da aplicação, as principais preocupações dos usuários seriam o período de locação e o custo. Os usuários querem executar seus aplicativos mais rápido e com o mínimo custo. No entanto, se os aplicativos não são de missão crítica (Real Time), os usuários podem tolerar um pouco de atraso de execução para uma maior redução de custos.

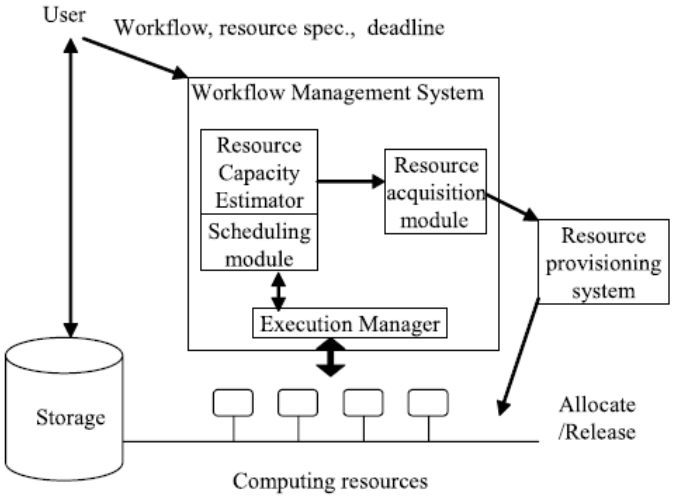


Figura 3.1– Fluxos de trabalhos para recursos elásticos. Adaptado de [20]

A figura 3.1 acima retrata o modelo de plataforma de computação para a execução de fluxos de trabalho sobre os recursos elásticos em um cenário simples de trabalho. Os recursos são elásticos com um período de locação discreta. Um usuário pede ao Sistema de Gerenciamento de Workflow (WMS) para executar um fluxo de trabalho dentro de um prazo para uma especificação de recurso necessário.

Alguns problemas foram endereçados para a elasticidade baseada em Workflow. Para isso, foi criado o algoritimo BTS (*Balanced Time Scheduling*). BTS estima o número mínimo de recursos computacionais necessários para executar um fluxo de trabalho dentro de um prazo determinado. Com isso, este trabalho propõe um algoritmo para melhorar algumas características do anteriormente citado BTS, chamando agora de PBTS (*Partitioned Balanced Time Scheduling*), que determina o melhor número de recursos de computação por unidade de tempo de carga em ambientes de computação elástica, minimizando o custo bruto durante a vida útil de todo aplicativo.

Fundamentalmente, o algoritmo de PBTS é uma extensão do algoritmo de BTS para recursos elásticos e herda todos os benefícios. Primeiro, o PBTS é escalável para fluxos de trabalho muito grandes com dezenas de milhares de tarefas e arestas. Em segundo lugar, ele pode lidar com fluxos de trabalho com tarefas paralelas, cujas subtarefas são executadas simultaneamente em recursos distintos. Além das características herdadas, PBTS faz um uso pleno da elasticidade dos recursos e fornece ao usuário um plano de alocação de recursos com o menor custo. PBTS é projetado para ajustar a alocação de recursos em tempo de execução

para que possa encontrar a capacidade de recursos com menor custo, quando as tarefas são concluídas antes do tempo previsto conforme especificado na Figura 3.2.

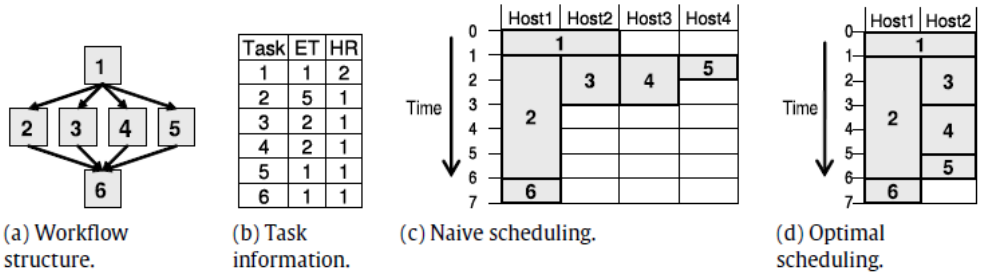


Figura 3.2- Comparação de estratégias de agendamento para minimizar desperdício em um fluxo de trabalho de exemplo. Adaptado de [20]

Neste trabalho, foi proposta uma arquitetura para a execução de fluxos de trabalho em recursos de computação elástica. Também foi proposto o algoritmo chamado PBTS para estimar a capacidade de recursos mínimos para executar um fluxo de trabalho dentro de um prazo determinado. O PBTS estima a capacidade de recursos por tempo de partição, de modo que minimiza o custo do recurso, satisfazendo os prazos. Além disso, controla todos os fluxos de trabalho e determina não só a capacidade de recursos, mas também o calendário e as exigências das tarefas.

No artigo [04], os autores procuram resolver problemas dos modelos de desempenho irrealistas nos quais usualmente empregam-se modelos lineares e filas, que não estão à altura de controlar aplicações complexas na Internet sem comprometer os SLAs. Eles acreditam que esses problemas podem ser resolvidos usando uma suíte de modelagem, controles e técnicas de análise baseados em *Statistical Learning Machine* (SLM). É proposto um *framework* com três componentes.

Primeiro: modelos robustos de desempenho os quais permitam prever o futuro da configuração e das cargas de trabalho.

Segundo: reduzir o consumo de recursos, implantando um simulador de políticas de controle (*control policy simulator*), o qual compara através dos modelos de desempenho diferentes políticas para adicionar ou realocar recursos.

Terceiro: para robustez são implantadas técnicas de gerenciamento de modelos originadas da literatura SML, como: treinamento on-line e detecção de pontos de mudança, para ajustar os modelos quando as mudanças são observadas na execução da aplicação.

Importante ressaltar que o controle e a gestão são modelos agnósticos, em que são utilizados modelos genéricos que permitem o "plug and play" de uma ampla variedade de modelos de desempenho.

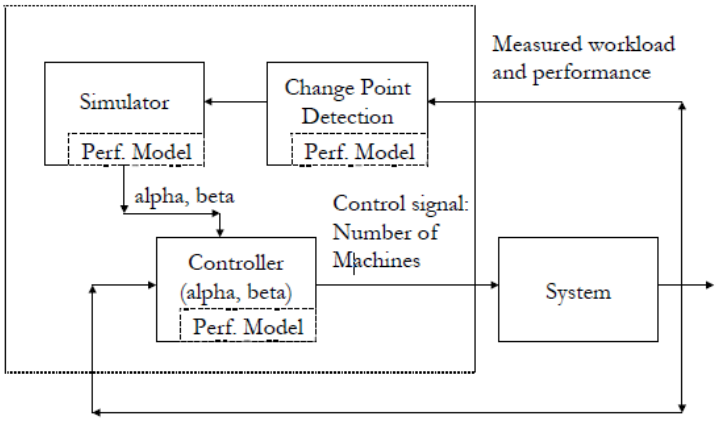


Figura 3.3– Diagrama de arquitetura do framework de controle. Adaptado de [04]

3.4 Caracterizações de Spikes e Flash Crowds

Obter cargas de trabalho com picos significativos requer traços de cargas de trabalho realistas que são difíceis de obter quando se fala de elasticidade dos serviços de Internet. Criar um modelo de trabalho e gerar carga sintética é uma abordagem popular, entretanto não existe uma caracterizacão e modelos de *Spikes*. Neste artigo foram analisadas cinco cargas de trabalho de *Spike* nas quais os picos variam significativamente em muitos aspectos importantes, tais como declividade, magnitude, duração e localização espacial. O artigo propõe e valida um modelo de *Spikes* que permite sintetizar volumes e picos de dados, podendo, assim, ser utilizado por usuários de computação em nuvem ou por provedores para testar sua infraestrutura.

O objetivo do artigo [21] é criar um modelo não para prever picos, mas modelar as mudanças no volume de carga de trabalho e popularidade de objetos individualmente. A ideia não é criar um modelo complexo que imite cada detalhe dos traços reais, mas um modelo estatisticamente simples que possa ser experimentalmente prático e capture características importantes. Na comparação com terremotos, embora não sejamos capazes de prever terremotos individuais, é útil para caracterizar as propriedades gerais destes, de modo a obter

uma previsão adequada.

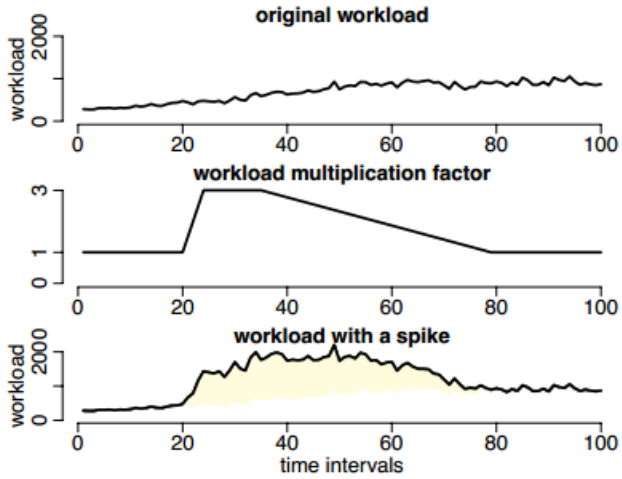


Figura 3.4– Perfil de Carga de trabalho de Spike. Adaptado de [21]

A figura 3.4 acima, na parte superior, exemplifica um caso de um perfil normal de carga de trabalho, sem *Spike*. O gráfico central atua como um fator de multiplicação da carga de trabalho com parâmetros de tempo e magnitude que definem o perfil da carga de trabalho e também seu relativo aumento de volume. No terceiro gráfico, o perfil de carga de trabalho durante o *Spike* é obtido como uma multiplicação do perfil normal (superior) e (centro).

As redes sociais e os fóruns de discussão têm se tornado um lugar onde as pessoas se comunicam e expressam sua opinião livremente. Os usuários frequentemente apontam para links externos para fundamentar as suas discussões usando aplicativos como o Digg. A carga de tráfego pesado súbita imposta aos websites externos vinculados faz com que eles fiquem sem resposta que conduz ao efeito *Flash Crowd* na infraestrutura desses sites.

No artigo [22], mostra-se que uma grande parte dos sites que tornam-se populares através de histórias em fóruns de discussão públicos, sofrem com o fenômeno *Flash Crowd*. Esses sites, conforme se tornam mais populares, apresentam uma latência e variação de tempo de resposta cada vez maior. Para apoiar a hipótese deste artigo são medidos periodicamente e ao longo de um extenso período os tempos de download para todos os URLs externos que foram submetidos a um fórum de discussão social. A ideia do artigo é prever picos repentinos de tráfico de rede antes do tempo.

Para realizar a previsão de carga de trabalho, o artigo começa a experiência no

momento em que a história foi submetida. Foram capturados os dados de teste em diferentes períodos depois que a história foi submetida. Os dados de treino foram capturados dentro de todo o tempo (que foi de 120 horas), enquanto que os dados de teste foram capturados em 5, 10, 15, 30, 60, 120, 300, 600, 900 minutos e o tempo completo (que é de 120 horas) desde que a história foi submetida. Além disso, para a previsão, foram utilizados recursos relacionados a estatísticas, feedback dos usuários e estrutura da comunidade, além da adesão.

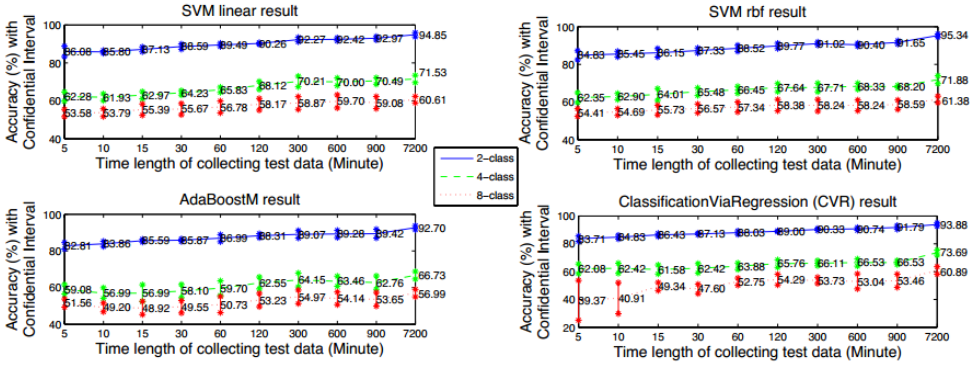


Figura 3.5 – Taxa de acerto dos quatro melhores classificadores testados. Adaptado de [22].

Em resumo, o trabalho analisa o tráfico de rede dos websites e procura encontrar o melhor classificador de dados para prever novas situações em que ocorram flutuações de demanda de tráfico de rede. Para isso, foram testados sete algoritmos de classificação e conforme figura 3.7 foi comprovado que o SVM teve o melhor resultado de classificação do tráfego de rede.

3.5 Considerações

O Quadro abaixo relaciona os principais aspectos de cada abordagem dos artigos analisados para efeito comparativo entre os mesmos:

Artigo/Sessão	**Proposta**	**Comparação com o trabalho**	**Modelo**
SnowFlock: Virtual Machine	Solução para rápida clonagem de VMs, instanciação rápida de máquinas virtuais. Visa à	Artigo preocupa-se com a instanciação rápida de VMs caso tenha demanda, replicação parcial e	*Cloud* (IaaS)

***Cloning as a First-Class Cloud Primitive* [17].**	escalabilidade e ao desempenho da clonagem (replicação parcial do estado completo da VM). Não se preocupa com a confiabilidade dos clones.	rápida da VM através da semântica *fork*. Não exemplifica modelos de carga de trabalho, trata sobre tipos de elasticidade e custos de *overhead*.	
***Kaleidoscope: Cloud Micro-Elasticity via VM State Coloring* [18].**	Micro elasticidade via coloração do estado da máquina virtual, replicação e instanciação rápida através da clonagem apenas do que está sendo usado no momento.	Artigo também trata sobre uma forma de economizar custos com os *overheads* na elasticidade da VMs através do método de transferência de estado, *State Coloring*.	*Cloud* (IaaS)
Cloud Scale Resource Management: Challenges and Techniques [19].	Trata do balanceamento de carga para o ambiente virtual no *VMware*, mostrando como é feito a limitação, reservas e compartilhamento de recursos para provisionamento de novas VMs. O Algoritmo tem como base de cálculo total, a soma de recursos de todos os *hosts*.	Implementa um algoritmo no qual é possível checar o consumo das VMs e provisionar mais recursos através da movimentação de instância com base no cálculo da soma de recursos de todos os hosts através da tecnologia *VMware* DRS (*Distributed Resource Scheduler*).	*Cloud* (IaaS)
Cost optimized provisioning of elastic resources for application workflows [20].	Algoritmo que estima os recursos necessários para execução de um fluxo de trabalho em um determinado período de tempo, melhoria do algoritmo BTS , o qual é chamado de PBTS, que consegue tratar grandes fluxos de trabalhos e determina o melhor número de recursos de computação por unidade de tempo de carga em ambientes de computação elástica, minimizando o custo bruto durante a vida útil de todo o aplicativo.	Algoritmo estima a necessidade de recursos para um fluxo de trabalho, consegue estimar e aperfeiçoar os recursos provisionados na nuvem, não diferencia os tipos de cargas de trabalhos. A proposta é melhorar o algoritmo BTS para o ambiente de elasticidade da nuvem.	*Cloud* (IaaS)
Statistical Machine Learning Makes Automatic Control Practical for	Técnica de aprendizado de máquina para prever cargas de trabalho utilizando SML (*Statistical Learning Machine*), implanta um simulador de controle de políticas para ajustar os modelos gerados quando mudanças são produzidas	Criar um *framework*, uma suíte de modelagem a fim de: Definir modelos de desempenho para prever o futuro, um controle para comparar políticas definidas com os modelos de desempenho, a fim de consumir menos recursos e, por fim, é definido	*Cloud* (SaaS)

Internet Datacenters [04].	na aplicação.	um gerenciamento para previsão e controle de mudanças. Artigo não faz menção a tipos de cargas de trabalho e como diferenciá-las.	
Characterizing, Modeling, and Generating Workload Spikes for Stateful Services [21].	Neste artigo foram analisadas cinco cargas de trabalho de Spike nas quais os picos variam significativamente em muitos aspectos importantes, tais como declividade, magnitude, duração e localização espacial. O artigo propõe e valida um modelo de *spikes* que permite sintetizar volumes e picos de dados podendo, assim, ser utilizado por ambos os usuários de computação em nuvem ou provedores para testar sua infraestrutura.	Não trata atributos relacionados a consumo de recursos de máquina como CPU ou memória, cria um modelo e simulador de picos relacionado com modelos de carga de trabalho de picos em serviços na Web de fatos que aconteceram no passado como Copa do Mundo, morte do Michael Jackson, entre outros, também trabalha apenas na camada de SaaS.	*Cloud* (SaaS)
Predicting Network Response Times Using Social Information [22].	O trabalho analisa o tráfico de rede dos websites e procura encontrar o melhor classificador de dados para prever novas situações em que ocorra flutuações de demanda de tráfico de rede, no referido caso, *Flash Crowds*.	Consegue-se identificar que o SVM foi o melhor classificador em uma análise de cargas de trabalho de tráfego de rede. Artigo trata apenas sobre tráfego de rede originado de *websites*, não aborda cargas de trabalho para VMs utilizando métricas como CPU ou memória.	*Cloud* (SaaS)

Para concluir a análise realizada nesse capitulo os artigos [17] e [18] tratam apenas sobre formas de elasticidade e gerenciamento de recursos no ambiente de elasticidade em nuvem. No artigo [19], foi demonstrado como o VMware faz balanceamento de carga através da tecnologia de DRS. No [20], é criado um algoritmo para estimar recursos necessários através de um modelo de desempenho; no [04] cria-se apenas uma suíte de modelagem com um ciclo para prever, controlar e gerenciar os recursos computacionais alocados e desalocados, e em ambos artigos [21] e [22] são caracterizados *Spikes* e *Flash Crowds*,

criados modelos e simulações apenas em serviços na Web e tratando apenas de tráfego de rede.

Nenhum dos artigos analisados tratou sobre a classificação de dados de diferentes cargas de trabalho a fim de identificar se são do tipo *Spike* ou *Flash Crowd*, bem como nenhum deles considera aspectos como a utilização de recursos como CPU e memória em uma máquina virtual. Portanto, neste trabalho será abordado um método para diferenciar cargas de trabalhos de *Spikes* e *Flash Crowds* através de uma simulação e classificação de dados aplicando a técnica de *Support Vector Machines* (SVM).

Capítulo 4

Proposta

A proposta deste trabalho é de criar um método para diferenciar tipos de cargas de trabalho classificando-as como *Spike* ou *Flash Crowd*. Para isso, foram realizadas definições e coletas de métricas de consumo de CPU e memória a fim de realizar simulações de cargas de trabalho e gerar vetores de características para treinamento e classificação utilizando as técnicas de SVM.

A abordagem adotada se baseia numa estratégia de alocação local de VMs, que após uma avaliação do perfil de uso da VM, evita que os recursos locais esgotem, migrando as VM que demandam processamento mais duradouro para um domínio remoto (Figura 4.1).

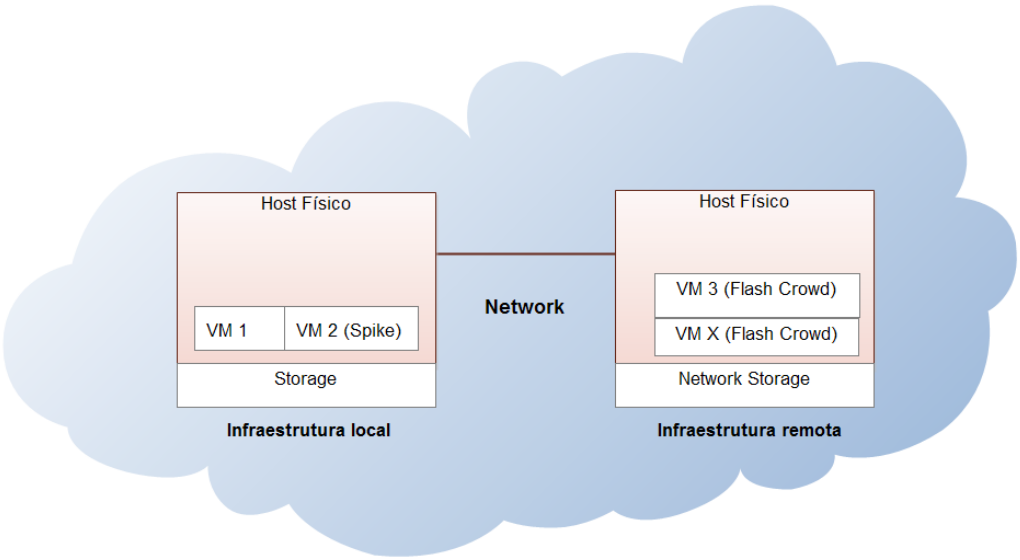


Figura 4.1– Visão geral do esquema incremental de escalabilidade e elasticidade da proposta.

Assim, pressupõe-se que todo o pedido de alocação de recursos derivado do consumo de CPU (baseado num limiar definido pelo administrador especialista do data center) automaticamente alocará uma VM na mesma máquina física. A razão para tal é a constatação que em média o tempo gasto para instanciação de uma VM na mesma máquina física é baixo, principalmente se compartilhado o mesmo *Storage*. Assim, foi assumido também que toda nova instanciação devido ao consumo de CPU disponível é um *Spike*.

Após o consumo da VM instanciada acontecer, um conjunto de atributos serão monitorados e uma técnica de aprendizado de máquina será utilizada para avaliar se o consumo desta VM pode ser classificado como um *Flash Crowd*. Em caso afirmativo, a VM será migrada para uma máquina que comporte uma VM com características para suportar o processamento mais pesado de carga de trabalho. Observe que esta estratégia é considerada porque a migração é mais custosa, pois a VM será transportada pela rede, podendo estar dentro do domínio do mesmo provedor ou em domínio público, mas, neste caso, o *storage* não estará anexado à máquina física, então, além do custo de migração da VM pela rede o mapeamento do sistema de arquivos também será via unidade de armazenamento de rede (*networked storage*).

Uma das principais dificuldades para fazer a diferenciação entre processamento derivado de instanciação é sua classificação como *Spike* ou *Flash Crowd*, dado que ambos consomem todos os recursos da CPU. Apesar de haver diferenciação no tempo de duração do consumo, a questão é onde está a fronteira entre o *Spike* e *Flash Crowd*.

Para auxiliar nesta tarefa a fim de encontrar a fronteira entre *Spike* e *Flash Crowd* foram definidos vários atributos que serão apresentados na próxima seção. Então, aplicando um classificador baseado em aprendizado de máquina, objetiva-se obter o modelo que representa o *Spike*, fez-se o mesmo para obter o modelo que representa *Flash Crowd*. Desta forma, conhecendo os modelos que claramente representavam *Spike* e *Flash Crowd*, foi gerada uma carga de trabalho que aproximava os dois modelos. Assim, chegou-se ao tempo de duração do processamento em que o classificador não conseguia mais distingui-los com acurácia. Para os propósitos dos métodos, consideramos este ponto a fronteira que distingue os dois tipos de processamento.

A seguir, serão apresentados os detalhes da classificação de processamento do tipo *Spike* e *Flash Crowd*, que nos permitiu concluir que a abordagem de incremento de

elasticidade e escalabilidade são promissoras para o método proposto.

4.1 Definição do ambiente e atributos de utilização de recursos da VM

Para definir os atributos que nos permitiriam caracterizar o *Spike* e *Flash Crowd*, vários testes baseados em medições de atributos reais de carga simulada foram obtidos. A seguir, serão relatadas as ferramentas e os ambientes resultantes destes testes que permitiram chegar ao resultado desejado.

O *System Monitor* ou *System Guard*, utilizado pela interface gráfica chamada KDE[22] do sistema operacional GNU/LINUX, foi utilizado para monitorar as máquinas locais. A coleta foi realizada dentro da máquina virtual (VM). Através do uso de funções avançadas do navegador de sensores da interface gráfica do *system monitor*, os atributos de interesse foram salvos em um arquivo.

Ao todo, foram utilizados vinte e três atributos coletados em um arquivo texto gerado pelo *System Monitor*. Os registros com os atributos foram coletados em intervalos de um segundo e, ao todo, foram coletados 800 registros para cada carga de trabalho sintetizada. Na Tabela 4.1, são relacionados os atributos coletados e uma breve descrição sobre cada um.

Tabela 4.1– Tabela com o nome das métricas coletadas e sua descrição.

Métrica Coletada	**Descrição**
CpuSystemTotalLoad	Sistema de CPUs – Carga total (1)
CpuSystemLoadSys	Sistema de CPUs – Carga de Sistema (2)
CpuSystemLoadAvg	Sistema de CPUs – Carga média (3)
CpuSystemUserLoad	Sistema de CPUs – Carga de usuário (4)
CpuSystemWaiting	Sistema de CPUs – Carga em espera (5)
CpuPagesIn	CPUs – Páginas Lidas (6)
CpuInterruptTotal	CPUs – Total de interrupções (7)
CpuContextSwitch	CPUs – Troca de contexto (8)
CpuCore01TotalLoad	CPU Núcleo 01 – Carga total (9)
CpuCore01SystemLoad	CPU Núcleo 01 – Carga de sistema (10)
CpuCore01UserLoad	CPU Núcleo 01 – Carga de usuário (11)
CpuCore02TotalLoad	CPU Núcleo 02 – Carga total (12)
CpuCore2SystemLoad	CPU Núcleo 02 – Carga de sistema (13)
CpuCore02UserLoad	CPU Núcleo 02 – Carga de usuário (14)

MemPhyUsed	Memória física utilizada (15)
MemPhyApplication	Memória física utilizada pela aplicação (16)
MemPhyCached	Memória física usada em caches (17)
MemPhyBuff	Memória física usada em buffers (18)
MemSwapUsed	Memória Swap utilizada (19)
DiskChangeTotalAcc	Total de acessos nos discos (20)
DiskRateRotalAcc	Taxa de acesso total aos discos (21)
DiskRateReadAcc	Taxa de leitura total aos discos (22)
DiskRateWriteAcc	Taxa de gravação total aos discos (23)

Um fragmento do log gerado pelo *System Monitor*, no arquivo texto de saída para o atributo carga total de CPU (*TotalLoad*), é mostrado na Figura 4.2.

```
jul 22 11:27:35 localhost cpu/system/TotalLoad: 100
jul 22 11:27:36 localhost cpu/system/TotalLoad: 32.5
jul 22 11:27:37 localhost cpu/system/TotalLoad: 21.0526
jul 22 11:27:38 localhost cpu/system/TotalLoad: 28.2051
jul 22 11:27:39 localhost cpu/system/TotalLoad: 23.0769
jul 22 11:27:40 localhost cpu/system/TotalLoad: 28.2051
jul 22 11:27:41 localhost cpu/system/TotalLoad: 22.5
```

Figura 4.2– Log para coleta do atributo *TotalLoadCPU* gerado em arquivo texto pelo *System Monitor*

4.2 Sintetização de cargas de trabalho

Para sintetizar as cargas de trabalho que permitiram coletar os atributos referentes a *Spike* e *Flash Crowd*, foi desenvolvido um programa (código) mostrado na figura 4.3, que consumia memória e CPU, de modo que o tempo de processamento pudesse ser controlado. Na estratégia de alocação de memória foi considerado *Lazy Allocation*, pois o programa não é capaz de alocar memória real, apenas virtual, mas para a coleta de dados foi suficiente pois nos gráficos de consumo conseguimos identificar os picos de memória mesmo que sejam apenas picos alocados, para efeito do trabalho exposto caracterizam-se próximos a picos reais.

Cada coleta teve duração média de dez minutos. Com esse simulador, foi possível criar picos com duração de dois segundos até a última coleta com picos de vinte e seis

segundos de duração, sendo incrementados a cada dois segundos. Também, através de programação *Shell Script*, foi agendado o tempo de intervalos que esse programa iria executar, demarcando um intervalo entres os picos gerados que podem ser visualizados nos gráficos das telas de consumo, começando em vinte e seis segundos de intervalo no primeiro teste até dois segundos no último teste, esse intervalo significa que o consumo de recursos retorna à próximo de zero por cento de consumo.

```
GERADOR DE CARGA DE TRABALHO
int main(int argc, char const *argv[])
{
time_t t,tempoStop; //inicializa variáveis
  // pega o número de segundos
t = time(NULL);
  //atribui mais 3 segundos a variável tempoStop
tempoStop =  t+3;  // Segundos alterados em cada carga de trabalho até chegar a picos de 28
segundos na ultima simulação, acréscimo de 2 a 3 segundo cada teste
while(t!=tempoStop){  // faz um laço comparando t com tempoStop
        t = time(NULL);        // atualizando t para segundos atuais
                printf ("carga"); // alocando cpu (loop printf)
                int *a = (int *) malloc (100*sizeof(int)); //alocando memória
                a = realloc (a,10000);
        printf("%ld e %ld\n",t,tempoStop);
        }
}
```

Figura 4.3- Código do programa que gera cargas de trabalho

A figura 4.4 mostra uma carga de trabalho obtida executando aplicações LAMP (Linux, Apache, MySQL e PHP), considerando normal em uma máquina virtual dentro de um provedor de nuvem. Pode-se notar que o consumo médio da CPU foi de 40% (variando de 20% a 60%). Em tal caso, pode-se observar que não existe um padrão específico de processamento de CPU, porque a demanda de processamento está fora do nosso controle.

Figura 4.4– Gráfico de consumo carga de trabalho (Normal).

Através do algoritmo da figura 4.3 começamos a controlar a demanda de processamento de forma manual dentro de uma máquina virtual, a seguir foram registradas através de figuras cópia da tela do monitor do sistema a cada carga de trabalho controlada, demonstrando visualmente o consumo de recursos nas figuras de 4.5 até 4.17. No total, foram realizadas 13 cargas de trabalho sintéticas. As figuras 4.5 até 4.12 mostram variações que o especialista do data center analisou e considerou Spike; as figuras 4.13 até 4.17 representam o que o especialista considerou *Flash Crowd*. A informação referente ao perfil de processamento *Spike* ou *Flash Crowd* foi utilizada para rotular os vetores de características coletadas para o treinamento do SVM. A carga de trabalho inicial teve surtos de processamento que duraram 2 segundos e o intervalo entre os surtos foi de 26 segundos, por exemplo, na figura 4.6 o pico de processamento durou 4 segundos e o intervalo entre os picos foi 24 segundos. A cada carga de trabalho foram adicionados 2 segundos de surto e diminuídos 2 segundos de intervalo entre os picos até chegarmos na figura 4.17 com 26 segundos de surto e 2 segundos de intervalo.

Figura 4.5- Gráfico de consumo carga de trabalho (Spike 01).

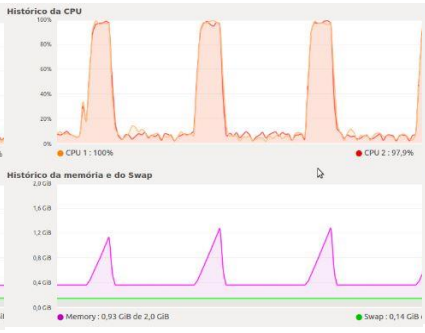


Figura 4.6- Gráfico de consumo carga de trabalho (Spike 02).

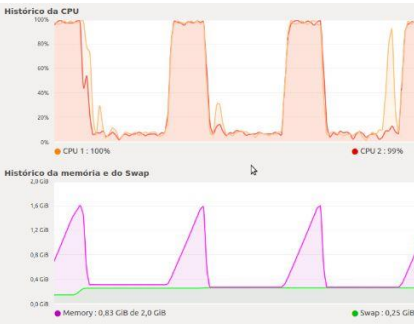
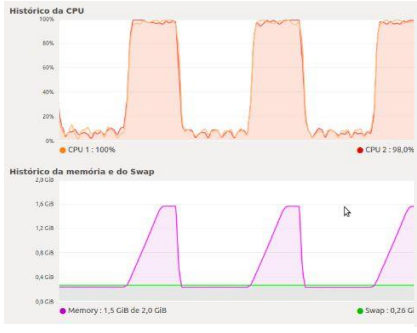
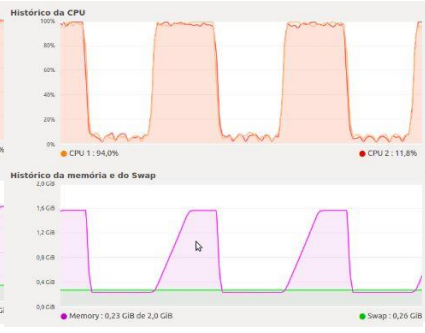


Figura 4.7- Gráfico de consumo carga de trabalho (Spike 03)





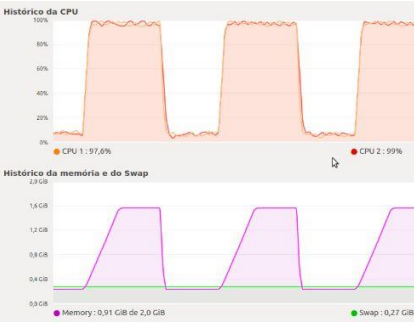


Figura 4.8 - Gráfico de consumo carga de trabalho (Spike 04).

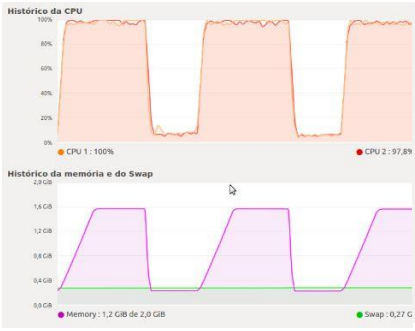


Figura 4.9 - Gráfico de consumo carga de trabalho (Spike 05).

Figura 4.10 - Gráfico de consumo carga de trabalho (Spike 06)

Figura 4.11- Gráfico de consumo carga de trabalho (Spike 07)

Figura 4.12- Gráfico de consumo carga de trabalho (Spike 08)

Figura 4.13-Gráfico de consumo carga de trabalho (Flash 09)

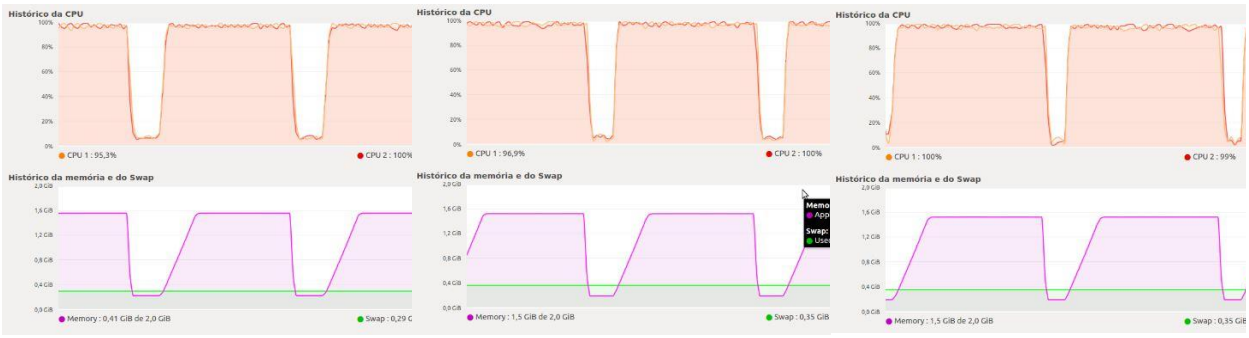


Figura 4.14- Gráfico de consumo carga de trabalho (Flash 10)

Figura 4.15-Gráfico de consumo carga de trabalho (Flash 11)

Figura 4.16-Gráfico de consumo carga de trabalho (Flash 12)

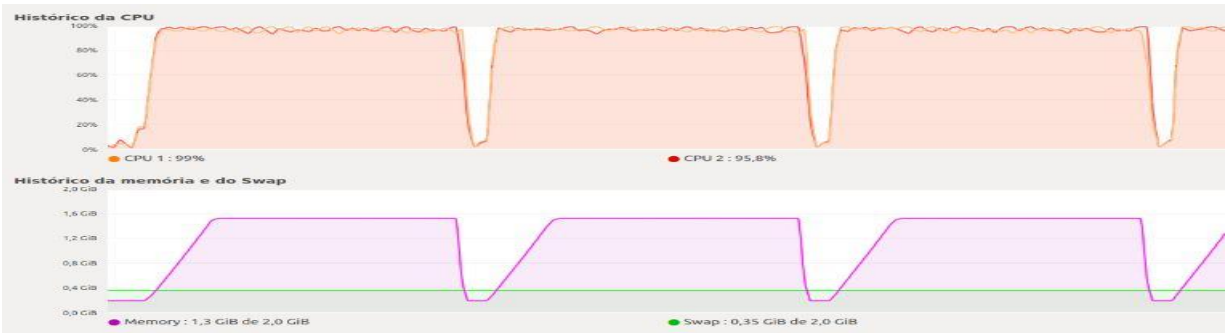


Figura 4.17- Gráfico de consumo carga de trabalho (Flash 13)

4.3 Aplicando aprendizado de máquina

Foi desenvolvido um script para fazer um pré-processamento no arquivo de log gerado pelo *System Monitor* para eliminar informações não relevantes e para gerar os vetores de características (figura 4.18).

```
0 1: 18.1818 2: 10 3: 1.27 4: 25.9259 5: 0 6: 0 7: 2050.62 8:
2747.1 9: 0 10: 0 11: 0 12: 81.8182 13: 0 14: 30.7692 15:
2.03603e+06 16: 1.00913e+06 17: 873676 18: 152080 19: 0 20: 0
21: 0 22: 0 23: 0
1 1: 71.0526 2: 19.5122 3: 1.58 4: 76.9231 5: 0 6: 0 7:
1134.22 8: 38481.1 9: 100 10: 0 11: 66.6667 12: 0 13: 0 14:
92.3077 15: 1.07724e+06 16: 1.2555e+06 17: 445008 18: 84136
19: 0 20: 0 21: 0 22: 0 23: 0
2 1: 100 2: 25 3: 2.19 4: 76.25 5: 0 6: 0 7: 830.044 8:
19300.6 9: 100 10: 17.3913 11: 77.7778 12: 100 13: 0 14: 0
15: 2.55831e+06 16: 1.99384e+06 17: 476340 18: 88248 19: 0
20: 0 21: 0 22: 0 23: 0
```

Figura 4.18– Fragmento dos arquivos com vetores de características

Neste caso, o tipo de aprendizado é o supervisionado, com o arquivo de vetores de características, convertido para formato do libSVM (biblioteca do SVM). Isto significa que, no início da linha vai o rótulo zero (0) para carga de trabalho considerada normal, um (1) para carga de trabalho do tipo *Spike* e dois (2) para carga do trabalho do tipo *Flash Crowd*. Essa rotulagem foi realizada por um especialista baseado nos gráficos de consumo e no tempo de resposta da VM.

Na figura 4.18, os atributos (Tabela 4.1) são identificados por números (de 1 a 23) que correspondem à numeração que aparece na descrição da Tabela 4.1 e pelo seu respectivo valor (sucedendo o sinal de ':'), e.g. '7: 1134.22,' significa que o atributo 'Interrupção total de CPUs' tem o valor 1134.22 para o caso de processamento com perfil *Spike* (1). Na figura 4.18, há outros exemplos para os demais atributos e tipos de perfil de processamento.

Após a geração dos vetores de características para os treze arquivos (figuras 4.5 a 4.17), que representam respectivamente perfil de processamento (*spike* e *flash crowd*) e os registros rotulados, como mostrados nas respectivas figuras, foram iniciados os testes usando a técnica de aprendizado de máquina, SVM. A hipótese era que se fosse feito o treinamento e teste com algo próximo de 100% para as várias combinações de arquivos Spike e Flash, se comprovaria que a rotulação feita pelo especialista estava correta. Caso contrário, as rotulações deveriam ser mudadas, significando que o que fora classificado como *Spike* poderia ser *Flash Crowd* ou vice-versa. Isso porque o conjunto de parâmetros havia sido testado e apresentava bons resultados. Existia, então, a dificuldade de escolher as combinações de arquivos, dado que não seria possível testar todas as combinações dos treze arquivos entre si. Foram feitos inicialmente os seis primeiros testes da tabela 4.2, depois deste, conclui-se que era melhor inspecionar melhor as possibilidades. Então, foram feitos os testes de 7 a 12 e, por último, foram feitos mais dois testes resultando em 14 combinações dos arquivos.

Em todos os testes, foram feitas divisões do *dataset* na proporção 50% para treinamento do modelo e 50% para teste do modelo. Isto significa que 400 registros de cada arquivo foram usados para treinamento e outros 400 para teste. No treinamento, havia 3 classes (normal, *spike* e *flash crowd*), então, usou-se *cross-validation* (validação cruzada) com k-*fold* igual a 3.O método de validação cruzada denominado *k-fold* divide o conjunto total de dados de treinamento em k subconjuntos, no caso de *k-fold* igual a 3 são realizados treinamentos em 3 subconjuntos de vetores, a validação cruzada consiste na repetição dos testes por 3 vezes, sendo que a média dos resultados é usada para estimação dos parâmetros e cálculo da acurácia do modelo. A "Fig. 20," mostra de maneira gráfica o processo de validação cruzada *k-fold* igual a 3.

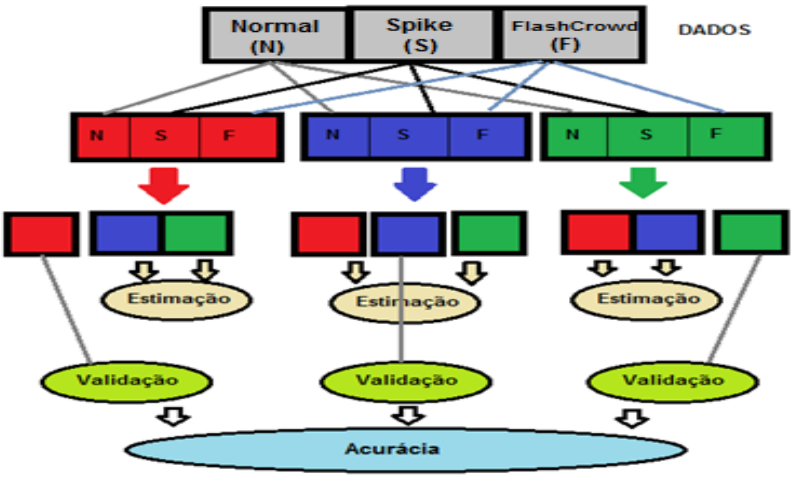


Figura 4.19 – Esquema de validação cruzada aplicada na proposta

Na tabela 4.2 mostra-se um conjunto de testes que envolvem a combinação de demandas e os resultados obtidos durante o treinamento e testes. Nos testes de 01 a 06 (tabela 4.2), foram combinados os arquivos dos extremos (teste01 com teste13) em direção ao centro das numerações das amostras (teste06 com teste08). Como dito anteriormente, a ideia era provar que as rotulações feitas pelo especialista se confirmavam quando o modelo era treinado e os vetores de teste eram identificados corretamente.

A contra hipótese era que se a identificação não ocorresse com acurácia alta, o *Spike* e o *Flash Crowd* não seriam facilmente discerníveis como relatava a literatura (principalmente da área de redes). As rotulações estavam incorretas ou os atributos não representavam características para *Spike* e *Flash Crowd*.

A surpresa agradável foi que todos os resultados foram muito bons, como pode ser visto na tabela 4.2. Porém, foi feita a seguinte consideração: quando os arquivos eram distintos para *Spikes* e *Flash*, o classificador de dados conseguia bons resultados porque as três classes eram distintas, quando não poderia se ter só duas classes o que também daria bom resultado.

Tabela 4.2– Combinação de arquivos e resultados de treinamento e teste

Teste no.	Perfis	*Cross-Validation* rate - treinamento (%)	*Accuracy rate* - teste (%)
Teste01	Normal, Spike01, Flash13	100,00	100,00
Teste02	Normal, Spike02, Flash12	99,92	99,92
Teste03	Normal, Spike03, Flash11	99.34	99.83
Teste04	Normal, Spike04, Flash10	99.92	100,00
Teste05	Normal, Spike05, Flash09	100,00	100,00
Teste06	Normal, Spike06, Spike08	100,00	99.33
Teste07	Normal, Spike06, Spike07	100,00	100,00
Teste08	Normal, Spike07, Spike08	100,00	100,00
Teste09	Normal, Spike07, Flash09	100,00	100,00
Teste10	Normal, Spike08, Flash09	100,00	100,00
Teste11	Normal, Spike08, Flash10	100,00	99,92
Teste12	Normal, Spike09, Flash10	100,00	88,67
Teste13	Normal, Flash09, Flash11	100,00	99,92
Teste14	Normal, Spike09, Flash12	99,75	91,83

Na tabela 4.2 em todas as combinações também está presente a carga de processamento considerada normal. Quando rotulado o perfil de processamento do tipo *Flash* 09 para *Spike* e realizando a combinação com cargas do tipo *Flash* conforme Teste 12 e 14 a acurácia foi relativamente menor.

Assim, investigou-se melhor a fronteira definida pelo especialista para *Spike* e *Flash*, resultando nas combinações envolvendo os arquivos de spike06 a flash10. Todos os resultados também foram bons, com exceção do teste12 conforme mencionado.

Como o flash09 já havia sido combinado com arquivos com numeração abaixo deste, testou-se combinações com numeração acima deste. O teste11 teve sucesso, mas o teste12 nem tanto. O que se pode concluir é que os arquivos até spike08 eram realmente *Spikes*.

Decidiu-se, então, intensificar a investigação com relação à rotulação, pois as outras duas contra hipóteses já haviam sido descaracterizadas com os bons resultados que as combinações dos testes de 01 a 06 apresentaram.

4.4 Analisando os vetores do Dataset

Com base nos resultados relatados na seção 4.3, todos os registros foram misturados formando uma só base; 50% dos registros da base para cada perfil: normal, *Spike* e *Flash Crowd* foram utilizados para treinamento com cross-validation (fold = 3) e os outros 50% da

base foram reservados para teste do modelo.

Foi feito mais um modelo para averiguar para quais perfis rotulados pelo especialista na fronteira entre os processamentos *Spike* e *Flash Crowd* obtinham melhor resultado com o classificador (Tabela 4.3). Se alguma combinação diferente da original gerasse melhor taxa de acerto, significaria que a fronteira entre *Spike* e *Flash* estaria rotulada erroneamente. Chega-se a esta conclusão devido ao bom resultado que o modelo apresentou nos testes anteriores à seção 4.3.

Tabela 4.3– Resultados de treinamento e teste todos os perfis juntos

Teste no.	**Perfis**	***Cross-Validation* rate - treinamento (%)**	***Accuracy rate* – teste(%)**
Teste00	Normal, Spike01-spike08, Flash09-Flash13	98.82	98.41 (5511/5600)

Os resultados da tabela 4.3 mostram que há uma espécie de região cinza nos perfis acima dos captados em spike08, com tendências a indicar que baseado no modelo de aprendizado fornecido pelo classificador, o perfil de *Spike* pode ser ampliado até o indicado anteriormente pelo especialista como *Flash Crowd*. O mais importante de todos os testes é que não parece haver dúvida que o perfil identificado pela carga de trabalho sintetizada nos perfis de spike01 até spike08 são realmente *Spikes*. Isto foi definido pelo especialista e confirmado pelo modelo gerado pelo classificador de dados, que as taxas de acerto são muito próximas de 100%.

Para os outros casos, fica a critério do administrador do ambiente de nuvem se deseja considerar o perfil do flash09 e flash10 com *Spike* ou manter a denominação capturada no perfil de carga de trabalho do spike08.

4.5 Seleção de atributos e novos testes

A seleção de atributos foi executada para tentar diminuir o ruído (imprecisão de classificação) e melhorar o desempenho do classificador. A partir dos 23 atributos da seção A foi realizada a seleção de atributos utilizando a técnica *filter*, usando a plataforma Weka (www.cs.waikato.ac.nz/ml/weka/). Usando *filte*r, uma técnica de seleção de atributos que usa características de dados para avaliação, independente do classificador a ser utilizado, foram

obtidos apenas 15 atributos. A Tabela 4.4 mostra quais os atributos que foram considerados discriminantes após o uso da técnica *filter* na seleção de atributos.

Tabela 4.4– Atributos selecionados pela técnica Filter

Métrica Coletada	**Descrição**
CpuSystemTotalLoad	Sistema de CPUs – Carga total (1)
CpuSystemLoad	Sistema de CPUs – Carga de sistema (2)
CpuSystemLoadAvg	Sistema de CPUs – Carga média (3)
CpuSystemUserLoad	Sistema de CPUs – Carga de usuário (4)
CpuSystemWaiting	Sistema de CPUs – Carga em espera (5)
CpuCore01TotalLoad	CPU Núcleo 01 – Carga total (9)
CpuCore01SystemLoad	CPU Núcleo 01 – Carga de sistema (10)
CpuCore01UserLoad	CPU Núcleo 01 – Carga de usuário (11)
CpuCore02TotalLoad	CPU Núcleo 02 – Carga total (12)
CpuCore2SystemLoad	CPU Núcleo 02 – Carga de sistema (13)
CpuCore02UserLoad	CPU Núcleo 02 – Carga de usuário (14)
MemPhyUsed	Memória física utilizada (15)
MemPhyApplication	Memória física utilizada pela aplicação (16)
MemPhyCached	Memória física usadas em caches (17)
MemPhyBuff	Memória física usada em buffers (18)

Em seguida foram repetidos todos os testes da Tabela 2 apenas com estes 15 atributos e os resultados apresentam-se na Tabela 5. Observe-se que nos teste01 – teste11 os resultados melhoram pouco, pois já eram muito bons. Porém, neste caso os testes foram refeitos para assegurar que os 15 atributos não alteravam negativamente os resultados anteriores (com 23 atributos). Mas, o objetivo principal foi estudar mais detalhadamente o comportamento do perfil *flash09*, pois a imprecisão resultante da classificação nos teste12 e teste14 (Tabela 2) deixou uma dúvida quanto a fronteira definida pelo especialista para os perfis *spike* e *flash crowd*. A seleção de características (atributos) foi utilizada para dirimir a dúvida com relação ao ruído resultante de características que poderiam estar atrapalhando a classificação, por não serem suficientemente discriminantes.

Como o problema não era referente a atributos, porque os resultados da classificação melhoraram, refizemos o teste12 e teste14 considerando os exemplos de vetores rotulados pelo especialista como *flash*, rerotulados agora como *spike* para avaliar se o resultado melhorava. A hipótese era de rotulação equivocada.

Tabela 4.5– Nova combinação de arquivos, treinamento e testes

Teste no.	**Perfis**	***Cross-Validation rate* - treinamento**	***Accuracy rate* - teste (%)**
Teste01	Normal, Spike01 e Flash13	100,00	100,00
Teste02	Normal, Spike02 e Flash12	100,00	100,00
Teste03	Normal, Spike03 e Flash11	99.91	99.91
Teste04	Normal, Spike04 e Flash10	100,00	100,00
Teste05	Normal, Spike05 e Flash09	100,00	100,00
Teste06	Normal, Spike06 e Spike08	100,00	100,00
Teste07	Normal, Spike06 e Spike07	100,00	100,00
Teste08	Normal, Spike07 e Spike08	100,00	100,00
Teste09	Normal, Spike07 e Flash09	100,00	100,00
Teste10	Normal, Spike08 e Flash09	99.83	99.83
Teste11	Normal, Spike08 e Flash10	100,00	100,00
Teste12	Normal, **Spike09** (re-rotulado) e Flash10	92,5	91,41
Teste13	Normal, Flash09 e Flash11	100,00	97.83
Teste14	Normal, **Spike09** (re-rotulado) e Flash12	91,66	92.66
Teste15	Normal, Spike01, Spike02, Spike03, Spike04, Spike05, Spike06, Spike07, Spike08, **Flash09**, Flash10, Flash11, Flash12 e Flash13	98,53	98,58
Teste16	Normal, Spike01, Spike02, Spike03, Spike04, Spike05, Spike06, Spike07, Spike08, Flash10, Flash11, Flash12 e Flash13	99.34	99.34

Observe-se que o resultado permaneceu praticamente o mesmo, o que nos permitiu concluir que o problema não é era de características não relevantes e nem de rotulação errônea, portanto a negativa da hipótese relatada na seção C não se confirmou. Assim, concluímos que o perfil armazenado em *flash09* representa a fronteira entre *spike* e *flash crowd*, devido aos resultados não significativos na taxa de acerto (Tabela 2 e Tabela 5), o que indica que há registros de *spike* e *flash* misturados no mesmo arquivo.

O teste15 representa o mesmo teste (Teste00) da Tabela 3 com 15 atributos. No teste16 retiramos os vetores do perfil *flash09* do *dataset* de treinamento e teste; observe que a taxa de classificação aumentou, o que mostra claramente que os registros do perfil *flash09* misturam vetores de *spike* e *flash crowd*. Assim, ficou comprovado que *flash09* contém vetores que identificam a fronteira entre os dois perfis de processamento e, portanto esta comprovada a hipótese do trabalho, que *spike* e *flash crowd* são distinguíveis.

Capítulo 5

Conclusão

Este trabalho apresentou uma estratégia para lidar com o surto de processamento em computação em nuvem e o uso do classificador SVM para identificar seus perfis, assunto não tratado pela literatura.

A sintetização de cargas de trabalho com perfil de *spikes* e *flash crowd*, assim predefinidas por um especialista, puderam ser comprovadas através da obtenção e validação (teste) do modelo dos perfis, utilizando-se o aprendizado de máquina, mais especificamente, o SVM. Além disto, o resultado dos testes com combinações de perfis *spike* e *flash* permitiram definir a fronteira entre os dois perfis de surto de processamento.

Os melhores resultados para acurácia de classificação foram obtidos após a seleção de atributos (características) – que eliminou 8 atributos considerados na avaliação com seleção manual de atributos, melhorando os resultados obtidos anteriormente. Um resultado importante desta fase dos testes foi a confirmação de que os atributos e a rotulação dos arquivos de perfis estavam corretos. Porém, foi muito importante identificar que no perfil *flash09* havia vetores misturados de *spike* e *flashcrowd*, o que definiu os valores daqueles atributos como a fronteira entre os dois perfis de surto de processamento.

Concluiu-se no desenvolvimento do trabalho que são necessários apenas 15 atributos referentes a consumo de CPU e memória para diferenciar *Spike* (perfis *spike01-spike08*) e *Flash Crowd* (perfis *flash10-flash13*). O perfil de processamento coletado em *flash09* não é conclusivo e, portanto não deve ser considerado na caracterização de nenhum tipo de surto.

Assim, nós consideramos os valores dos atributos do perfil *flash09* como a fronteira entre os dois tipos de surto de processamento.

Diferentemente dos trabalhos da literatura, esta proposta pode contribuir para a elasticidade na computação em nuvem, pois foi possível adotar uma estratégia de alocação local de VM (no mesmo *host* físico), sem se perder o poder de escalabilidade da nuvem. Assim, se durante o processamento for detectado o perfil de processamento *flash crowd* a VM pode ser migrada (instanciada) para outro host, inclusive em uma infraestrutura pública de nuvem. Esta estratégia minimiza os custos para o provedor de nuvem sem prejudicar o desenvolvimento da aplicação do cliente, pois se minimizará ou eliminará a sobrecarga de provisionamento de VM desnecessárias.

Como trabalho futuro se deseja confirmar que o uso desta abordagem no balanceamento de carga pode reduzir a sobrecarga em provedores de serviços.

Referências

[01] RAJKUMAR, B.; CHEE, S. Y.; SRIKUMAR, V.; JAMES, B.; IVONA, B. *Cloud computing and emerging IT platforms*: *Vision, hype, and reality for delivering computing as the 5th utility*. Vol. 25, ACM 2009, p.599-616.

[02] URGAONKAR, B et al. *Dynamic provisioning of multi-tier Internet applications*. Second International Conference on Autonomic Computing (ICAC), 2009.

[03] KALYVIANAKI, E et al. *Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters*. International conference on autonomic computing, 2009.

[04] BODIK P et al. *Statistical machine learning makes automatic control practical for Internet datacenters*. HotCloud, 2009.

[05] KRIEGER, A.; SIMON, J. *Autonomic Resource Management with Support Vector Machines*. Lyon Conference, 2011, p157-164.

[06] VECCHIOLA, C.; CHU, X.; BUYYA, R. *Aneka: A Software Platform for .NET-based Cloud Computing*. Australia, 2009.

[07] MELL, P.; GRANCE, T. *The NIST definition of cloud computing*. National Institute of Standards and Technology, 2009.

[08] ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R. H.; KONWINSKI, A.; LEE, G.; PATTERSON, D. A.; RABKIN, A.; STOICA, I.; ZAHARIA, M. *Above the clouds: A berkeley view of cloud computing*. EECS Departament, University of California, Berkeley, 2009.

[09] ROBINSON, D. *Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faser*. Emereo Pty Ltd, London, UK, 2008.

[10] LIU, S.; LIANG, Y.; BROOKS, M. *Eucalyptus: a web service-enabled infrastructure*. Proceedings of the 2007 conference of the center for advanced studies on Collaborative research, NY, USA. ACM, 2007, p.1-11.

[11] MOTERO, RUBENS.; LIORENTE, L.M.; FOSTER, I. *Virtual Infrastructure Management in Private and Hybrid Clouds*. Internet Computing IEEE. Volume 13, Issue 5, 2009, p.14-22.

[12] ZAHARIEV A. *Google App Engine*. Helsinki University of Technology, Seminar on Internetworking, 2009.

[13] LEAVITT N. *Is Cloud Computing Really Ready for Prime Time*. ComputerIEEE, Volume 42, Issue1, 2009, p.15-20.

[14] DELGADO, V. *Exploring the limits of cloud computing*. Master of Science Thesis Stockholm, Sweden. Kungliga Tekniska Högskolan (KTH), 2010.

[15] MONARD, M. C.; BARANAUSKAS, J. A. *Conceitos de aprendizado de máquina*.Sistemas Inteligentes - Fundamentos e Aplicações. Editora Manole, 2003, p.89–114.

[16] LORENA, A. C.; CARVALHO, A. C. P. L. F. *Uma introdução às Support Vector Machines*.Revista de Informática Teórica e Aplicada (RITA), Volume XIV, Número 2, 2007.

[17] LAGAR-CAVILLA, H. A.; WHITNEY, J.; SCANNELL, A.; PATCHIN, P.; RUMBLE, S. M.; LARA, E.; BRUDNO, M.; SATYANARAYANAN, M. *SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing*. EuroSys, Nuremberg, Germany, Abril 2009, p.1-12.

[18] BRYAN, Roy et al. *Kaleidoscope: Cloud Micro-Elasticity via VM State Coloring*. AT&T Labs Research. University of Toronto, 2011.

[19] GULATTI, A.; SHANMUGANATHAN, G.; AHMAD, I.; HOLLER, A. *Cloud Scale Resource Management: Challenges and Techniques*. Vmware Labs, 2009.

[20] EUN-KYU, B.; YANG-SUK, K.; JIN-SOO, K.; SEUNGRYOUL, M. *Cost optimized provisioning of elastic resources for application workflows*. Department of Computer Science, Korea Advanced Institute of Science and Technology, Oracle USA, Outubro 2010.

[21] BODÍK, P.; FOX, A.; FRANKLIN, M. J.; JORDAN, M. I.; PATTERSON, D. A. *Characterizing, Modeling, and Generating Workload Spikes for Stateful Services*. EECS Department, UC Berkeley, Berkeley, CA, USA, 2010.

[22] LIANG, C.; HIREMAGALORE, S.; STAVROU, A.; RANGWALA, H. *Predicting Network Response Times Using Social Information*. Center for Secure Information Systems. George Mason University, Fairfax, VA.Technical Reports, 2007.

[23] KDE. *System Monitor – ksysguard*. Acesso em: http://userbase.kde.org/KSysGuard, 2013.