Alocação Proativa de Recursos Virtualizados Aplicada à Computação em Nuvem

Paulo Cândido[1], Dionisio Leite[2], Marcos Santana[2], Regina Santana[2], Maycon Peixoto[1]

[1]Universidade Federal de Viçosa (UFV) – Campus Rio Paranaíba
Caixa Postal 22 – 38.810-000 – Rio Paranaíba – MG – Brasil

{paulo.candido,mayconleo}@ufv.br
[2]Universidade de São Paulo (USP)
São Carlos – SP – Brasil

{dionisio,mjs,rcs}@icmc.usp.br

Abstract. *In this paper are presented two approaches implemented for the component of scheduling policies of MACC (Metascheduler Architecture to Provide QoS in Cloud Computing). These approaches aim at providing quality of services to the Cloud Computing environment. The algorithm's role is to determine a set of virtual machines and their configurations to perform the services submitted by the client, respecting the contract stipulated previously. The experiments show that the proposed approaches allow to ensure the quality of requirements demanded by users.*

Resumo. *Neste trabalho são apresentadas duas abordagens implementadas para o componente de políticas de escalonamento do MACC (Metascheduler Architecture to provide QoS in Cloud Computing). Essas abordagens de metaescalonamento têm como objetivo fornecer qualidade de serviço ao ambiente de Computação em Nuvem. Os algoritmos possuem a função de determinar um conjunto de máquinas virtuais e suas configurações para a execução dos serviços submetidos pelo cliente, respeitando o contrato estipulado previamente. Os experimentos mostram que as abordagens propostas permitem assegurar os requisitos de qualidade solicitados pelos usuários.*

1. Introdução

Atualmente a computação pode ser entregue aos usuários de várias formas. Uma dessas formas é a Computação em Nuvem, que permite aos usuários acessarem serviços por meio da Internet, ou seja, sem a dependência direta de aplicações ou plataformas específicas [Armbrust et al. 2009]. Para fornecer esses serviços, esse modelo de computação conta com uma rede compartilhada de recursos e alto poder de processamento e armazenamento, que são fornecidos aos usuários de forma transparente, tornando a Computação em Nuvem tão atrativa [Mell and Grance 2011].

Hoje, os custos com *hardware* são baixos, porém o custo de manutenção de grandes conjuntos de *hardware* se torna alto, pois envolve treinamento de pessoal especializado e consumo elevado de energia [Assunção et al. 2010]. Com a Computação em Nuvem, as preocupações com treinamento de pessoal, consumo de energia e ciclos de processamento ociosos ficam a cargo do provedor de serviço e não mais a cargo do usuário.

Assim, os custos que são repassados aos usuários referem-se apenas aos recursos que ele realmente utiliza.

O provedor de serviços, baseado no nível de acordo de serviço (SLA - *Service Level Agreement*), entrega máquinas virtuais aos usuários de acordo com o contrato, a fim de prover o uso máximo dos recursos contratados. Como a virtualização permite dois tipos de escalabilidade: horizontal (quando o número de máquinas virtuais é incrementado) e vertical (quando os recursos da máquina virtual são incrementados), o provedor de serviços precisa decidir qual tipo de escalabilidade que irá utilizar para atender o SLA contratado pelo usuário. Uma das formas de tomar essa decisão é utilizando mecanismos orientados à qualidade de serviço (QoS) que baseados no SLA, fornecem a melhor configuração para os usuários.

[Peixoto et al. 2010] propõe um metaescalonador para Computação em Nuvem que se preocupa com os níveis de QoS dos usuários baseando-se no SLA contratado. Esse metaescalonador permite verificar, em várias camadas, qual a melhor configuração para atender as requisições dos usuários. Dentre essas camadas, existe o escalonador local que utiliza uma política de alocação dinâmica de máquinas ou SDA (*Slotted Allocation Dynamic*) que consegue atingir bons níveis de qualidade de serviço. No entanto, essa política não se preocupa com o menor preço a ser entregue ao usuário.

Sendo assim, serão propostas duas políticas para alocação de máquinas virtuais capazes de atender os requisitos de QoS impostos no SLA, alocando apenas o necessário para o cumprimento desses requisitos com o menor custo possível.

A contribuição deste trabalho é fornecer políticas de alocação de VMs que usam tanto a escalabilidade horizontal quanto vertical que auxiliarão não só na composição de preços baixos, mas também na maior utilização dos recursos físicos dos provedores de serviço de Nuvem.

O restante do artigo é organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados à metaescalonamento para o levantamento de melhorias que podem ser desenvolvidas. Na Seção 3 é apresentada a arquitetura do MACC, bem como o desenvolvimento das soluções para alocação de recursos. A Seção 4 apresenta os resultados obtidos e as discussões a respeito das melhorias obtidas. Na Seção 5 são apresentadas as conclusões e direcionamentos futuros deste trabalho. Ao final, são apresentadas as referências bibliográficas que embasam este artigo.

2. Trabalhos Relacionados

A distribuição de requisições de serviços pode ser feita utilizando *Brokers*, escalonadores e metaescalonadores. Os metaescalonadores foram utilizados, a princípio, na Computação em Grade (que pode ter várias camadas envolvidas no seu mecanismo de operação) e atualmente esse conceito foi estendido para Computação em Nuvem [Chard 2011].

O Trabalho de [Lai et al. 2005] introduz o *Tycoon*, um "mercado" de recursos baseado em participação proporcional. Este trabalho propõe a alocação de recursos computacionais de maneira eficiente e econômica através de leilões. Cada provedor executa um processo leiloeiro que aloca seus recursos físicos para uma ou mais máquinas virtuais.

[Iosup et al. 2007] combinaram as abordagens hierárquica e descentralizada para

conseguir uma solução eficiente para o problema de alocação de recursos em *Grid*. Sua solução consiste em conectar várias organizações, cada qual com sua infraestrutura, de maneira que elas cooperem entre si. Os autores utilizaram conexões *peer-to-peer* entre as organizações, descentralizando o trabalho. O cooperativismo das organizações aumenta o rendimento do conjunto como um todo.

[Huang et al. 2011] apresentam o CASA (Community Aware Scheduling Algorithm). Este é um algoritmo para metaescalonamento, cuja abordagem é dinâmica e descentralizada. Sua aplicação é específica para Computação em Grade e Computação em Nuvem. Consiste em duas fases, sendo a primeira, a busca pelo melhor nó onde a tarefa pode ser alocada. Na segunda fase, é feito o escalonamento dinâmico de forma interativa, visando apoiar ou renegociar a decisão tomada na primeira fase.

Especificamente, os Metaescalonadores possuem técnicas para realizar a alocação de máquinas virtuais. O processo de alocação de máquinas virtuais em Computação em Nuvens é complexo, pois a melhor configuração de *software* e *hardware* deve ser feita para garantir que os alvos de QoS possam ser atendidos, enquanto ocorre a maximização da utilização do sistema [Calheiros et al. 2011].

No trabalho de [Quiroz et al. 2009] é proposto um mecanismo dinâmico de alocação de máquinas virtuais. Nesse trabalho, é necessário determinar tanto o número de instâncias virtualizados quanto os seus tipos. Essa abordagem em duas fases é semelhante à utilizada pelo MACC, porém não trata aspectos relacionados à restrição temporal (*deadlines*) como é tratado pelo MACC.

Diferentemente do MACC que trabalha de forma proativa, outros trabalhos [Chieu et al. 2009] [Singhal et al. 2011] [Zhu et al. 2006] propõem um algoritmo reativo, baseado em teoria de controle, para alocação dinâmica de máquina virtuais. Entretanto, os autores não apresentam de maneira clara a sobrecarga sobre a realização de mudanças no cenário, pois a capacidade é alterada à medida que a exigência dos requisitos dos usuários sofre mudanças.

Alguns trabalhos já utilizam o MACC como suporte teórico às suas pesquisas, tais como: [Suresh and Vijayakarthick 2011] [Khoshdel et al. 2011] [Adhipta et al. 2011] [de Alencar et al. 2011]. Outro trabalho proposto em [Vernekar and Game 2012], basicamente deriva o conceito proposto pelo MACC [Peixoto et al. 2010] na construção de um projeto de alocação de recursos baseado em componente.

3. Modelagem do Sistema

O *Metascheduler Architecture to provide QoS on the Cloud Computing* ou apenas MACC, é uma arquitetura de metaescalonador apresentada por [Peixoto et al. 2010] que tem como princípio o gerenciamento de serviços e recursos de infraestrutura. A cada serviço recebido, é alocado um conjunto adequado de recursos visando um modelo econômico e o cumprimento dos requisitos de QoS.

Nessa arquitetura apresentada pelos autores, há um componente responsável por aplicar as políticas de escalonamento, que posteriormente serão executadas por outro componente nomeado LRAM (*Local Resource Allocation Manager*). A aplicação de políticas tem como base a carga de trabalho, o estado dos recursos locais e os requisitos de QoS. Os parâmetros necessários para a tomada de decisão como carga de trabalho

(quantidade de instruções dos serviços) e estado dos recursos serão estimados, calculados ou descobertos por outros componentes do MACC.

O MACC possui três políticas de escalonamento nativas. O primeiro algoritmo é utilizado como base para validação dos seus subsequentes. Trata-se de um algoritmo aleatório sem qualquer princípio inteligente.

O segundo, "SDA" (*Slotted Allocation Dynamic*), tem o objetivo de criar máquinas virtuais com as mesmas configurações para todos os serviços, variando apenas a quantidade de VMs que serão alocadas, configurando assim a escalabilidade horizontal. Como a configuração das VMs é inflexível, os recursos são oferecidos em degraus, e uma pequena diferença entre cargas de trabalho pode justificar a criação de uma nova máquina virtual que não será utilizada completamente.

O algoritmo "SDA" possui duas variações. Na primeira, as VMs são alocadas com 2 núcleos virtuais cada, sendo a capacidade mínima dos *datacenters* modelados. À esta variação dá-se o nome de "SDA2c". Na segunda variação, as VMs são alocadas cada uma com 4 núcleos virtuais. Essa segunda variação foi nomeada "SDA4c".

O último algoritmo, "PDA" (*Predictive Allocation Dynamic*), propõe uma solução para o problema apresentado no "SDA". As máquinas virtuais podem ser diferentes entre si, podendo ter os núcleos de processamento virtuais com velocidades distintas. Isso diminui o efeito de degraus criado pelo segundo algoritmo, entretanto, a quantidade destes núcleos ainda é fixa.

Para decidir a velocidade dos núcleos, o algoritmo calcula a quantidade total de MIPS necessários para cumprir o *deadline* dos serviços recebidos. Caso este valor supere o limite físico da máquina hospedeira, a carga é dividida igualmente em duas máquinas virtuais.

3.1. PDA *Aggressive*

Para o problema proposto, foram implementadas duas soluções. A primeira foi nomeada "PDA *Aggressive*", baseada no "PDA".

Na estrutura do MACC, há um componente responsável pela contabilização dos custos de criação e utilização de VMs. Estes dois custos compõem o custo total para a execução dos serviços. No ambiente modelado, o custo de criação caracteriza-se pela soma dos custos unitários de cada componente (processamento, armazenamento, memória, etc) multiplicado pela quantidade alocada. O custo de utilização é igual ao custo de criação multiplicado pelo tempo de utilização e dividido por uma constante de normalização.

Através de uma análise neste componente, percebe-se que o custo para alocar duas VMs com dois núcleos cada, é maior que o custo para alocar uma VM com quatro núcleos. De maneira semelhante, o custo para alocar dois núcleos com x MIPS é maior que o de alocar um único núcleo com $x * 2$ MIPS. Através dessa premissa, o algoritmo "PDA" foi modificado para acumular o máximo possível de MIPS em um núcleo e o máximo de núcleos possíveis em uma VM, economizando, assim, no custo com a criação.

Como as VMs criadas com essa política tem a quantidade de núcleos variável, a solução torna-se mais flexível em relação ao "PDA", ou seja, as máquinas virtuais são

mais justas em relação ao problema, diminuindo os recursos ociosos.

Neste algoritmo foi adicionada uma taxa de segurança para que haja uma folga entre o *deadline* requisitado e o oferecido, evitando o descumprimento do SLA por fatores externos, como a latência da rede. No Algoritmo 1 é representado o pseudocódigo dessa solução.

```
1  mips ← ceil(carga / deadline);
2  mips ← mips + (mips * taxaSeguranca);
   /* nesse ponto, mips vale o montante de MIPS necessários          */
3  nucleos ← 1;
4  while mips > mipsMaximo do
5  |   mips ← mips / 2;
6  |   nucleos ← nucleos * 2;
7  end
   /* nesse ponto, mips já está ajustado, nucleos contém a quantidade de
      núcleos necessários                                            */
8  vms ← 1;
9  while nucleos > nucleosMaximo do
10 |   nucleos ← nucleos / 2;
11 |   vms ← vms * 2;
12 end
   /* fim do algoritmo, tem-se a quantidade de vms, cada uma com uma
      quantidade de nucleos com determinado mips                     */
```

Algoritmo 1: PDA *Aggressive*

3.2. PAR - *Prudent Allocation of Resources*

Apesar do "PDA *Aggressive*" atingir bons níveis de qualidade de serviço, há um problema com *datacenter* heterogêneo (possui *hosts* com diferentes configurações) quando há sobrecarga. Como o intuito do primeiro algoritmo é acumular o máximo de recursos em um único elemento (MIPS em núcleos e núcleos em VMs), ele poderá criar máquinas virtuais impossíveis de serem instanciadas, caso os *hosts* com o maior poder computacional estejam ocupados.

O MACC possui um componente que oferece as informações sobre o estado dos *hosts*, que até então não era utilizado. Neste segundo algoritmo, o estado dos *hosts* é consultado e as constantes passam a variar ($mipsMaximo$ e $nucleosMaximo$). Assim, os valores dependerão dos *hosts* disponíveis.

Há também, uma preocupação em não se utilizar *hosts* mais potentes que o necessário. Caso o pacote de serviços recebidos seja considerado pequeno, medido em MI (milhões de instruções), ele necessitará de uma máquina virtual mais simples, que implica em um hospedeiro também simples. Com essa estratégia, os *hosts* potentes serão poupados e continuarão disponíveis para cargas maiores que venham a ser submetidas, tendo em vista que não há previsão da chegada de pacote de serviços.

Outra vantagem desse algoritmo é o fato de que as máquinas virtuais agora poderão ser criadas com configurações diferentes, isso aumenta a flexibilidade da solução apresentada. Neste algoritmo, cada serviço será tratado individualmente e agrupado de maneira que a VM criada tenha o tamanho adequado para aquele grupo. Assim, as configurações serão definidas pela relação entre os serviços e os *hosts*.

O gráfico da Figura 1 apresenta o relacionamento entre as variáveis e o requisito de QoS (referente ao tempo) utilizados pelo PAR.

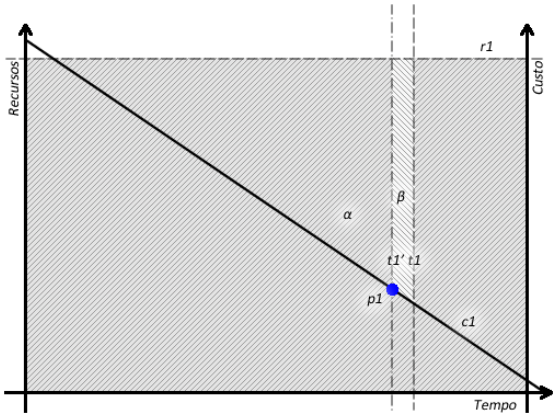

Figura 1. QoS e Objetivo

A reta *r1* representa o limite físico de recursos, enquanto a reta *t1* representa o requisito do SLA referente ao tempo. A curva *c1* representa a proporção mínima entre recursos e tempo. Soluções abaixo dessa curva são impossíveis de reproduzir.

A área $\alpha \cup \beta$, será chamada de área factível. Um resultado pertencente à área infactível acima da curva *c1*, significa um ferimento no acordo SLA. A reta *t1'* representa o fator de segurança para o requisito tempo do SLA. Este evitará que o SLA não seja ferido por elementos externos, como a latência da conexão. Desta forma, a região β deve ser evitada e a área α é chamada de área segura.

O objetivo do algoritmo é atingir um resultado pertencente à área segura. Dentro dessa área, qualquer resultado oferecido será satisfatório para o cliente, então o provedor deve moldar o resultado para que lhe seja viável. Na área α, o resultado deve tender para o menor custo e o maior tempo possível. Dessa forma, o provedor economiza recursos (maior tempo) e reduz os gastos. O ponto *p1* é o resultado ideal, tendo o menor custo e maior tempo dentro da área segura, sendo o objetivo do algoritmo a ser construído.

Outro fator que foi considerado na construção do PAR, foi a maneira como os serviços serão executados. Os serviços são recebidos em formas de pacotes com uma ou várias unidades. Cada unidade de serviço será processada por um único núcleo. Isso afeta a maneira como as máquinas virtuais são alocadas. Para uma unidade de serviço com uma elevada quantidade de instruções (medidas em MI), não há efeito alocar vários núcleos pois apenas um será utilizado. Assim, uma unidade de serviço será inevitavelmente entregue com atraso, caso tenha mais instruções que o melhor núcleo consegue processar no tempo *deadline*.

Na primeira linha do algoritmo 2 (um pseudocódigo do PAR) é chamada a função "*sort*". Essa irá ordenar os *hosts* de acordo com a sua capacidade atual. Os primeiros serão os com maior capacidade computacional. O termo *"host"* é uma lista contendo todos os *hosts* disponíveis no *datacenter*. Essa informação é disponibilizada pelo componente "MDSM" do MACC.

No primeiro laço de repetição, iniciado na linha dois, todos os serviços serão visitados e agrupados de acordo com seu tamanho (em MI). O ponto de referência é o melhor *host* disponível do *datacenter*. O grupo é iniciado vazio, e os serviços são adicionados até que o tamanho total exceda a capacidade do melhor *host* disponível. Nesse momento,

```
1  sort(host);
2  while ∀ servico do
3      if o grupo conseguir ser executado pelo host [0] se esse serviço for adicionado then
4          adiciona o serviço ao grupo;
5      else
6          fecha o grupo;
7          nucleos ← host [0].getFreePesNumber();
8          if nucleos > grupo.size() then
9              nucleos ← grupo.size();
10             mips ← grupo.biggest;
11             mips ← min(mips, mipsMaximo);
12         else
13             mips ← ceil((grupo.getMI() + taxaSeguranca )/ nucleos);
14         end
15         addVM(nucleos, mips);
16         grupo.clear;
17         remove o host da lista de hosts disponíveis;
18     end
19 end
20 if grupo.size() > 0 then
21     fecha o grupo;
22     mips ← ceil((grupo.getMI() + taxaSeguranca )/ deadline);
23     nucleos ← 1;
24     while mips > mipsMaximo do
25         mips ← mips / 2;
26         nucleos ← nucleos * 2;
27     end
28     if nucleos > grupo.size() then
29         nucleos ← grupo.size();
30         mips ← grupo.biggest;
31         mips ← min(mips, mipsMaximo);
32     end
33     addVM(nucleos, mips);
34 end
```

Algoritmo 2: PAR - *Prudent Allocation of Resources*

é criado um novo grupo com esse último serviço e o grupo anterior é fechado.

Na linha oito é verificado se a quantidade de núcleos designados é maior que a quantidade de serviços no grupo alocado. Não adiantará alocar mais *hosts* que serviços, pois eles não serão utilizados. O que se pode fazer é elevar o MIPS dos núcleos ao máximo permitido pelo *host*.

Ao término do laço iniciado na linha dois, o fato de existir algum serviço no grupo, indica que não houve necessidade de alocá-lo para o melhor *host* disponível e será alocado separadamente. Nesse caso, não há de se preocupar com a quantidade de núcleos que será necessária, pois certamente não será mais que o melhor *host* consegue oferecer. Foi utilizada para essa segunda parte, uma abordagem bem parecida com o "PDA *Aggressive*", porém também foi realizado na linha vinte e oito o controle para quando forem alocados mais núcleos do que serviços.

Em relação ao "PDA *Aggressive*", seu critério de alocação é mais responsável e ponderado. Dessa característica surgiu seu nome: *"Prudent Allocation of Resources"*, ou "Alocação Prudente de Recursos".

4. Resultados Obtidos

Os resultados apresentados foram obtidos através de simulações utilizando o *framework CloudSim* [Calheiros et al. 2010]. Os parâmetros consultados durante a simulação para a

validação do algoritmo proposto, são apresentados da seguinte forma:

- **Taxa de Segurança:** Este parâmetro pertence aos algoritmos criados neste trabalho. Seu propósito é manter um nível de segurança entre o QoS oferecido e o requisitado. É aceitável qualquer valor real positivo entre zero e um. Nas simulações feitas, o valor utilizado foi de 0.25, ou seja, vinte e cinco por cento. Isso significa que serão alocados vinte e cinco por cento de recursos a mais que o necessário, diminuindo o tempo de execução, para contrabalancear com o tempo de alguns fatores externos, como a latência da rede e o tempo da admissão do serviço.
- **Algoritmo de Alocação de VMs:** Este parâmetro define qual algoritmo será utilizado para alocação de recursos locais. Poderá assumir cinco valores, sendo: Aleatório, SDA, PDA, PDA *Aggressive* e PAR. Os três primeiros são nativos no MACC e os dois últimos são as soluções apresentadas nesse trabalho.
- **Tipo de Serviço:** Define o tipo dos serviços que serão submetidos durante a simulação. Os valores possíveis são: leve, médio, pesado e muito pesado, sendo que o tamanho em MI para cada um, compreende entre cinco mil a seis mil; vinte mil a vinte e cinco mil; noventa mil a noventa e cinco mil; cento e cinquenta mil a cento e cinquenta e cinco mil MI respectivamente. Com esses valores, espera-se testar cenários variados que vão dos mais simples aos sobrecarregados.

Para realizar comparações justas entre os algoritmos, serão utilizadas as métricas de qualidade de serviço:

- **Custo Total:** Este é o custo gasto para se executar todos os serviços dos usuários. O valor é representado em unidades monetárias. O objetivo é minimizar este valor sem ferir os outros requisitos de QoS.
- **Tempo médio de Serviço:** Este valor retrata o tempo médio entre a submissão do serviço e o seu recebimento do ponto de vista do usuário. De acordo com os objetivos apresentados, este valor deve se aproximar ao máximo do *deadline* sem ultrapassá-lo. Por essa métrica não é possível inferir se algum serviço ultrapassou o *deadline*, porém, a métrica "Confiabilidade" indicará isso.
- **Confiabilidade:** Este é um valor em percentual que indica o quão confiável foi o provedor em relação aos serviços executados. Se todos os serviços forem executados cumprindo o *deadline* acordado, o valor da confiabilidade será de cem por cento. Entretanto, a cada serviço descumprido, o valor é decrementado, apontando uma falha no sistema.

Várias simulações foram executadas variando os parâmetros de entrada, a fim de comparar o desempenho dos algoritmos propostos. Os resultados referentes ao custo obtido, em unidades monetárias, são apresentados na Tabela 1 e no gráfico da Figura 2.

Tabela 1. Relação de Custos

	Leve	Médio	Pesado	Muito Pesado
Aleatório	63.162, 48	83.318, 37	167.318, 37	239.318, 37
SDA 4c	8.102, 64	53.285, 73	$1,14 \times 10^{14}$	$1,14 \times 10^{14}$
PDA	6.866, 81	27.645, 26	191.645, 87	20.831.232, 76
PDA *Aggressive*	6.849, 33	27.026, 95	139.057, 09	7.357.990, 41
PAR	6.849, 33	27.026, 95	116.016, 95	237.113, 37

Nota-se que no algoritmo "SDA 4c" houve um aumento significativo no custo em situações de sobrecarga. Considerando apenas o custo, o algoritmo "Aleatório" obteve

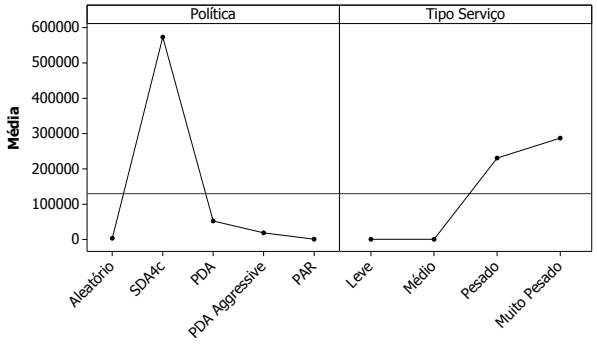

Figura 2. Relação de Custos

resultados acima da média nas situações de carga leve e média, mas em situações de sobrecarga, obteve resultados melhores que alguns algoritmos.

Os algoritmos "PDA *Aggressive*" e "PAR" obtiveram resultados iguais para situações de carga leve e média. Esse comportamento era esperado, visto que a implementação do "PAR" baseia-se no "PDA *Aggressive*". Essa melhoria trata situações onde os grupos de serviços criados não chegam a ocupar os melhores *hosts* do *datacenter*.

Analisando os resultados referentes à Confiabilidade, mostrados na Tabela 2 e no gráfico da Figura 3, percebe-se que no algoritmo "Aleatório" (apesar de conseguir bons custos em situações de sobrecarga) seu grau de confiabilidade foi 0%. Isso implica dizer que poucos recursos foram alocados, abaixando o custo total, porém esses recursos não foram suficientes para executar as tarefas no tempo pré-determinado.

Tabela 2. Relação de Confiabilidade

	Leve	**Médio**	**Pesado**	**Muito Pesado**
Aleatório	$99,83\%$	$77,67\%$	$0,00\%$	$0,00\%$
SDA 4c	$100,00\%$	$99,83\%$	$0,00\%$	$0,00\%$
PDA	$100,00\%$	$97,50\%$	$40,00\%$	$0,00\%$
PDA *Aggressive*	$100,00\%$	$100,00\%$	$100,00\%$	$0,00\%$
PAR	$100,00\%$	$100,00\%$	$100,00\%$	$99,33\%$

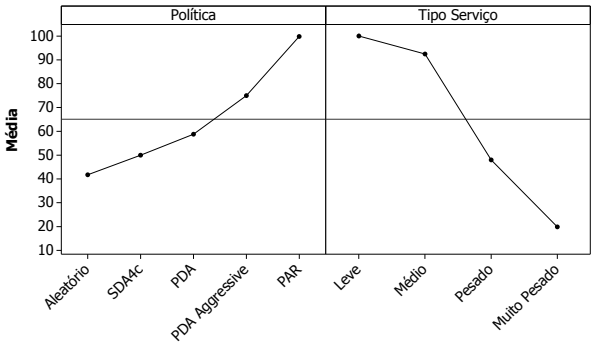

Figura 3. Relação de Confiabilidade

Exceto o algoritmo "Aleatório", todos os outros obtiveram cem por cento de confiabilidade para a carga leve. Na carga média, houve uma pequena perda em algumas abordagens. Na carga pesada, o algoritmo "SDA" se mostrou ineficiente e houve uma

queda significativa no “PDA”. Já o “PDA *Aggressive*” e o “PAR” continuaram estáveis. No último tipo de carga, todos os algoritmos, com exceção do “PAR”, obtiveram o grau de confiabilidade zerado. Nessa última situação, o “PAR” obteve um resultado satisfatório, em vista da carga excessiva de trabalho que foi submetida ao *datacenter*.

Cruzando-se as informações de confiabilidade e custo, observa-se que o algoritmo “SDA” em situações de sobrecarga, além de elevar o custo acima da média, obteve baixo grau de confiabilidade. Isso confirma que esse algoritmo é funcional, porém não é a melhor opção para um *datacenter*.

Em uma análise do tempo médio, apresentada na Tabela 3 e no gráfico da Figura 4, observa-se que os algoritmos “Aleatório” e “SDA” extrapolaram os tempos de resposta esperados em situações de sobrecarga.

Tabela 3. Relação de Tempo

	Leve	**Médio**	**Pesado**	**Muito Pesado**
Aleatório	$8,05s$	$35,67s$	$149,41s$	$246,53s$
SDA 4c	$11,38s$	$26,13s$	$92,42s$	$152,32s$
PDA	$6,98s$	$35,16s$	$38,98s$	$61,03s$
PDA *Aggressive*	$31,39s$	$31,90s$	$34,82s$	$55,63s$
PAR	$31,69s$	$31,91s$	$33,59s$	$38,13s$

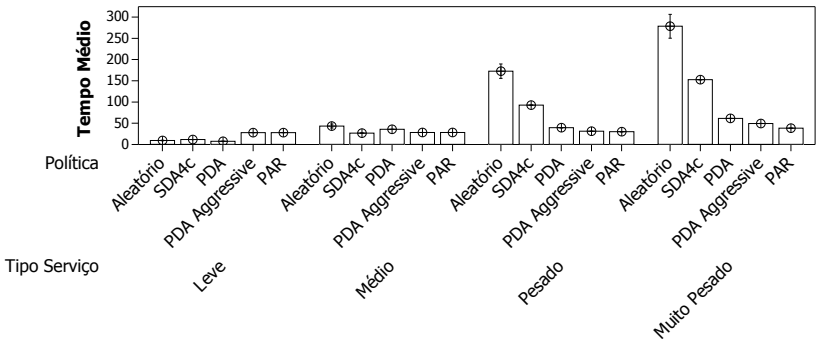

Figura 4. Relação de Tempo

Nota-se um fato importante em relação ao tempo médio: todos os algoritmos nativos, na carga leve, configuram o processamento em um tempo bem abaixo do *deadline*; que na modelagem apresentada é de quarenta segundos. As VMs geradas por esses algoritmos possuem mais recursos que o necessário, diminuindo o tempo, mas também elevando o custo. Já o “PDA *Aggressive*” e o “PAR” configuram VMs mais simples, com o objetivo apenas de atingir o *deadline*. Essa é uma medida para economia de recursos, e como visto no gráfico da Figura 2, foi eficaz.

Melhores resultados podem ser obtidos aproximando ainda mais o tempo médio do *deadline*, sem diminuir o nível de confiabilidade. O aumento do tempo implica na redução do custo, gerando mais economia para o provedor.

5. Conclusão

Este trabalho apresentou duas soluções possíveis para o problema de dimensionamento de máquinas virtuais aplicado ao MACC. A primeira solução, “PDA *Aggressive*”, tem um

comportamento de agregação. Foi baseada em uma solução nativa, "PDA", e endossada com uma abordagem acumulativa. Quanto mais núcleos forem acumulados em uma única VM, menor o custo por núcleo. De maneira semelhante, quanto maior a potência dos núcleos, menor será o custo por unidade de velocidade (MIPS).

A redução do custo condiz com um dos objetivos apresentados para este trabalho. Entretanto, nota-se que em situações de sobrecarga, essa solução não é viável. O estado dos recursos não é consultado para a tomada de decisão, dessa forma, recursos inexistentes podem ser alocados de maneira equivocada. Esse erro causa um aumento significativo no custo final e no tempo de execução.

A partir dessa premissa, foi desenvolvida a segunda solução, "PAR". Esse algoritmo tem a vantagem da consulta do estado dos recursos locais antes de alocá-los. Outra característica é o modo de interpretar os serviços. Nos algoritmos nativos e no "PDA *Aggressive*", a alocação de recursos é realizada proativamente. No "PAR" os serviços são verificados individualmente e agrupados de acordo com os recursos disponíveis. Os resultados mostram que o "PAR", na maioria das situações dentro dos parâmetros analisados, gera economia de recursos e respeita o QoS definido em SLA.

Em suma, a contribuição deste trabalho para a comunidade foi a apresentação do "PDA *Aggressive*" e "PAR": duas políticas de escalonamento de recursos locais que podem ser aplicadas ao MACC, uma arquitetura para gestão infra-estrutural de Computação em Nuvem. Provedores que utilizarem o MACC associados a essas políticas poderão oferecer a seus clientes recursos sob demanda, com baixo custo e cumprindo com os requisitos de QoS.

6. Agradecimentos

Os autores agradecem a ajuda financeira da FAPEMIG.

Referências

Adhipta, D., Hassan, M., and Mahmood, A. (2011). Elasticity study: Dbrain project grid-cloud integration. *Software Engineering and Computer Systems*, pages 652–662.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Assunção, M. D., Costanzo, A., and Buyya, R. (2010). A cost-benefit analysis of using cloud computing to extend the capacity of clusters. *Cluster Computing*, 13:335–347.

Calheiros, R., Ranjan, R., and Buyya, R. (2011). Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 295 –304.

Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., and Buyya, R. (2010). Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience, 2011*, pages 23–50.

Chard, K. (2011). *DRIVE: A Distributed Economic Meta-Scheduler for the Federation of Grid and Cloud Systems*. Tese de Doutorado - Victoria University of Wellington, Wellington, Nova Zelandia.

Chieu, T. C., Mohindra, A., Karve, A. A., and Segal, A. (2009). Dynamic scaling of web applications in a virtualized cloud computing environment. In *ICEBE '09: Proceedings of the 2009 IEEE International Conference on e-Business Engineering*, pages 281–286, Washington, DC, USA. IEEE Computer Society.

de Alencar, J., Andrade, R., Viana, W., and Schulze, B. (2011). P2pscheme: a p2p scheduling mechanism for workflows in grid computing. *Concurrency and Computation: Practice and Experience*.

Huang, Y., Bessis, N., Norrington, P., Kuonen, P., and Hirsbrunner, B. (2011). Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm. *Future Generation Computer Systems*.

Iosup, A., Epema, D. H. J., Tannenbaum, T., Farrellee, M., and Livny, M. (2007). Inter-operating grids through delegated matchmaking. *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 13:1–13:12.

Khoshdel, V., Motamedi, S., Sharifian, S., and Farhadi, M. (2011). A new approach for optimum resource utilization in cloud computing environments. In *ICCKE2011, International Conference on Computer and Knowledge Engineering*, pages 285 – 292, Machhad, Iran.

Lai, K., Rasmusson, L., Adar, E., Zhang, L., and Huberman, B. A. (2005). Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, 1:169–182.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, Information Technology Laboratory.

Peixoto, M. L. M., Santana, M. J., Estrella, J. C., Tavares, T. C., Kuehne, B., and Santana, R. H. C. (2010). A metascheduler architecture to provide qos on the cloud computing. In *ICT '10: 17th International Conference on Telecommunications*, pages 650 – 657, Doha, Qatar. IEEE Computer Society.

Quiroz, A., Kim, H., Parashar, M., Gnanasambandam, N., and Sharma, N. (2009). Towards autonomic workload provisioning for enterprise grids and clouds. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 50 –57.

Singhal, S., Wang, Z., and Zhu, X. (2011). Utilization and slo-based control for dynamic sizing of resource partitions. *IFIP Lecture Notes in Computer Science (LNCS)*, 3775(3775):133–144.

Suresh, A. and Vijayakarthick, P. (2011). Improving scheduling of backfill algorithms using balanced spiral method for cloud metascheduler. In *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 624 –627.

Vernekar, S. and Game, P. (2012). Component based resource allocation in cloud computing. In *Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012) held in Visakhapatnam, India, January 2012*, pages 907–914. Springer.

Zhu, X., Wang, Z., and Singhal, S. (2006). Utility-driven workload management using nested control design. In *American Control Conference, 2006*, page 6 pp.