Mecanismos para consolidação de servidores

Max Rosan dos Santos Júnior

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação

Orientador: Prof. Dr. Daniel Batista

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, março de 2014

Mecanismos para consolidação de servidores

Esta é a versão original da dissertação elaborada pelo
candidato Max Rosan dos Santos Júnior, tal como
submetida à Comissão Julgadora.

Resumo

DOS SANTOS JÚNIOR, M. R. **Mecanismos para consolidação de servidores**. 2014. 66 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2014.

Com o aumento na utilização dos Centros de Processamento de Dados (CPDs), causado principalmente pelo advento da computação em nuvem, uma das principais preocupações na gerência de CPDs passou a ser o consumo eficiente de energia. Um dos problemas que dificulta o consumo eficiente de energia nos CPDs é a subutilização das máquinas físicas; por exemplo, há registros de que, em média, servidores em CPDs do Google tiveram um uso de 10% a 50% da sua capacidade computacional máxima no ano de 2013. O problema é que nessa faixa de utilização o consumo de energia não fica na faixa de 10% a 50% do consumo máximo de energia, mas sim em uma faixa maior. Com o objetivo de utilizar os servidores de forma mais eficiente em termos energéticos, um dos métodos propostos é fazer a consolidação de servidores por meio da virtualização, onde busca-se alocar a menor quantidade possível de servidores para todas as máquinas virtuais sem descumprir os acordos de nível de serviço. Nesse contexto, esta dissertação propõe uma heurística, nomeada de *Toyoda-based Algorithm for Server Consolidation* (TASC), para alocação dinâmica de máquinas virtuais baseada na heurística de Toyoda, que por sua vez é uma heurística para resolver o problema da mochila. Além da heurística TASC, uma formulação em programação linear com o mesmo objetivo também é apresentada. A fim de avaliar os ganhos em termos de consumo de energia da heurística TASC, foi implementado um simulador com suporte a traces com registros de utilização de máquinas localizadas em CPDs do Google. Resultados das simulações mostram que o TASC consegue melhorar a alocação dos servidores quando comparado com as alocações feitas pelo algoritmo *First Fit Decreasing* (FFD), um algoritmo geralmente utilizado na literatura para comparar os ganhos de novas propostas de alocações de máquinas virtuais. Os ganhos do TASC variaram de 1% a 30%. Em adição, esta dissertação também apresenta um modelo para estimar o consumo de energia de um CPD, considerando como principal fator o uso de CPU dos servidores. Esse modelo foi construído com base em experimentos realizados para estudar o consumo de energia de um servidor real.

Palavras-chave: consolidação de servidores, virtualização, economia de energia, computação verde.

Abstract

DOS SANTOS JÚNIOR, M. R. **Mechanisms for server consolidation**. 2014. 66 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2014.
With the increase in the use of data centers, caused mainly by the advent of cloud computing, the energy efficiency became a major concern in the data center management. One of the problems that hinders the energy efficiency in data centers is the under-utilization of physical machines; for example, there are reports that, on average, servers in Google data centers used 10% to 50% of its maximum computing capacity in the year of 2013. The problem is with this utilization, the energy consumption is not in the range of 10% to 50% of maximum power consumption, but in a wider range. In order to use servers more efficiently in terms of energy consumption, one of the proposed methods is to consolidate servers through virtualization, which seeks to allocate the least possible amount of servers for all virtual machines and fulfilling the service level agreements. In this context, this dissertation proposes a heuristic, named Toyoda-based Algorithm for Server Consolidation (TASC) for dynamic allocation of virtual machines, based on Toyoda's heuristic, which is a heuristic to solve the knapsack problem. Besides the TASC heuristic, a linear programming formulation for the same purpose is also presented. In order to evaluate the gains in terms of energy consumption when the TASC heuristic is used, a simulator that supports traces of servers utilization provided by Google was implemented. Simulation results show that TASC can improve the allocation of servers when compared to the allocations made by the First Fit Decreasing (FFD) algorithm, an algorithm commonly used in the literature to compare the gains of new proposals for allocation of virtual machines. Gains ranged from 1% to 30% . In addition, this dissertation also presents a model for estimating the energy consumption of a data center, considering the CPU usage as the main factor. This model was built based on experiments conducted to study the power consumption of a real server.

Keywords: server consolidation, virtualization, energy saving, green computing.

Conteúdo

Lista de Símbolos

$S_i[r]$	Conjunto de máquinas virtuais (MV) alocadas no servidor i na rodada r.
$C_i^t[r]$	Demanda por CPU pela MV i na rodada r.
$M_i^t[r]$	Demanda por memória pela MV i na rodada r.
$C_i^s[r]$	Quantidade de CPU ofertada pelo servidor i na rodada r.
$M_i^s[r]$	Quantidade de memória ofertada pelo servidor i na rodada r.
$S[r]$	Lista de servidores disponíveis na rodada r.
F_i^c	Quantidade de CPU livre do servidor i.
F_i^m	Quantidade de memória livre do servidor i.
$Macs$	Lista de servidores disponíveis.
$Tasks$	Lista de todas as MVs.
C_i^t	Demanda por CPU pela MV i.
M_i^t	Demanda por memória pela MV i.
C_i^s	Quantidade de CPU ofertada pelo servidor i.
M_i^s	Quantidade de memória ofertada pelo servidor i.
W_{on}^j	Consumo inicial de energia de um servidor j.
W_{cpu}^j	Taxa de consumo de energia em Wh por uso de CPU de um servidor j.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Com o advento da computação em nuvem, a computação distribuída, antes restrita a ambientes voltados para computação de alto desempenho, passou a ser uma realidade para boa parte da população. Jogos [OnL13], armazenamento de arquivos [Dro] e mesmo aplicativos como editores de texto [Goo14] que antes eram executados localmente, passaram a ser armazenados e processados remotamente, bastando para o usuário final ter acesso à Internet por dispositivos como celulares e TVs inteligentes, além de aplicativos clientes bem mais leves do que os anteriormente instalados em computadores de propósito geral. Além da necessidade de conexões à Internet de boa qualidade, para que todo o ecossistema das nuvens funcione corretamente é necessário que haja capacidade disponível nos CPDs das empresas que fornecem os serviços [AFG+10].

Uma inevitável consequência da popularização dos serviços fornecidos pelas nuvens tem sido o crescimento da quantidade de recursos disponibilizados pelos CPDs para dar conta da imprevisibilidade da demanda. Enquanto antes os CPDs eram voltados para a execução de tarefas bem conhecidas que utilizavam alto poder computacional, agora eles precisam lidar com a execução de tarefas dos mais diversos perfis [BBA10]. Mais recursos e mais CPDs para atender a demanda das aplicações em nuvens leva ao aumento no consumo de energia caso não sejam aplicados mecanismos que evitem a utilização de mais recursos do que o necessário.

Dentre as várias particularidades que contribuem para o aumento no consumo de energia dos CPDs, agora utilizados para a execução de serviços para nuvens [Col11, BB10], há a baixa utilização de uma quantidade grande de servidores. Essa baixa utilização contribui não só para o custo no consumo de energia, como também para um custo maior na manutenção, dado que muitos componentes internos dos computadores tem um tempo de vida útil que é contabilizado independente da sua taxa de utilização. Vogel já relatava em 2008 que o grande número de servidores não utilizados ou com baixa utilização era um dos maiores problemas constatados pelos departamentos de TI de grandes empresas [Vog08]. Na época era estimado que em média, somente 15 a 20% da capacidade dos servidores eram utilizados naquelas empresas. Em 2013 o problema continuou sendo relatado tanto pela indústria [Kan13] quanto pela academia [AGPR13]. Por exemplo, o Google revelou, através de um monitoramento em alguns de seus servidores utilizados para a execução de serviços remotos, que esses servidores funcionaram em média de 10 a 50% de sua capacidade durante um período de 3 meses[BCH13]. Sem dúvida essa faixa representa um aumento significativo em relação à realidade de 2008, mas ainda está longe do ideal que seria manter as máquinas funcionando próximo dos 100%. A frase de Urs Höelzle, vice-presidente de infraestrutura técnica do Google resume a urgência por soluções que resolvam o problema da subutilização de servidores: "*If you want to save*

energy, get rid of your servers" [Kan13] que em tradução livre quer dizer "Se você quer economizar energia, livre-se dos seus servidores".

A baixa utilização dos servidores pode ser ocasionada por diversos fatores, um deles é a dedicação de servidores a determinados serviços ou sistemas operacionais. Com a virtualização já é possível rodar diversos serviços e sistemas operacionais em um mesmo servidor físico e ainda manter o isolamento entre eles. As chamadas Máquinas Virtuais (MVs) permitem que um único servidor possa desempenhar papel de mais de um servidor dedicado. Por meio então da virtualização, é possível fazer a consolidação de servidores, passando a executar sistemas operacionais com os serviços, que antes rodavam em servidores dedicados, em MVs e fazer a distribuição dessas MVs na menor quantidade possível de servidores. Embora a consolidação de servidores possa representar um risco em termos de disponibilidade dos serviços, já que se uma máquina física apresentar defeito várias MVs ficarão indisponíveis, como pode ser visto nos trabalhos de Dahbur et al. [DMT11] e Latif et al. [LAAA14], soluções baseadas em replicação podem reduzir esse efeito negativo.

Virtualização não é a única forma de consolidar servidores. Outra opção é realizar a consolidação no nível de aplicação, colocando aplicações que antes rodavam em servidores dedicados para rodar em um único servidor [Vog08]. No entanto, no nível de aplicação, passa a ser difícil garantir o isolamento entre as aplicações; a consolidação no nível do sistema operacional, ou com virtualização, passa a ser uma solução mais viável, uma vez que é mais fácil de garantir o isolamento. Todavia, para realizar a consolidação, é necessário fazer a alocação adequada dos servidores que vão rodar as MVs. Caso não haja a alocação adequada, pode haver um espalhamento das MVs entre os servidores, o que também ocasionaria uma baixa utilização dos servidores.

No processo de alocação das máquinas físicas é necessário considerar a demanda das MVs por recursos e a oferta desses recursos nos servidores, para que o desempenho dos serviços rodando nessas MVs não seja afetado em demasia, no caso, por exemplo, de um servidor ser sobrecarregado com muitas MVs. Além do mais, em nuvens, como a *HP Cloud* [hps13] e a *Amazon EC 2* [ama13], adota-se políticas onde, caso a MV fique indisponível por um determinado intervalo de tempo, haja descontos no pagamento do usuário, como uma espécie de multa para o provedor. Dessa forma também é necessário atender a maior parte possível dessa demanda. Não atendendo as demandas por recursos das MVs, há o que se chama de quebra de Acordo de Nível de Serviço (SLA – *Service-Level Agreement*).

A alocação de servidores físicos para atender as demandas das MVs pode ser tanto de forma estática como dinâmica. Na alocação estática sabe-se de antemão toda a demanda das MVs antes da execução de cada uma. Na dinâmica verifica-se periodicamente as demandas das MVs e, quando necessário, realiza-se a realocação de servidores para atender a demanda e desalocar servidores subutilizados. Dependendo da natureza do serviço oferecido, dificilmente sabe-se quanto de recurso uma MV pode necessitar no futuro, algo que não acontecia no passado em que os CPDs eram ambientes dedicados e particulares de grandes empresas. Dessa forma, na alocação estática MVs poderiam requisitar mais recursos do que realmente necessitam, fazendo com que houvesse espalhamento e subutilização dos servidores. Assim, a alocação dinâmica de MVs surge como uma abordagem mais adequada para CPDs onde o provedor desconhece os serviços oferecidos pelas MVs.

Alguns trabalhos como os de Murtazaev et al.[MO+11] e Berl et al. [BKB07] propuseram algoritmos de alocação dinâmica para resolver o problema da alocação modelando o problema como um problema de *bin packing* ou *vector packing*. A fim de verificar outra abordagem, esta dissertação

apresenta uma modelagem do problema como um problema da mochila e como solução apresenta um algoritmo que tem como base a heurística proposta por Toyoda [Toy75] para resolver o problema da mochila multidimensional. Apesar da heurística ser voltada para um número genérico de dimensões, apenas duas dimensões foram consideradas aqui para a solução: CPU e memória. Esses dois valores foram escolhidos por serem recursos geralmente escassos em um servidor.

Nesta dissertação, além da heurística proposta, foi desenvolvida um formulação baseada em programação linear para calcular uma estimativa do consumo de energia de um CPD. O objetivo dessa formulação é verificar a proximidade dos valores retornados pela heurística, para instâncias pequenas, em relação à solução ótima.

Além da heurística e da formulação baseada em programação linear, durante o desenvolvimento do trabalho, experimentos de medição também foram realizados com o objetivo de confirmar que servidores atuais apresentam as características relatadas pela indústria e pela academia com relação à baixa eficiência energética quando os mesmos encontram-se subutilizados. Os resultados dos experimentos confirmaram a ineficiência energética que justifica a importância de manter os servidores funcionando próximo aos 100% de sua capacidade computacional.

Experimentos de simulação realizados com traces disponibilizados pelo Google com informações a cerca da utilização de alguns de seus servidores confirmaram que a heurística proposta obteve ganhos em termos de consumo de energia quando comparado com um algoritmo bastante usado na literatura para a comparação de propostas de alocação de MVs (de 1 a 30%). Além disso, observou-se uma redução na migração dos servidores (em torno de 40%) e que as alocações ficaram próximas das ótimas em cenários pequenos que conseguiram ser executados com a formulação baseada em programação linear (cerca de 20% do ótimo).

É importante observar que além da subutilização dos servidores, equipamentos necessários para o funcionamento desses servidores também são responsáveis por uma parcela do alto consumo de energia de um CPD, como os equipamentos de refrigeração. Soluções para a gerência desses recursos não serão tratadas nessa dissertação. Algumas informações sobre propostas para lidar com esses recursos podem ser encontradas em [coo14] e [CFM+14].

1.1 Objetivos

O principal objetivo desta dissertação é:

- Propor uma heurística para realizar a consolidação de servidores.

Os objetivos secundários são:

- Desenvolver um modelo para estimar o consumo de energia em um CPD;
- Verificar como é afetado o desempenho de serviços executados em MVs;
- Verificar o consumo de energia de um servidor com e sem MVs.

1.2 Contribuições

A contribuições deste trabalho são as seguintes:

- Uma nova heurística para realizar a consolidação de servidores através da virtualização;
- Um simulador para avaliação de desempenho de algoritmos para alocação de MVs;
- Um modelo para estimar consumo de energia de um CPD;
- Uma formulação em programação linear que devolve a alocação ótima, consolidada, de um conjunto de MVs.

1.3 Organização do trabalho

Esta dissertação está organizada da seguinte forma: no Capítulo 2 são apresentados os conceitos necessários para o entendimento do trabalho. No Capítulo 3 são discutidos os resultados dos experimentos realizados envolvendo o consumo de energia de um servidor com virtualização e sem virtualização a fim de confirmar que computadores modernos podem tornar-se subutilizados quando a demanda por recursos computacionais é imprevisível. No Capítulo 4 é apresentada a principal contribuição da dissertação que é uma nova heurística para consolidação de MVs, e a formulação em programação linear. Neste capítulo também são apresentados os resultados dos experimentos de simulação realizados para avaliar os ganhos obtidos com a utilização da heurística. Os ganhos foram obtidos através da comparação das alocações da heurística com aquelas retornadas pela formulação em programação linear e com aquelas retornadas pelo algoritmo First Fit Decreasing (FFD), um algoritmo bastante utilizado na literatura para comparação de novas propostas de alocação de MVs. No Capítulo 5 são comentados trabalhos relacionados ao tópico desta dissertação. Por fim, o Capítulo 6 apresenta as conclusões obtidas a partir dos resultados dos experimentos realizados.

Capítulo 2

Conceitos

Neste capítulo são apresentados alguns conceitos necessários para a compreensão da dissertação. Inicialmente são apresentados conceitos relacionados à virtualização. Em seguida é feita uma breve descrição de programação linear e então é apresentado o problema da mochila. Finalizando o capítulo, é apresentado o problema de *bin packing* e feita uma comparação deste problema com o problema da mochila. Modelar o problema de alocação de MVs utilizando programação linear, o problema da mochila ou o problema de *bin packing* são algumas das várias formas de resolver este problema.

2.1 Virtualização

Virtualização consiste em fazer uma implementação em software de uma camada de hardware, ou mesmo um dispositivo inteiro, dando a ilusão para as camadas imediatamente acima de que estão rodando em um dispositivo real [SVLN13].

Embora a virtualização de recursos computacionais tenha se difundido bastante nos últimos anos em computadores pessoais e em servidores, ela começou de fato a ser utilizada na época dos *mainframes*, como é o caso da *Virtual Machine Facility/370* descrita no famoso artigo de Creasy [Cre81]. As soluções de virtualização de computadores pessoais apareceram como uma forma de rodar vários sistemas operacionais em uma mesma máquina física, solução útil em casos onde os usuários possuíam um sistema operacional principal mas precisavam rodar um aplicativo que possuía versão para outro sistema operacional. Hoje em dia a virtualização tem sido utilizada principalmente por grandes CPDs como forma de melhor utilizar os recursos computacionais disponíveis. Máquina com múltiplos núcleos podem por exemplo utilizar virtualização de modo a tirar um proveito maior de todos os núcleos. Cada sistema operacional pode nesse caso utilizar um núcleo inteiro.

As MVs consistem da virtualização de um hardware de servidor, ou seja, é um software que emula o comportamento de uma máquina física e faz com que o sistema operacional que esteja executando na máquina virtual comporte-se como se estivesse executando de fato em uma máquina real. Uma máquina virtual emula partes de uma máquina real para que isso seja possível. Essa emulação é realizada através dos chamados emuladores.

Emuladores não são a única forma de implementar a virtualização. Há também os *hypervisors*, que estão em uma camada abaixo das MVs, executando no sistema operacional do servidor ou mesmo sendo o sistema operacional do servidor. Esses *hypervisors* são responsáveis pelo gerenciamento das MVs, tanto no acesso a recursos como na criação ou destruição delas.

Emuladores e *hypervisors* não necessariamente aparecem separadamente em soluções para virtualização de computadores. É possível ter soluções baseadas em *hypervisors* que utilizem emulação também.

2.1.1 Emuladores

Os emuladores em computação permitem que um comportamento de um sistema seja reproduzido dentro de um outro sistema computacional. Eles podem tanto ser implementados em software quanto em hardware. A emulação pode ser utilizada por diversos motivos, desde para evitar o retrabalho de reescrever um software para um outro sistema determinado ou mesmo para fazer testes de um novo software em ambientes emulados e controlados antes de partir para testes no sistema final. Umas das emulações utilizadas na virtualização de computadores é a emulação de CPU, onde os *opcodes* de um arquitetura são traduzidos para *opcodes* de uma outra arquitetura.

A emulação é uma forma de executar diversas máquinas virtuais de maneira isolada e mesmo de arquiteturas diferentes sobre uma mesma máquina física. Na virtualização permitida pelo software de virtualização QEMU [qem13], por exemplo, não só a CPU é emulada como toda a arquitetura de um computador pessoal, como barramentos, memórias e placas de rede. A emulação de boa parte da camada de hardware permite o isolamento entre as máquinas virtuais.

Alguns exemplos de emuladores são QEMU [jpc14], Bochs e JPC.

2.1.2 Hypervisor

Um hypervisor é responsável pelo gerenciamento das máquinas virtuais em um servidor. Ele pode ser implementado tanto em software como em hardware. É por meio de um hypervisor que é realizada a criação, gerenciamento e destruição de máquinas virtuais. Ele Está localizado entre as máquinas virtuais e o sistema operacional do servidor, ou mesmo pode desempenhar o papel de um sistema operacional de um servidor e controlar o acesso das máquinas virtuais ao hardware.

Alguns exemplos de hypervisor são QEMU [qem13], VMWare VSphere [vsp13] e VirtualBox [vir13].

2.2 SLA

Define-se SLA como um acordo formal entre duas partes, o provedor de serviços e o consumidor [TB05]. No caso de CPDs oferecendo serviços na nuvem, esse acordo acontece entre os usuários e o provedor. O SLA é responsável por definir diversas características do serviço a ser oferecido ao usuário, que podem ser qualidade do serviço, prioridade desse serviços sobre outros, responsabilidades do provedor e do usuário, etc. O SLA facilita o acordo entre o provedor e o usuário, deixando claro que tipo de serviço o provedor deve fornecer e como o usuário deve agir.

No caso da computação na nuvem, o SLA pode ser definido como sendo a quantidade de cada recurso do CPD que uma máquina virtual deve pode usar. Como espaço em disco rígido é um recurso bem mais barato se comparado à memória RAM e à CPU, geralmente considera-se mais esses dois últimos recursos ao definir SLAs para a execução de aplicações na nuvem, como também considerado nos trabalhos de Murtazaev [MO+11] e Bobroff [BKB07]. É possível haver ainda detalhes no acordo relacionados com características da conexão via rede, como latência máxima permitida, métricas de desempenho de um determinado serviço e o tempo em que uma máquina virtual ficará disponível.

Mecanismos que fazem consolidação de servidores devem levar em consideração os requisitos dos SLAs a fim de evitar que eles sejam descumpridos [BKB07]. Em caso de quebra de SLA por parte do provedor, pode-se haver uma punição na forma de crédito adicional ao usuário.

2.3 Programação linear

Programação Matemática, de acordo com Strayer e Stryer [SS89], consiste em utilizar modelos matemáticos para modelar problemas reais. Esses modelos são compostos por variáveis e constantes e expressões que relacionam tais variáveis. Tem-se sempre uma função objetivo, cujas variáveis podem estar sujeitas a uma série de restrições; e busca-se a tupla de variáveis que fazem essa função ter valor ótimo.

Na Programação (Matemática) Linear, especificamente, as restrições são formadas por inequações e equações lineares, e a função objetiva também é linear. Na Programação Linear Inteira, uma variação da Programação Linear, busca-se limitar as variáveis a valores inteiros.

A formulação de um problema utilizando programação linear costuma ter a seguinte forma:

$$\text{função: } C^T X \tag{2.1}$$

$$\text{restrito a: } AX \leq B \tag{2.2}$$

Onde C e B são vetores de constantes, X é um vetor de variáveis e A uma matriz de constantes.

Existem algoritmos exatos para resolver problemas de programação linear, como o simplex. No caso de problemas de programação linear inteira, soluções podem ser encontradas por meio do método *branch-and-bound*. O *branch-and-bound* enumera todas as possíveis soluções para um determinado problema de otimização e percorre esse espaço de soluções como se estivesse percorrendo uma árvore. O método usa uma técnica de poda para eliminar a verificação desnecessária de possíveis soluções. No caso da Programação Linear Inteira, é possível ainda relaxar as variáveis para assumirem valores não inteiros e, desse modo, utilizar os Simplex para calcular possíveis soluções. Nesse caso algoritmos de pós-processamento devem ser empregados para aproximar os valores reais encontrados para valores inteiros que façam sentido para o problema sendo resolvido.

Vários softwares e bibliotecas já foram propostos para solucionar e modelar problemas de Programação Linear, como Matlab [lpm13], GLPK [lpg13] e CPLEX [lpc13]. O CPLEX, em particular foi utilizado para a implementação da formulação em programação linear que será apresentado no Capítulo 4

2.4 Problema da mochila multidimensional

O Problema da Mochila Multidimensional (PMM), como pode ser visto no livro de Kellerer, Pferschy e Pisinger [KPP04], é uma variação do conhecido Problema da Mochila (PM), considerando agora mais de uma restrição. No PM há diversos objetos, cada um com um peso e um valor (custo, preço, etc...) pré-definido. Esses objetos devem ser colocados em uma mochila de modo a maximizar

a soma dos valores de todos os objetivos dentro da mochila. A restrição nesse problema é que a mochila possui um limite de peso que deve ser respeitado.

A generalização do PM com mais de uma restrição é chamada de d-PM, onde d é o número de restrições utilizadas. Nesse problema definimos $N = \{1, 2, 3, 4, 5, ..., n\}$, onde $n \geq 1 \in \mathbb{Z}$, como sendo um conjunto de elementos que se quer selecionar para colocar em uma mochila M. Cada elemento de N tem uma d-tupla $w_i = (w_{i1}, w_{i2}, ..., w_{id})$ e um ganho p_i associados, para $1 \leq i \leq n$. No entanto, M não pode levar mais do que o valor total de $(c_1, c_2, ..., c_d)$ e deve-se fazer uma escolha de objetos de forma a maximizar o ganho total. Abaixo estão a função objetivo e as restrições do PMM:

$$\text{maximizar: } \sum_{j=1}^{n} p_j x_j \tag{2.3}$$

$$\text{restrição: } \forall i \leq d \sum_{j=1}^{n} w_{ji} x_j \leq c_i, i = 1, 2, 3, ..., n. \tag{2.4}$$

$$x_j \in \{0, 1\}, j = 1, ..., n. \tag{2.5}$$

As variáveis x_j, para $j \in N$, definem quais objetos na solução estarão na mochila. Caso o objeto j esteja, x_j assume 1. Caso não esteja, assume 0.

Várias abordagens já foram propostas para resolver o PMM. Apesar de ser considerado um problema NP-Difícil, há algoritmos baseados em programação dinâmica que são considerados pseudo-polinomais. Fréville [Fré04] faz uma revisão dos métodos propostos para resolver o PMM. São apresentados algoritmos exatos, como programação dinâmica e *branch-and-bound*, que podem ser interessantes dependendo do número de restrições.

Como pode ser visto no Capítulo 4, o problema da alocação de servidores pode ser modelado como um problema da mochila. No presente trabalho, os itens são as máquinas virtuais e a mochila um servidor. As restrições podem ser dada em funções de valores como uso de CPU ou uso de memória das máquinas virtuais e CPU e memória disponíveis pelo servidor.

2.4.1 Heurística de Toyoda

Pelo fato do PMM ser um problema *NP-Difícil*, seus algoritmos exatos rapidamente explodem no tempo à medida que as instâncias do problema crescem. Motivado por isso, Toyoda [Toy75] propôs uma heurística voltada para o PMM capaz de encontrar soluções aproximadas de grandes instâncias do problema com uma baixa complexidade temporal, mas que são, no entanto, próximas ao ótimo, segundo o autor. Modelando o problema da alocação de servidores para máquinas virtuais como mochila, é possível haver cenários de grandes instâncias do PMM, o que faz com que a proposta de Toyoda seja adequada para o problema, já que pode ser necessário um baixo tempo de resposta para a soluçã, com também uma solução próxima à ótima.

Visando reduzir a complexidade temporal, a heurística proposta não usa enumeração. No lugar da enumeração o autor adota o gradiente efetivo (em uma tradução livre para *effective gradient*), que é um valor relativo para cada item na mochila. Esse gradiente é calculado usando os chamados

vetores de penalidade (*penalty vectors*), termo usado para denominar também as dimensões de cada item ou a tupla de valores associada a cada item. Os itens então são ordenados e com base nessa ordenação são escolhidos os itens que vão para mochila.

Para propor a heurística, o autor reescreve o PMM como:

$$\text{maximizar: } \sum_{j=1}^{n} p_j x_j \tag{2.6}$$

$$\text{restrição: } \forall i \leq d \sum_{j=1}^{n} f_{ji} x_j \leq 1, i = 1, 2, 3, ..., n. \tag{2.7}$$

$$x_j \in \{0, 1\}, j = 1, ..., n. \tag{2.8}$$

$$f_{ji} = w_{ji}/c_i, i = 1, ..., d \text{ e } j = 1, ..., n. \tag{2.9}$$

Dessa forma, os vetores de penalidade de cada item j passam a ser dados por $P_j = (f_{j1}, f_{j2}, f_{j3}, ..., f_{jd})$. Tendo mais de uma dimensão, há uma dificuldade em saber de fato quanto de recurso da mochila um item vai utilizar, o quanto de recurso da mochila já está sendo utilizado. Poderia ser usado por exemplo o módulo do vetor, mas isso representaria bem o quanto o item usaria da mochila. A ideia então é utilizar o produto interno do vetor de penalidade com o vetor de uso da mochila P_u. O vetor de uso da mochila representa o quanto de cada recurso já está sendo utilizado, o produto interno então define a penalidade daquele item ir para a mochila. Para definir se o item deve ir ou não para a mochila, o valor utilizado é dado por $G_j = \frac{p_j}{U_j}$ (gradiente efetivo), onde p_j é o ganho associado ao item e $U_j = \frac{P_j P_u}{|P_u|}$. Assim, quanto maior o valor do ganho associado ao item (menor penalidade), maior chance o item tem de ir para a mochila; e quanto maior a penalidade, menor a chance do item ir para a mochila. O item, que na iteração tem maior valor para G_j, vai para a mochila. Note que P_u vai sendo alterado a medida que algoritmo itera sobre os itens para escolher qual o próximo item a ir para a mochila.

2.5 Problema do *bin packing*

O Problema de *bin packing* bidimensional consiste, de forma semelhante ao da Mochila, em mapear um determinado conjunto de objetos em um conjunto de caixas. No entanto, no *bin packing* o objetivo é fazer o mapeamento de tal forma que se consiga colocar o maior número possível de objetos no menor número de caixas. Também de forma semelhante ao Problema da Mochila Multidimensional, no *bin packing* os objetos e as caixas possuem dimensões. Cada caixa pode conter objetos desde que esses objetos possam ser organizados dentro do volume da caixa.

De modo geral, no problema do *bin packing* há um conjunto de n itens retangulares $j \in J = \{1, ..., n\}$, cada um tendo comprimento w_j e altura h_j. É dado também um conjunto infinito de caixas idênticas de comprimento W e H. O problema consiste em alocar os itens para o menor número possível de caixas. Como dito em Lodi, Martello e Monaci [LMM02], o *bin packing* bidimensional é um problema *NP-Difícil*. Ainda no *survey* desses três autores, são apresentadas algumas heurísticas

para o *bin packing* bidimensional, algumas delas usadas como base para alguns algoritmos propostos de alocação de servidores que serão resumidos no Capítulo 5.

O *bin packing* é utilizado é utilizados como base para resolver o problema da alocação em vários trabalhos, como no trabalho do Murtazaev [MO+11]

Capítulo 3

Motivação

Para realizar a consolidação de servidores por meio da virtualização, é necessário procurar fazer o mapeamento das máquinas virtuais para o servidores de forma que não afete os serviços oferecidos pelas máquinas virtuais, ao mesmo tempo que os servidores utilizados por essas máquinas virtuais tenham maior parte possível dos seus recursos utilizada. Dessa forma, a fim de verificar o comportamento e o desempenho de um servidor em vários cenários de estresse no uso de recursos por máquinas virtuais, alguns experimentos foram realizados.

Além dos experimentos relacionados com desempenho com a virtualização, foram realizados experimentos para mostrar o consumo de energia com ou sem a virtualização. Por mais que a virtualização promova alguma degradação do desempenho dos serviços oferecidos, é possível ver nos gráficos mais adiante de consumo de energia e desempenho que com a virtualização é possível ter uma redução nos custos quando leva-se em consideração o valor para manter serviços em máquinas virtuais em relação a serviços rodando em servidores dedicados.

3.1 Ambiente dos experimentos

Os experimentos foram executados com a emulador QEMU com KVM, tendo como hypervisor a libvirt.

3.1.1 Configuração da máquina física

- CPU: Intel i7-3610QM (4 núcleos mais 4 núcleos virtuais) (2.3 GHz até 3.3 GHz)
- Memória RAM: 8 GB
- Hard disk: 750 GB - 5.400rpm
- Sistema operacional: Linux Fedora 18 x64

3.1.2 Configuração das máquinas virtuais

- vCPU: 1
- Memória: 512 MB
- Hard disk: 1 GB

- Arquitetura emulada: i386
- Sistema operacional: Ubuntu 12.04.2 LTS

3.1.3 Benchmark com sysbench

A ferramenta utilizada para realizar os experimentos de estresse com as máquinas virtuais foi o sysbench na versão 0.3.1 [sys13]. O sysbench foi criado com o propósito de avaliar alguns parâmetros de um sistema operacional ou de um servidor que venha a ser utilizado de forma intensa como servidor para bancos de dados. Com ele é possível pegar parâmetros relacionados a desempenho em operações de entrada e saída com arquivos, desempenho do escalonador do sistema, alocação e uso de memória, ou mesmo desempenho da CPU.

Os testes do sysbench que envolvem CPU são realizados com um algoritmo de força bruta para verificação de números primos. Dessa forma não se verifica aqui operações de ponto flutuante por exemplo.

3.2 Desempenho com virtualização

Foram realizados experimentos para verificar o detrimento no desempenho em serviços do tipo *IO-bound* e *CPU-bound* quando executados em máquinas virtuais em relação a um servidor dedicado. Até 12 máquinas virtuais foram executadas de forma simultânea. Essas máquinas virtuais possuem a mesma configuração explicitada na Subseção 3.1.2. Como o servidor utilizado para executar as máquinas virtuais possuia somente 8 GB de RAM, foi possível executar somente até 12 máquinas virtuais com 512 MB cada de RAM, o que permitia também que outros processos necessários ao uso do servidor fossem também executados. No entanto, com 12 máquinas virtuais, com uma vCPU cada, já é possível verificar como o desempenho pode ser afetado com a virtualização.

3.2.1 CPU

Os experimentos envolvendo a CPU foram realizados com o sysbench com os seguintes parâmetros: `"--test=cpu --cpu-max-prime=50000 run"`. Analisando o gráfico na Figura 3.1, como o servidor tem 4 cores de fato e cada máquina virtual tem uma vCPU, o tempo médio de execução do sysbench no servidor e nas máquinas virtuais pouco varia até 4 máquinas virtuais. Após 4 máquinas virtuais, o tempo médio necessário para a execução dos experimentos sysbench começa a crescer de forma linear. Ou seja, começa haver um aumento considerável no tempo de execução quando o número de vCPUs utilizadas de forma intensa é maior que o número de núcleos disponíveis pelo servidor. Assim é necessário haver um casamento no número de vCPUs de fato utilizadas com o número de núcleos disponibilizados pelo servidor, evitando assim o subuso dos núcleos ou mesmo afetar o desempenho dos serviços oferecidos pelas máquinas virtuais.

3.2.2 Memória RAM

Utilizando também os sysbench foi verificado como o acesso à memória RAM pode ser afetado com o uso de máquinas virtuais; o sysbench aqui foi executado com os seguintes parâmetros: `"--test=memory --memory-block-size=1M --memory-total-size=10G run"`.

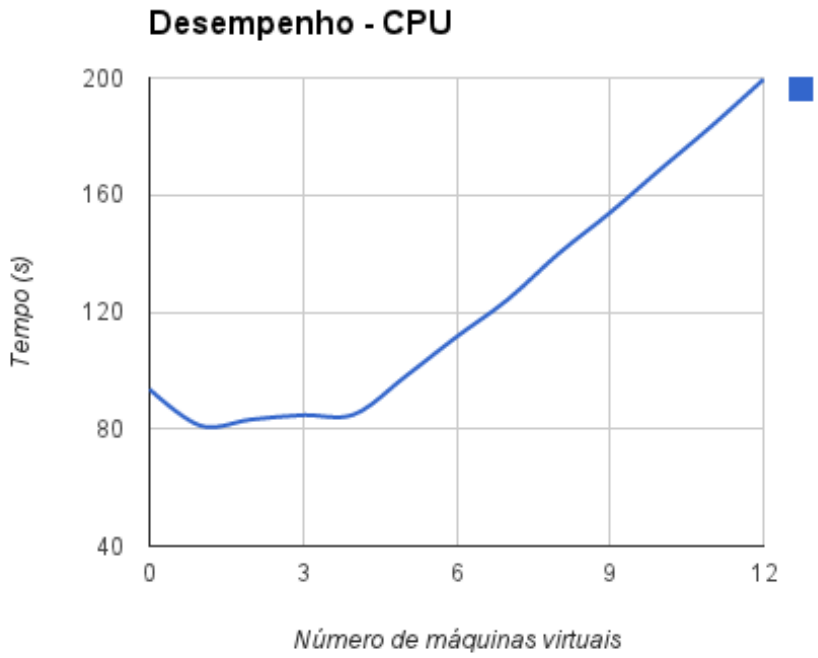

Figura 3.1: *Experimentos envolvendo o uso de CPU.*

Quando o acesso à memória RAM passa a ser disputado também há um redução na vazão média dos dados que podem ser lidos ou escritos nela. É possível ver a curva no gráfico da Figura 3.2 mostrando uma queda na vazão média dos dados à medida que o número de máquinas virtuais vai crescendo. Além de considerar a queda na vazão da escrita e leitura quando se aumenta o número de máquinas virtuais em um servidor, é preciso considerar deixar algum espaço na memória RAM reservado para os processos necessários para o funcionamento de um servidor, como hypervisor por exemplo.

3.3 Consumo de energia

Também foram realizados alguns experimentos com a finalidade de verificar o consumo de energia quando se utiliza ou não a virtualização. Nas subseções adiante são mostrados os consumos de energia em diversos cenários de estresse envolvendo o uso da virtualização em um servidor. Dos resultados é possível notar que a virtualização causa um pequeno aumento no consumo da energia quando comparada ao uso de um servidor dedicado, no entato é possível notar uma redução significativa quando se compara o uso de várias máquinas virtuais e vários servidores dedicados.

3.3.1 Ambiente

Para realizar os experimentos foram utilizados servidor e máquinas virtuais com as mesmas configurações descritas nas subseções 3.1.2 e 3.1.1. Já para fazer as medições do consumo de energia foi utilizado o wattímetro digital, que pode ser visto na Figura 3.3.

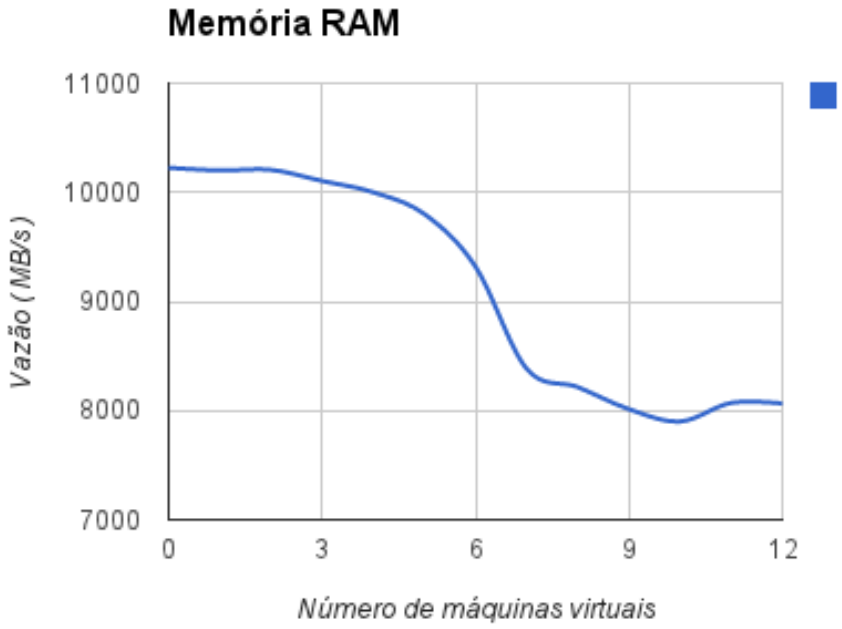

Figura 3.2: *Experimentos envolvendo o uso de memória RAM.*

3.3.2 Consumo de energia de um servidor

Foi realizado um experimento para verificar o consumo de energia do servidor sem qualquer máquina virtual, apenas com um processo sysbench com os parâmetros:

```
"--test=cpu --cpu-max-prime=50000 run"
```

É possível ver na Figura 3.4 que o consumo é próximo ao de uma máquina virtual executando o sysbench com os mesmos parâmetros na Figura 3.5.

3.3.3 Consumo de energia de uma máquina virtual

Para verificar como a virtualização pode afetar o consumo de energia em um servidor, foram realizadas medições de um servidor com uma máquina virtual executando o sysbench com os mesmos parâmetros de execução dados na subseção anterior. No gráfico da Figura 3.5 é possível ver o consumo em função do tempo.

Pouco antes no instante de 30s dá-se início o processo de boot da máquina virtual, onde há um pico no consumo de energia no servidor. É possível ver que o boot da máquina virtual é processo custoso em termos energéticos. Aproximadamente no instante de 60s o sysbench é executado e permanece em execução até o instante de 150s aproximandamente, onde dá-se início ao processo de desligamento da máquina virtual, que por sua vez pouco exige do processamento do servidor.

Na Figura 3.6 há um gráfico comparativo de consumo com um servidor executando sysbench sem virtualização e outro com uma máquina virtual executando o sysbench. Nota-se que a virtualização provoca um aumento de 1 a 5 W na potência.

Se considerar dois servidores com a mesma configuração aqui utilizada e duas máquinas virtuais em um único servidor executando sysbench com esses parâmetros, é possível ver por exemplo que as máquinas virtuais provocam um menor consumo de energia. Na Figura 3.7 tem-se o comparativo entre o consumo duplicado do experimento com um servidor sem virtualização e um servidor com

Figura 3.3: *Wattímetro utilizado nos experimentos.*

duas máquinas virtuais executando sysbench. É possível notar que o diferença considerável entre os dois consumos.

3.3.4 Consumo de energia de um servidor com mais de uma máquina virtual

À medida que se aumenta o número de máquinas virtuais em um servidor é esperado que também haja um aumento no consumo de energia do servidor, já que passa a exigir maior poder de processamento. A fim de verificar isso, foram realizadas diversas medições no servidor executando até 12 máquinas virtuais, como é possível notar no gráfico da Figura 3.8.

No gráfico da Figura 3.8, até aproximadamente o instante 50s, é possível o consumo provocado pelo processo de *boot* nas máquinas virtuais. Logo depois é possível ver o acréscimo no consumo de energia no servidor provocado pelo número de máquinas virtuais. Como o servidor possui 4 cores, o consumo de energia, após 4 máquinas virtuais, permanece aproximadamente o mesmo.

Também para verificar o acréscimo no consumo de energia causado pelo virtualização comparou-se a execução do sysbench com 8 threads de execução em um servidor sem máquina virtual e a execução de sysbench em 8 máquinas virtuais em um servidor de forma simultânea. O gráfico comparativo está na Figura 3.9. Dos gráficos nas figuras 3.6 e 3.9, é possível ver que há somente um pequeno acréscimo no consumo de energia causado pela virtualização.

3.3.5 Consumo de energia por uso de CPU

Pelo gráficos das subseções anteriores é possível notar que o aumento de uso de CPU provoca de fato um aumento no consumo de energia do servidor. Para verificar com que rapidez o consumo aumenta, foi construído o gráfico da Figura 3.10. No gráfico verde tem-se as medições de consumo em função da porcentagem de uso da CPU, já no vermelho tem-se o gráfico reconstruído usando duas regressões lineares. O valor da porcentagem é dado pela soma do uso dos cores da CPU. Como o servidor usado nos experimentos aqui tem 4 núcleos, o valor vai até 400%, já que os valores de uso dos núcleos são somados. Para construir esse experimento foi utilizado o *cpulimit* e *sysbench*.

É possível ver que a rapidez com que o consumo de energia cresce diminui a partir do uso do

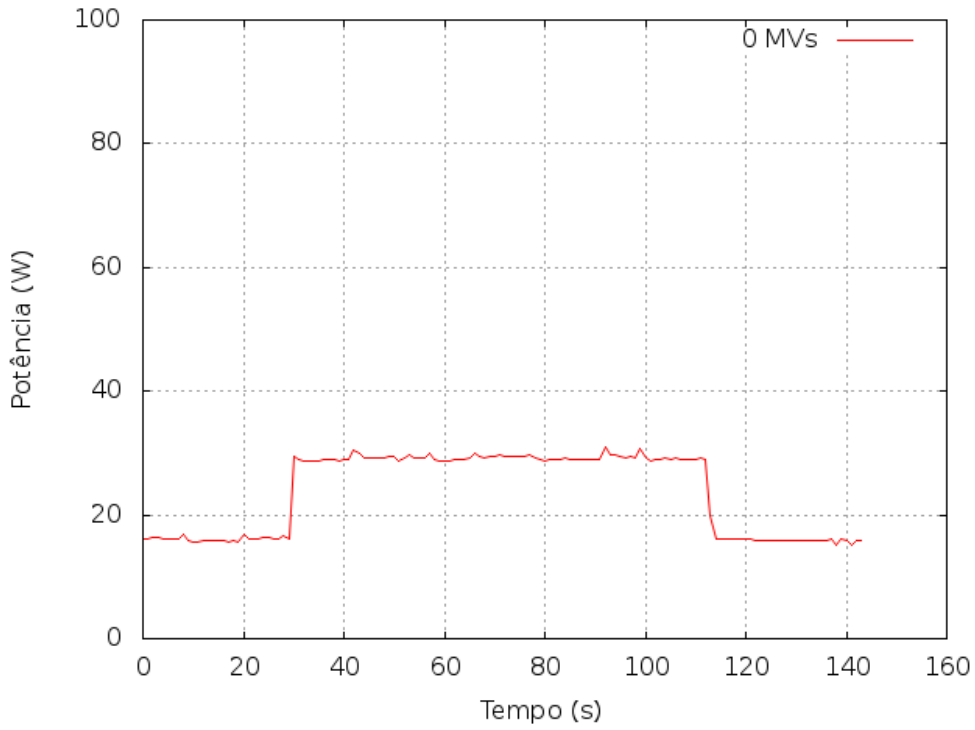

Figura 3.4: *Gráfico do consumo de um processamento em um servidor sem MV.*

segundo núcleo. Considerando duas máquinas virtuais, cada usando 100% de dois cores, é possível notar que o consumo de energia provocado com essas duas máquinas virtuais seria menor do que dois servidores usando somente um core. O que corrobora o resultado no gráfico 3.7. De forma análoga é possível pensar para um número maior de cores e máquinas virtuais. É importante ressaltar que aqui está sendo estressado é somente a CPU do servidor.

Essa mudança na função de consumo ocorre porque o processador aqui utilizado possui a tecnologia chamada *SpeedStep* [spe14], que muda a frequência do relógio e alimentação da CPU conforme o seu uso. Isso faz com que então, como se pode ver no gráfico, a função consumo de energia varie quanto a rapidez no crescimento.

No trabalho de Barroso e Holzle [BH07] também há um estudo do consumo de energia de acordo com o uso da CPU, mostra que entre o 10% e 50% uma máquina física comum tem uma baixa eficiência energética. Esse resultado pode ser visto também do gráfico da Figura 3.10, onde com um núcleo o consumo de energia cresce com maior rapidez que quando se passa a usar dois cores ou mais. Essa baixa eficiência encontrada por Barroso e Holzle é explicada também pelo subuso de outras partes do hardware. O problema é que, ainda de acordo com Barroso e Holzle, a maioria dos servidores em um CPD comum está nessa faixa de uso, entre 10% e 50%.

3.4 Consumo de energia de acordo com o uso de memória

Foram também realizados alguns experimentos para verificar o consumo de energia em função do uso percentual de memória. O benchmark [1] desenvolvido para o experimento foi construido de forma a forçar o *page fault* e assim gerar maior tráfego na hierarquia de memória.

[1] https://github.com/maxrosan/Balancer-Simulator/blob/master/experiments/memory/memory.c

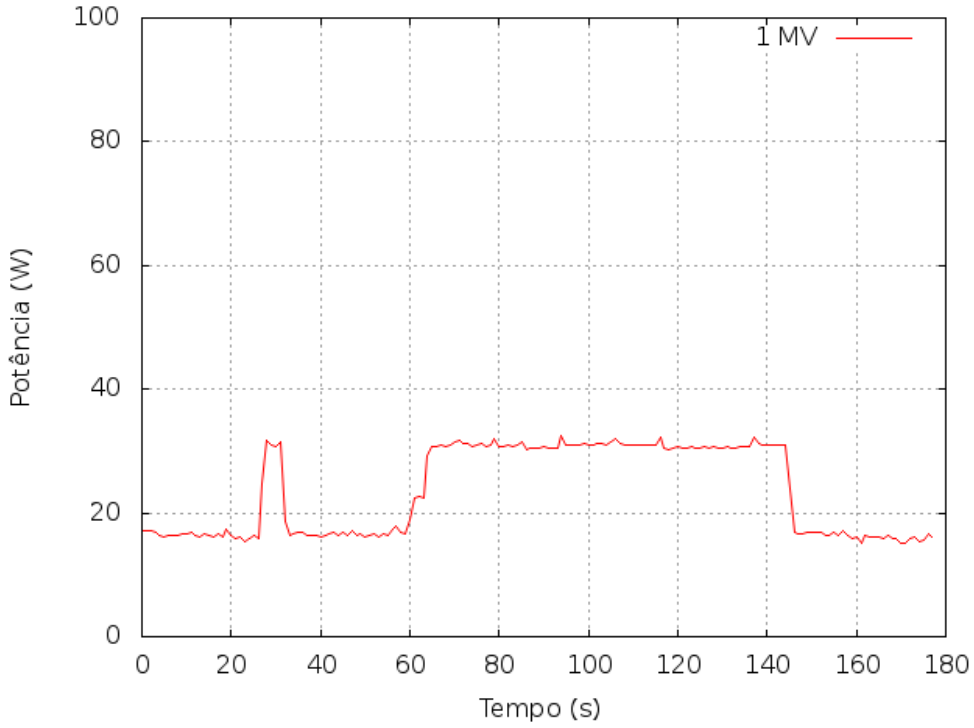

Figura 3.5: *Consumo de energia com 1 máquina virtual.*

Como pode ser notado no gráfico comparativo na Figura 3.11, o percentual de uso da memória pouco influencia no consumo da memória. No trabalho de Hicks, Walnock e Owens [HWO97], pode ser visto que a forma como a memória é utilizada (padrões de acesso e leitura) influencia mais no consumo de energia do que o percentual de uso dela. Como o benchmark implementado aqui somente provoca *page fault*, pouco é possível ver diferença no consumo da energia.

3.5 Conclusão

A partir dos experimentos realizados é possível ver que o uso da virtualização, pelo menos com o QEMU/KVM, causa um acréscimo no tempo de resposta dos serviços tanto do tipo *CPU-bound* como *IO-bound*. Um número excessivo de máquinas virtuais em um servidor pode até mesmo degradar bastante a qualidade de um serviço oferecido por qualquer uma dessas máquinas virtuais, como é possível ver nos gráficos das subseções 3.2.1 e 3.2.2.

Pelos experimentos realizados e discutidos na Seção 3.3, é possível ver que através da virtualização é possível reduzir o consumo de energia, passando os serviços antes executados em servidores dedicados para máquinas virtuais. Como é possível na redução no gráfico da Figura 3.7, comparando o consumo no uso de dois servidores idênticos e duas máquinas virtuais rodando sysbench.

Apesar da camada de virtualização poder provocar uma pequena queda no tempo de resposta dos serviços, a redução no consumo de energia com o uso da virtualização faz com que se torne uma proposta interessante em termos de custo. Com um mapeamento adequado das máquinas virtuais para o servidores é possível minimizar essa queda do desempenho dos serviços, ou seja, alinhando a demanda das máquinas virtuais com a oferta de recursos dos servidores é possível ter somente o pequeno *overhead* no tempo de resposta causado pela camada de virtualização, além da economia no consumo de energia. Motivado por esse problema do mapeamento de máquinas virtuais nos

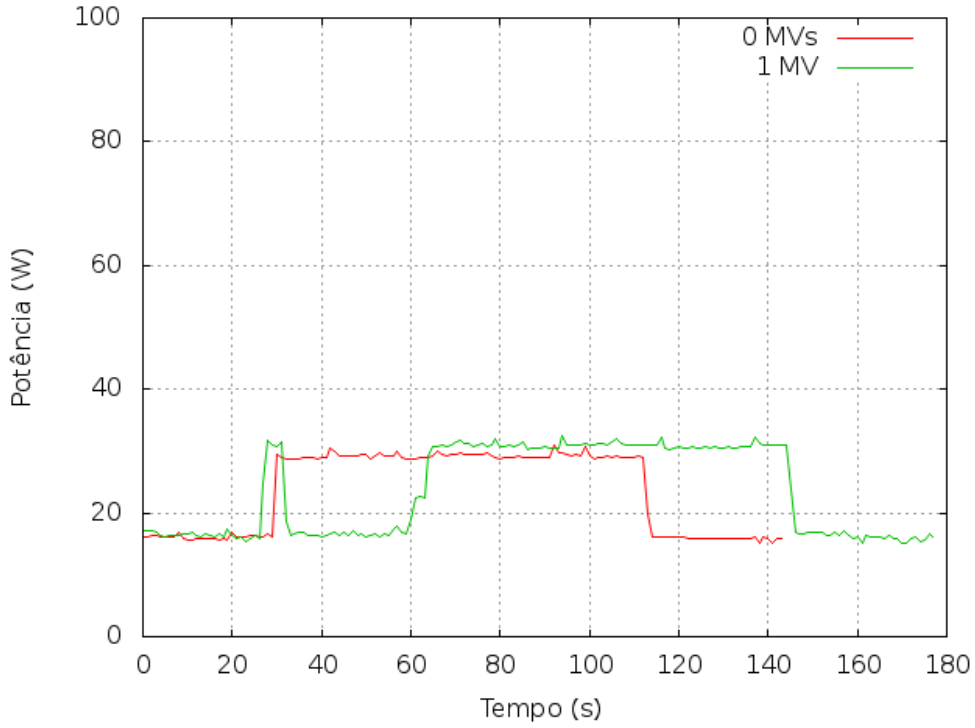

Figura 3.6: *Gráfico comparativo do consumo no processamento em um servidor sem MV e um servidor com MV.*

servidores, o presente trabalho propõe um algoritmo para resolvê-lo.

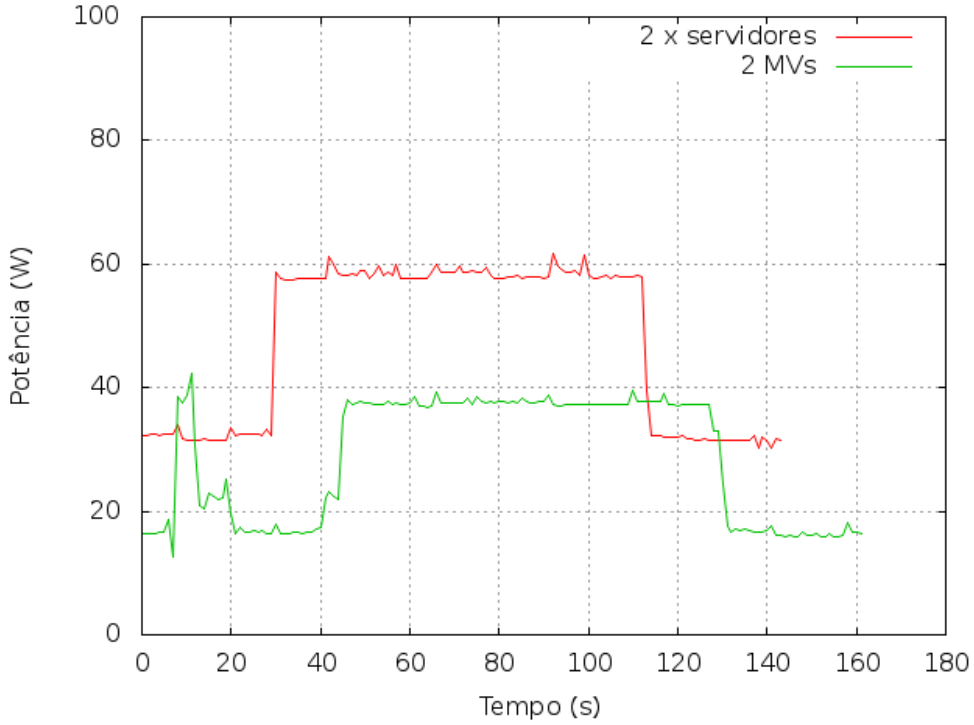

Figura 3.7: *Gráfico comparativo do consumo no processamento em um servidor sem MV e um servidor com MV.*

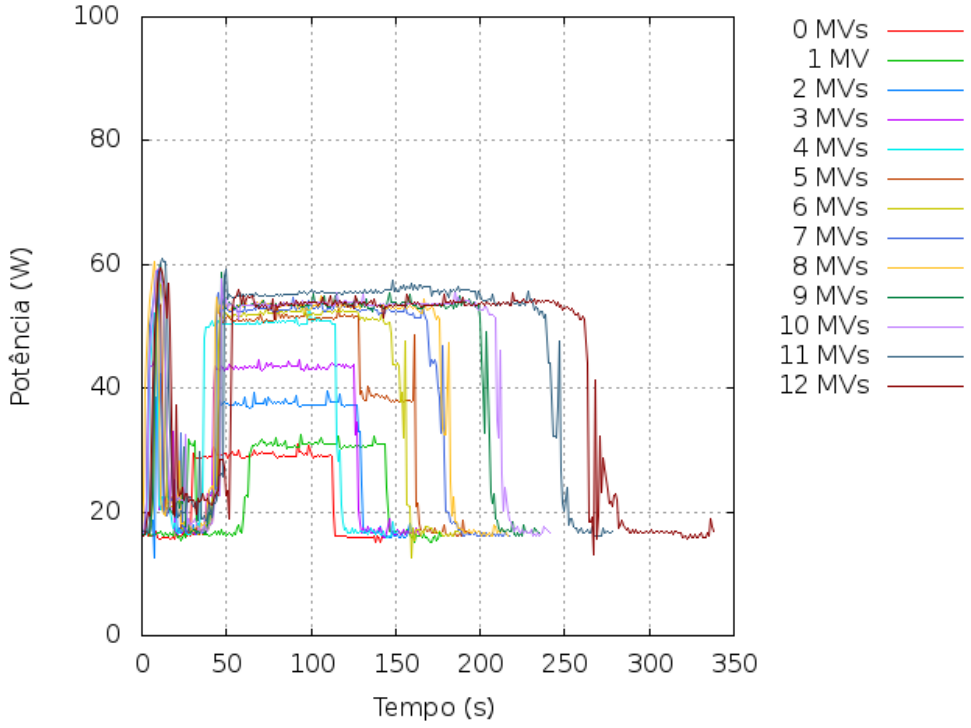

Figura 3.8: *Consumo de energia com até 12 máquinas virtuais.*

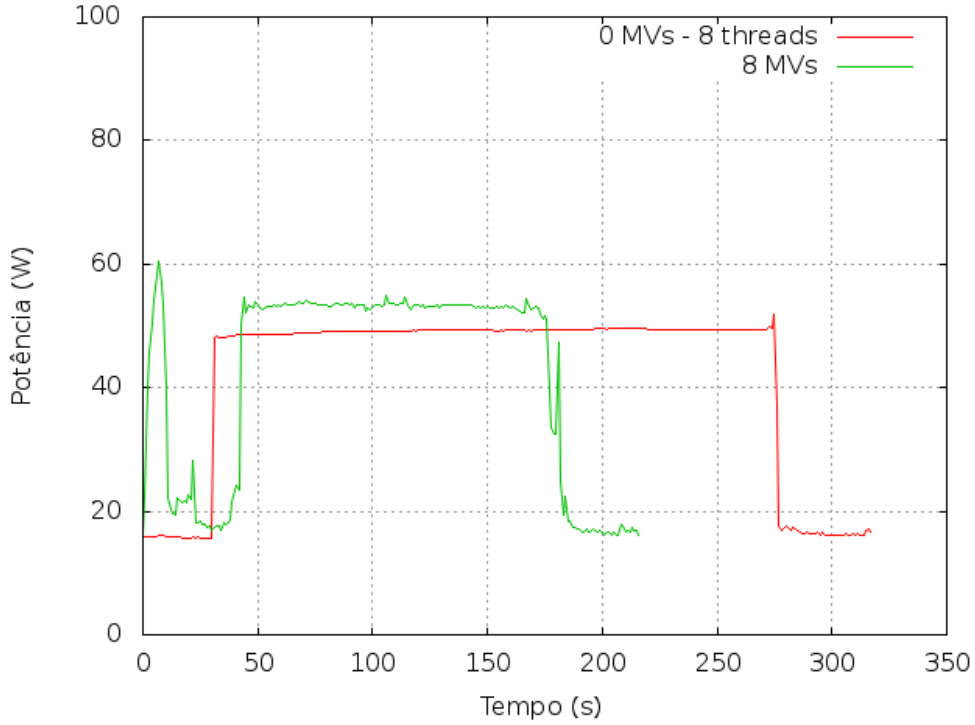

Figura 3.9: *Consumo de energia do servidor sem MVs e com 8 threads de execução e do servidor com 8 máquinas virtuais.*

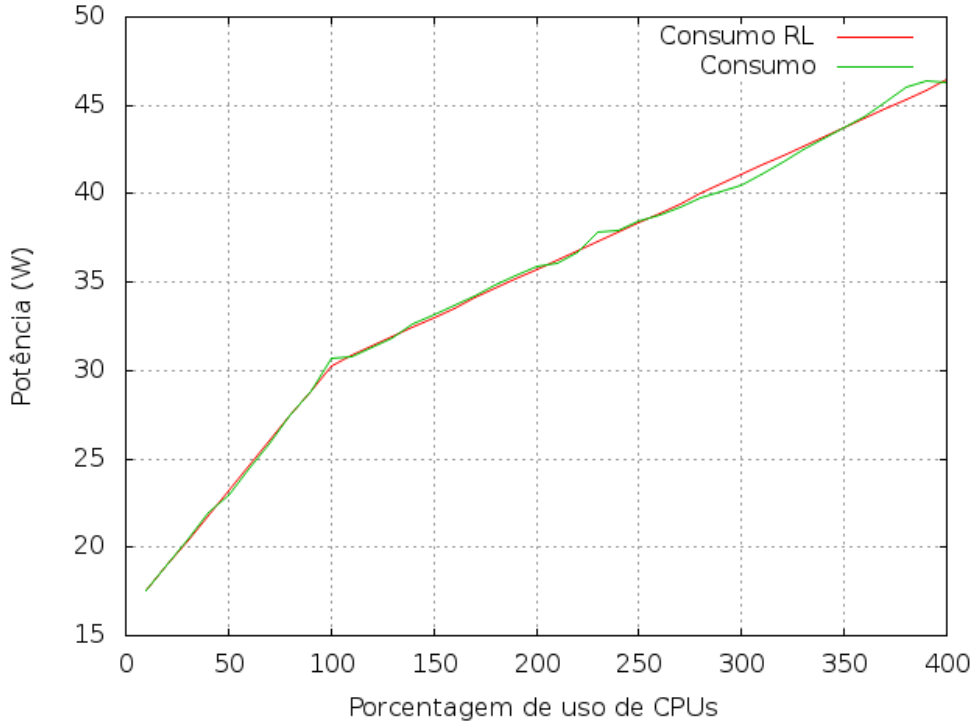

Figura 3.10: *Comparativo.*

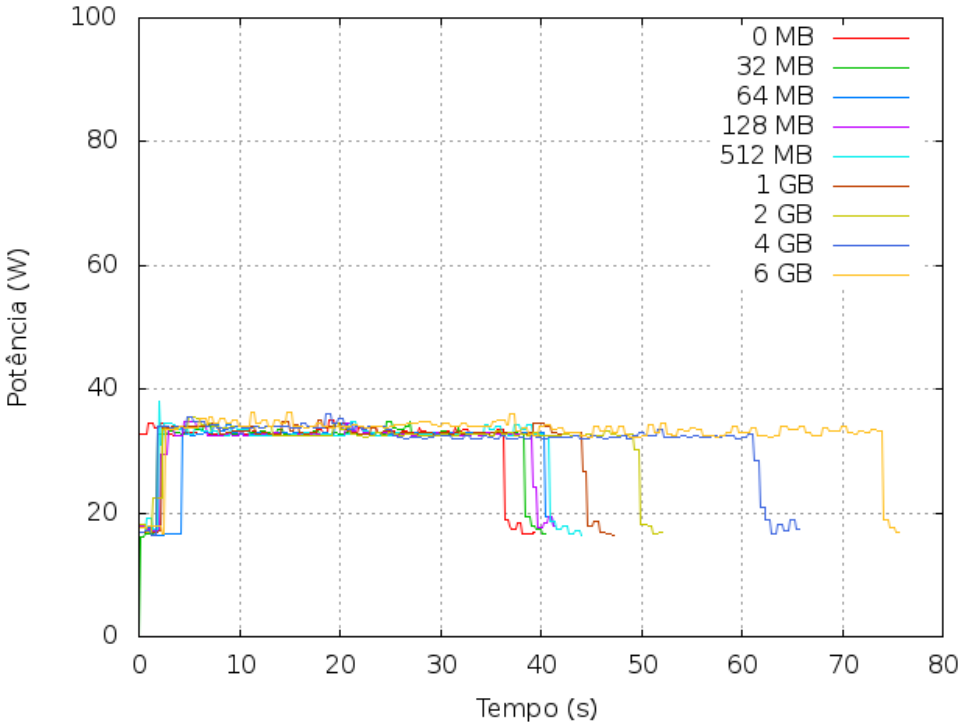

Figura 3.11: *Comparativo do consumo de energia de acordo com uso de memória.*

Capítulo 4

Proposta de algoritmo para alocação

Em uma alocação de servidores, objetivando a consolidação de servidores, busca-se alocar os servidores para as máquinas virtuais de forma a evitar o espalhamento dessas máquinas virtuais, e assim evitar o mau uso dos recursos computacionais ofertados, fazendo assim também um uso mais eficiente de energia. Procura-se também atender a demanda por recursos das máquinas virtuais utilizando a menor quantidade possível de servidores.

Como dito no Capítulo 1, o algoritmo de alocação aqui proposto é dinâmico. Os algoritmos dinâmicos periodicamente realizam o remapeamento das máquinas virtuais para os servidores sempre objetivando reduzir o número de servidores com máquinas virtuais executando, e assim realizar a consolidação. Por fazer esse remapeamento periódico, pode ocasionar migrações entre servidores, o que pode ser uma desvantagem em relação a abordagens estáticas, já que as migrações podem ser processos custosos. No entanto, como pode ser visto no trabalho de Clark et al. [CFH+05], já é possível hoje realizar a *live migration*, que permite que a máquina virtual continue a executar durante parte da migração, reduzindo assim o *downtime*. Isso também permite a adoção de abordagens dinâmicas, como a aqui proposta.

A fim de propor um algoritmo para fazer a alocação de servidores, buscando um uso mais eficiente da energia e dos recursos, o problema da consolidação foi inicialmente modelado como um Problema de Programação Inteira (PI), como pode ser visto na Seção 4.1. Pelo fato de ser Programação Inteira, os algoritmos para busca da solução exata podem ter complexidade exponencial dependendo das valores de entrada. Buscou-se então uma heurística para resolver o problema da alocação mais eficiente em termos energéticos. A heurística adotada para ser utilizada como base foi a heurística proposta por Toyoda [Toy75], que por sua vez foi proposta para resolver o Problema da Mochila.

Para utilizar a heurística de Toyoda, o problema da alocação de servidores foi remodelado usando como base o Problema da Mochila Multidimensional. Cada servidor constitui uma mochila e as máquinas virtuais formam os objetos que vão para a mochila. As restrições de recursos como CPU e memória de cada servidor formam as restrições das mochilas. Já que é desejado maximizar o uso de um servidor, o ganho associado a cada máquina virtual é dado de acordo com o uso dos recursos que a máquina virtual faz. Para calcular o ganho, maior valor entre uso de CPU e uso de memória é pego e então esse valor é associado ao ganho da máquina virtual enquanto objeto em uma mochila; espera-se, por meio desse ganho, maximizar o uso de um servidor. Por meio dessa maximização do uso do servidor procura-se colocar o maior número possível de máquinas virtuais no menor número possível de servidores, uma vez que foi constatado pelos experimentos no Capítulo 3 que um servidor pode ter seu uso mais eficiente quando seus recursos, como CPU por exemplo,

estão sendo utilizados próximos a suas capacidades máximas.

A heurística de Toyoda foi escolhido por sua complexidade temporal, por ter um tempo de resposta pequeno para instâncias grandes do Problema da Mochila. Diferentemente de algumas outras propostas para resolver o problema da mochila, a heurística proposta por Toyoda não usa enumeração, ou seja, não enumera todos os possíveis subconjuntos de itens para então escolher um desses subconjuntos como solução. Em vez de usar enumeração, a ideia por trás da heurística é usar o gradiente efetivo, também formulado em outro trabalho do Toyoda [ST68].

Foi necessário, a fim de otimizar os resultados retornados pela heurística de Toyoda, realizar alguns passos adicionais antes e depois de executar o algoritmo. Antes de aplicar a heurística para resolver o problema da mochila em cada servidor, é necessário realizar uma ordenação na lista de servidores e na lista de máquinas virtuais. Tal ordenação faz-se necessária para que os servidores com mais recursos computacionais possam executar o maior número possível de máquinas virtuais, e os servidores com menos recursos possam ser desligados.

Após realizar a ordenação dos servidores e das máquinas virtuais e fazer a alocação com a heurística de Toyoda para cada servidor, pode haver máquinas virtuais que não foram mapeadas para nenhum servidor. Isso pode ocorrer pelo fato do algoritmo dividir a instâncias em várias outras para fazer cálculos em paralelos, assim pode haver casos onde as máquinas virtuais que não foram mapeadas ainda possam ser executadas em algum servidor sem causar quebra de SLA. Mesmo que algumas das máquinas virtuais restantes possam ser executadas somente com quebra de SLA, tem-se que procurar um servidor para cada uma dessa máquina virtual de tal forma que essas máquinas virtuais possam atender maior parte de sua demanda por recursos.

Para comparar o resultado da heurística aqui proposta, foi escolhido o First Fit Decreasing (FFD), que é uma heurística utilizada para resolver o problema do Bin Packing, que por sua vez é utilizado como base para modelar e resolver o problema da consolidação de servidores.

As seções que se seguem contém a descrição dos algoritmos propostos para a alocação com a heurística de Toyoda e o mapeamento das máquinas virtuais não mapeadas pela a heurística, assim como também os resultados dos experimentos envolvendo a implementação do algoritmo.

4.1 Modelagem com Programação Linear

Antes de apresentar a heurística proposta para a consolidação, aqui será apresentado o modelo usado para estimar o consumo de energia de um CPD e comparar os resultados da heurística proposta com um algoritmo exato.

Para modelar o consumo de energia em um ambiente semelhante a um CPD, é necessário considerar todo dispositivo do CPD que faz algum consumo de energia elétrica. No entanto, diante da complexidade e da heterogeniedade em termos de equipamento que há entre os diferentes CPDs, um modelo de consumo genérico que seja aplicável a qualquer CPD torna-se complicado de ser construído. Apesar de alguns trabalhos como o de Basmadjian, Niedermeier e De Meer [BNDM12] visarem isso. Mesmo considerando somente o consumo dos servidores de um CPD, é difícil construir um modelo genérico que se aproxime dos valores de consumo real.

A nível de simplificação do modelo, optou-se aqui por considerar como principal fator para o consumo de energia o uso de CPU e foi considerada somente uma arquitetura de CPU *multicore*. A arquitetura escolhida possui uma tecnologia chamada Intel *SpeedStep*, que altera a alimentação

e frequência de acordo com o uso de CPU, como verificado no Capítulo 3. Por alterar a frequência e a alimentação, a função do consumo de energia pelo uso de CPU acaba também mudando, como também pode ser visto nos gráficos do Capítulo 3. Para construir a função do gráfico da Figura 3.10, é necessário definir uma função por partes, uma vez que há duas retas (obtidas a partir de duas regressões lineares). Reduzindo a escala do eixo de CPU para 0 até 1, a função para gráfico pode ser construída a partir da seguinte função:

$$f(x) = \begin{cases} 18 + 48x, & \text{se o consumo de CPU } x \leq 0,25, \\ 24,675 + 21,3x, & \text{caso contrário.} \end{cases}$$

A função f retorna a potência dado o instante de tempo, como aqui se quer o consumo de energia e usa-se como base o intervalo de tempo da amostragem do trace do Google, detalhado mais adiante, o intervalo considerado aqui é de 300s, ou de $\frac{300}{3600} = 0,083$ hora. Assumindo então que o uso de CPU de um servidor permanece o mesmo dentro desse intervalo, basta multiplicar $f(x)$ por $0,0833$ para se obter o consumo em Wh. Dessa forma obtem-se $g(x)$ que devolve de fato o consumo de energia:

$$g(x) = \begin{cases} 1,5 + 3,9x, & \text{se o consumo de CPU } x \leq 0,25, \\ 2,05 + 1,8x, & \text{caso contrário.} \end{cases}$$

Baseado nessa função, é possível construir o consumo de energia de um servidor da seguinte forma: $E(x) = W_{on} + xW_{cpu}$, onde x é o consumo de CPU e varia de 0 a 1, e W_{cpu} e W_{on} podem assumir diferentes valores. Considerando o conjunto de servidores em $Macs$, tem-se que o consumo total desses servidores é dado por: $T = \sum_{j \in Macs} E(X_j) = \sum_{j \in Macs} W^j_{on} + X_j W^j_{cpu}$, onde X_j é o valor de uso de CPU no servidor j. Rotula-se W_{cpu} e W_{on} com j uma vez que podem mudar de servidor para servidor.

É possível também ocorrer de um servidor não ter qualquer tarefa ou máquina virtual rodando e o uso de CPU ser próximo de 0, podendo até mesmo o servidor ser desligado. Nesse caso, pode-se considerar o servidor com o consumo zero, e assim: $T = \sum_{j \in Macs} E(X_j) = \sum_{j \in Macs} z_j W^j_{on} + X_j W^j_{cpu}$, onde z_j é igual 0 se o servidor j está desligado, ou 1 caso contrário.

Sabendo quais máquinas virtuais estão mapeadas para o servidor j, é possível calcular X_j da seguinte forma: $X_j = \sum_{i \in Tasks} \frac{C^t_i}{C^s_j} x_{ij}$, x_{ij} é igual 1 se i está rodando no servidor j, e 0 caso contrário. Tem-se então que $T = \sum_{j \in Macs} z_j W^j_{on} + \sum_{i \in Tasks} \frac{C^t_i}{C^s_j} W^j_{cpu}$.

Considerando então que essa função calcule o consumo de energia em um dado intervalo de tempo, é necessário considerar também outro processos que consomem CPU também, como a migração. Assumindo então que W_{mig} é a constante que contém o valor consumido de energia durante a migração de uma máquina virtual, é possível redefinir T para também considerar a migração: $T = \sum_{j \in Macs} z_j W^j_{on} + \sum_{i \in Tasks} \frac{C^t_i}{C^s_j} W^j_{cpu} + y_{ij} W_{mig}$, onde y_{ij} é igual a 1 se i estiver migrando para j, ou 0 caso contrário.

Fixando os valores então de valores de y_{ij}, de W_{mig} e de, para cada servidor j, W^j_{on}, e W^j_{cpu}, e também considerando as variáveis x_{ij} e z_j como variáveis binárias de decisão, é possível buscar um mapeamento para as máquinas virtuais em $Task$ de forma que T tenha o valor mínimo para os servidores em $Macs$.

Modelando então o problema como um problema Programação Linear Inteira, obtém-se então a função em 4.1, com as restrições dadas nas fórmulas 4.2, 4.3 e 4.4. Essas restrições garantem que não haja quebra de SLA das máquinas virtuais no mapeamento. A restrição 4.4, em particular, garante que toda máquina virtual seja mapeada a algum servidor.

$$\min \sum_{j \in Macs} z_j W_{on} + \left(\sum_{i \in Tasks} x_{ij} \left(\frac{C_i^t}{C_j^s} W_{cpu} + y_{ij} W_{mig} \right) \right) \tag{4.1}$$

Sujeito a:

$$\forall j \in Macs \left(\sum_{i \in Tasks} x_{ij} C_i^t \right) - z_j C_j^s \leq 0 \tag{4.2}$$

$$\forall j \in Macs \left(\sum_{i \in Tasks} x_{ij} M_i^t \right) - z_j M_j^s \leq 0 \tag{4.3}$$

$$\forall i \in Tasks \sum_{j \in Macs} x_{ij} = 1 \tag{4.4}$$

$$\forall i \in Tasks \; \forall j \in Macs \; 0 \leq x_{ij} \leq 1 \tag{4.5}$$

$$\forall j \in Macs \; 0 \leq z_j \leq 1 \tag{4.6}$$

$$\forall i \in Tasks \; \forall j \in Macs \; x_{ij} \in \mathbb{Z} \tag{4.7}$$

$$\forall j \in Macs \; z_j \in \mathbb{Z} \tag{4.8}$$

4.1.1 Variáveis

Variável	Valor
x_{ij}	$x_{ij} = \begin{cases} 1, & \text{se a MV i vai para o servidor j,} \\ 0, & \text{caso contrário.} \end{cases}$
z_j	$z_j = \begin{cases} 1, & \text{se o servidor j está ligado,} \\ 0, & \text{caso contrário.} \end{cases}$

Tabela 4.1: *Tabela de variáveis.*

A Tabela 4.1 contém as variáveis utilizadas no modelo e os possíveis valores que elas podem assumir.

4.1.2 Constantes

Já a Tabela 4.2 contém as constantes utilizadas no modelo do problema.

A motivação para não considerar a memória RAM na função objetiva vem das informações providas geralmente nos traces de workload e do resultado relatado na subseção 3.4. Os traces geralmente informam, quanto à memória RAM, somente o percentual de memória utilizada; não informam, por exemplo, a quantidade de *page faults*, que é uma informação maior relevância no que concerne o consumo de energia em função do uso da memória. Dado que somente o percentual de memória RAM utilizada pouco pode influenciar no consumo de energia de um servidor, o percentual de RAM utilizada no servidor não foi considerado na função. Dessa forma aqui somente foi

W_{on}^{j}	Representa o consumo inicial de energia de um servidor j em um determinado estado. No caso de um servidor com processador *DVFS*, como o utilizado nos experimentos, W_{on}^{j} pode assumir outros valores de acordo o uso de CPU.
W_{cpu}^{j}	Relação do consumo de CPU do servidor j por uso.
W_{mig}	Apróximo do consumo de energia durante a migração de uma MV.
y_{ij}	$y_{ij} = \begin{cases} 1, & \text{se a MV i está migrando para o servidor j,} \\ 0, & \text{caso contrário.} \end{cases}$

Tabela 4.2: *Tabela de constantes.*

considerado o taxa do consumo de energia por uso de CPU, dada pela constante W_{cpu}^{j}, onde j é o servidor.

A constante W_{mig} é valor aproximado do quanto de energia adicional é necessário para realizar uma migração de uma máquina virtual de um servidor para outro, dessa forma procura-se evitar que uma máquina virtual se desloque entre servidores. O valor dessa constante só é considerado quando a máquina virtual está migrando entre diferentes servidores (no caso $y_{ij} = 1$, ou $y_{ij} = 0$ caso contrário).

Com base nos valores obtidos com o servidor utilizado nos experimentos, os servidores modelados aqui usam os seguintes valores:

Variável	Valor	Condição
W_{on}	1,5 Wh	se o uso da CPU for menor que 25%
W_{on}	2,05 Wh	se o uso da CPU for maior que 25%
W_{cpu}	3,9 Wh	se o uso da CPU for menor que 25%
W_{cpu}	1,8 Wh	se o uso da CPU for maior que 25%
W_{mig}	2 Wh	

Tabela 4.3: *Tabela de valores usados nos experimentos.*

As constantes W_{on} e W_{cpu} mudam de valor de acordo com o estado do servidor no momento antes do algoritmo rodar. Se, por exemplo, o servidor estiver usando 75% da capacidade de sua CPU, $W_{cpu} = 1,8$ Wh e $W_{on} = 2,09$ Wh.

Esses valores da Tabela 4.3 foram calculados de acordo com as duas retas do gráfico 3.10. Para calcular o valor W_{mig}, considerou-se que a migração ocupa em média um minuto um núcleo da CPU de cada servidor, sendo assim $W_{mig} = (30 + 30)(\frac{60}{3600}) = 1$ Wh.

4.1.3 Resultados

As tabelas 4.4, 4.5 e 4.6 contêm os valores obtidos a partir da execução de dois algoritmos: o TASC (heurística proposta aqui) e um exato (*branch-and-bound* implementado pelo CPLEX). Esses testes foram realizados para verificar o desempenho dos algoritmos para instância pequenas do problema, com até 50 máquinas virtuais e 50 servidores; e comparar os resultados das heurísticas com o resultado de um algoritmo exato. Os traces utilizados nos testes foram gerados usando o formato do trace do Google. Todos os servidores possuem valores unitário para CPU e memória e os valores de CPU e memória para as máquinas virtuais foram gerados de forma aleatória, usando a função *random* de Python, que por sua vez utiliza o algoritmo *Mersenne Twister* [ran13].

Cada trace possuem valores equivalentes a um monitoramente de 3000 segundos, ou, 50 minutos.

Os testes se limitaram a 50 servidores e 50 máquinas virtuais por causa do tempo de execução do *CPLEX* para uma instância maior. Para um teste com 60 servidores e 60 máquinas virtuais o tempo de execução levou dias para uma rodada do trace. Os arquivos desses traces utilizados nos testes podem ser baixados [1].

A Tabela 4.4 possui a soma de valores do consumo de energia em cada rodada, calculado de acordo com o modelo explanado anteriormente. Comparando a coluna dos valores obtidos com o CPLEX com os demais algoritmos, é possível ver que, como esperado, o algoritmo exato utilizado pelo CPLEX faz uma alocação dos servidores de forma que o consumo de energia é menor em relação do que ao TASC. Os consumos calculados para o TASC ficaram próximos.

Já a Tabela 4.5 contém o número de migrações para cada algoritmo. É possível ver novamente que o CPLEX faz com que haja um número menor de migrações em relação ao TASC. Esse número menor de migrações em relação às heurísticas se dá também pelo fato que, pelo modelo, é dada uma punição para cada migração realizada.

E a Tabela 4.6 contém o tempo de resposta para a execução de cada algoritmo para os traces gerados. É possível ver que o a execução com o CPLEX necessita de um tempo bem maior que ao TASC.

É possível ver dos resultados dos experimentos que o TASC, em relação ao CPLEX, possui resultados piores (TASC aumenta, em média, 20% o consumo de energia em relação ao CPLEX). No entanto, um algoritmo exato, como o *branch and bound* (algoritmo implementado pelo CPLEX), exige um tempo proibitivo para um ambiente como o de um CPD, uma vez que um CPD pode conter servidores executando serviços e processos críticos em termos de *downtime*, ou mesmo considerando a dinamicidade na demanda de recursos por parte de serviços e processos. Dessa forma, o TASC surge como uma opção para ambientes com um número maior de servidore, pelo seu tempo de resposta e por ter um resultado para o consumo de energia próximo do algoritmo exato.

Número de MVs	Número de servidores	TASC	CPLEX
10	10	313,32 Wh	261,12 Wh
20	20	611,65 Wh	516,36 Wh
30	30	871,01 Wh	724,71 Wh
40	40	1204,93 Wh	998,30 Wh
50	50	1519,93 Wh	1258,10 Wh

Tabela 4.4: *Tabela de total de energia consumida de acordo com a alocação de cada algoritmo.*

Número de MVs	Número de servidores	TASC	CPLEX
10	10	73	21
20	20	145	49
30	30	208	74
40	40	304	107
50	50	387	136

Tabela 4.5: *Tabela com o número de migração de acordo com cada alocação.*

[1] Traces disponíveis em https://github.com/maxrosan/Balancer-Simulator/tree/master/config/traces/generated

Número de MVs	Número de servidores	TASC	CPLEX
10	10	0,00075s	7,82s
20	20	0,00208s	8,1s
30	30	0,00348s	9,14s
40	40	0,00558s	18,23s
50	50	0,00795s	16,67s

Tabela 4.6: *Tabela com o tempo médio para execução de cada algoritmo.*

4.2 Alocação como problema da mochila

Apesar da solução com um algoritmo exato de Programação Linear Interira ser viável para instâncias pequenas do problema, o mesmo não ocorre para instâncias grandes, com mais de 1000 máquinas virtuais ou servidores por exemplo. Buscando então realizar a consolidação alocando a menor quantidade possível de servidores para instâncias maiores, partiu-se então para uma abordagem heurística. Como descrito anteriormente, usou-se como base a heurística de Toyoda. Nesta seção é descrita a heurística desenvolvida.

A primeira parte do algoritmo para consolidação consiste na ordenação das máquinas virtuais e do servidor, e então para cada servidor é executada a heurística de Toyoda. Para reduzir o número de servidores ligados ou mesmo aumentar a carga nos servidores com mais recursos e diminuir nos servidores com menos, a ordenação, tanto dos servidores como das máquinas virtuais, é realizada de forma decrescente. Para ordenar objetos representados por duplas, como um servidor ou máquina virtual, é necessário associar uma chave a cada um desses objetos. O valor dessa chave determina a prioridade um objeto sobre outro na ordenação. Como a ordenação é decrescente, os objetos com as chaves de maior valor têm maior prioridade sobre os objetos de chave com menor valor. As funções escolhidas para calcular as chaves das máquinas virtuais e dos servidores foram aqui denominadas de, respectivamente, $task\_sort\_keys$ e mac_sort_keys. Essas funções estão definidas nos Algoritmos 1 e 2.

Algorithm 1 Função mac_sort_keys

procedure $mac\_sort\_keys(i)$
  return $(F_i^c)^2 + (F_i^m)^2$
end procedure

Algorithm 2 Função $task\_sort\_keys$

procedure $task\_sort\_keys(i)$
  return $\max(C_i^t,\ M_i^t)$
end procedure

Essas funções definidas nos Algoritmos 1 e 2 mostram melhores resultados para o trace do workload do Google. A definição delas pode ser justificada pelo espalhamento dos valores de CPU e memória tanto das tarefas como dos servidores, como visto nas figuras 4.1 e 4.2.

Os gráficos nas figuras 4.1 e 4.2 foram construídas a partir dos valores de CPU e memória dos eventos das tarefas e servidores do trace do Google. O gráfico da Figura 4.1 foi construído a partir de todos as entradas de eventos dos servidores, já o da Figura 4.2 foi construído usando somente o primeiro arquivo de eventos das tarefas, o task_usage/part-00001-of-00500.csv. A descrição do

Algorithm 3 Função *score_task_knapsack*

procedure *score_task_knapsack*(i)
  return $(C_i^t)^2 + (M_i^t)^2$
end procedure

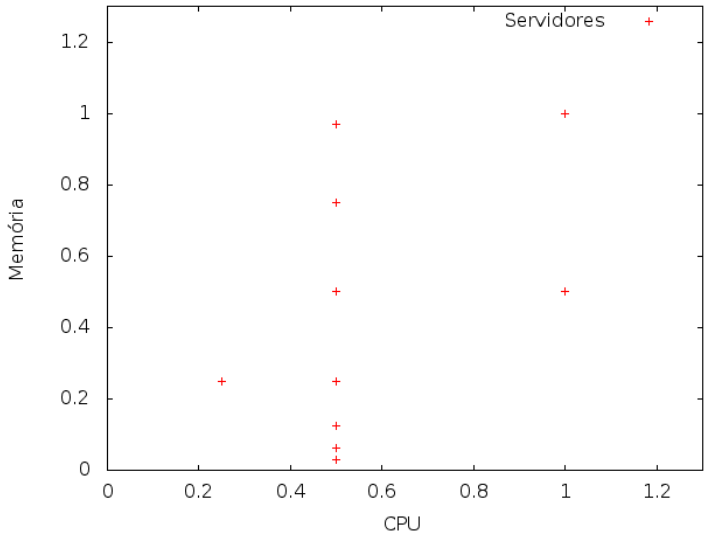

Figura 4.1: *Espalhamento dos servidores.*

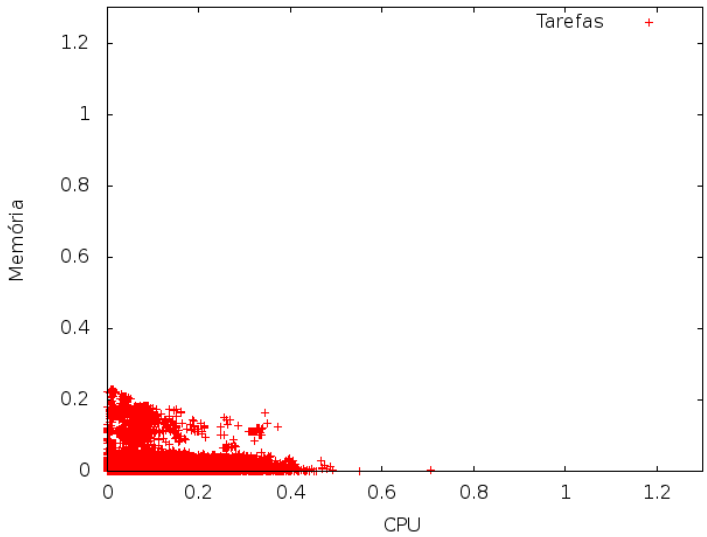

Figura 4.2: *Espalhamento das tarefas.*

trace do workload do Google usa o termo tarefas [tra13], no entanto é possível fazer uma analogia com máquina virtual, uma vez que em relação a CPU e memória uma tarefa pode demandar tanto recurso quanto uma máquina virtual.

Pela disposição do pontos na Figura 4.2, é possível notar um desequilíbrio nas demandas de CPU e memória. As tarefas geralmente demandam mais um recurso do que outro, dificilmente tendo valores próximos da igualdade. Pegando o módulo do par CPU e memória, tarefas onde as demandas por CPU e memória são mais próximas poderiam ter maior prioridade na ordenação do que tarefas que exigem mais de um recurso só. Dessa forma, tarefas que demandam mais de recurso tenderiam a ser alocadas em servidores com menor poder computacional, podendo causar quebra

de SLA.

Já pelo gráfico da Figura 4.1 é possível notar que os valores de CPU e memória dos servidores estão mais próximos. Como há muitos servidores com os mesmos valores para CPU e memória, muitos pontos no gráfico se sobrepuseram; mesmo assim ainda é possível notar que os pontos estão mais afastados do eixo da CPU e do eixo da memória. Assim os servidores podem ser ordenados pelo quadrado do módulo do par de CPU e memória e os servidores com maior poder computacional disponível de fato serão os primeiros a serem alocados na lista decrescente. Escolheu-se o quadrado do módulo a fim de evitar o cálculo da raiz quadrada.

Como aqui o objetivo é reduzir o número de servidores utilizados, a lista de servidores e a lista de máquinas virtuais são construídas a partir de duas outras. A lista de servidores é construída a partir da concatenação da lista ordenada de forma decrescente de servidores utilizados e da lista de servidores não utilizados ordenada da mesma forma. De forma análoga ocorre com a lista de máquinas virtuais. Dessa forma, tenta-se mapear primeiramente as máquinas virtuais nos servidores ainda utilizados para então passar para os servidores não utilizados. Tenta-se também ao construir as listas dessa forma mapear primeiro as máquinas virtuais que utilizam mais recursos, evitando assim quebras de SLA. O Algoritmo 4 mostra a ordenação e alocação dos servidores por meio da heurística de Toyoda.

Como cada máquina virtual é um objeto no Problema da Mochila, a máquina virtual deve ter a ela associado um ganho. Pela definição do Problema da Mochila, como base nesse ganho o objeto será escolhido ou não para uma mochila, ou um servidor no caso. Esse ganho é então calculado aqui pela função definida no Algoritmo 3. Essa função calcula o módulo do vetor da demanda de recursos por parte da máquina virtual. Assim a máquina virtual que exigir mais recursos de um servidor, terá maior prioridade. Tomando com base a ordenação realizada antes de rodar a heurística de Toyoda, as máquinas virtuais com maior exigência de recursos tenderão a ir para os servidores com maior disponibilidade de recursos, evitando assim quebras de SLA.

Por mais que a proposta de Toyoda seja voltada para grandes instâncias do problema da mochila, foi necessário fazer uma divisão das listas de servidores e máquinas virtuais após a ordenação para que houvesse um processamento em paralelo, diminuindo assim o tempo de resposta. Após a ordenação das listas de máquinas virtuais e servidores, as listas são divididas de acordo com o número de threads, dado por $nThreads$, como é possível ver nas linhas 8 e 12 do Algoritmo 4.

Após a divisão das listas ordenadas de servidores e máquinas virtuais, as threads são criadas e possuem como entrada uma lista de servidores e uma lista de máquinas virtuais. As divisões da lista de servidores e máquinas virtuais são processadas aos pares em cada thread. Cada thread então chama a implementação da heurística de Toyoda.

4.3 Alocação das máquinas restantes

Pode ocorrer de algumas máquinas virtuais não serem mapeadas para qualquer servidor pelo algoritmo anterior. Considerando o momento da execução do Algoritmo 4 logo após sair do laço, todos os servidores foram verificados e as máquinas virtuais que restaram caem nos seguintes casos: ou não puderam ter todas as demandas atendidas nesses servidores, ou perderam em prioridade para outras máquinas virtuais e os servidores das iterações restantes não puderam atender suas demandas. Dessa forma, após o laço, as máquinas virtuais restantes podem provocar quebras de

Algorithm 4 Algoritmo para mapeamento das máquinas virtuais.

1: $macs\_f \leftarrow list(\{m \in Macs \mid S_i^t \neq \emptyset\})$ ▷ Pega as máquinas que estão alocadas e as ordena logo em seguida.
2: $macs\_f \leftarrow sort(macs\_f, mac\_key\_sort)$
3: $macs\_e \leftarrow list(\{m \in Macs \mid S_i^t = \emptyset\})$ ▷ De forma semelhante aos dois passos anteriores, pega as máquinas que não estão alocadas e as ordena logo em seguida.
4: $macs\_e \leftarrow sort(macs\_e, mac\_key\_sort)$
5: $macs \leftarrow macs\_f + macs\_e$
6: $tasksNotRunning \leftarrow list(\{i \in Tasks \mid \forall j \in Macs \ . \ i \notin S_i^t\})$
7: $tasksNotRunning \leftarrow sort(tasksNotRunning, task\_key\_sort)$
8: $macsLists \leftarrow divList(macs)$
9: $tasksLists \leftarrow divList(tasksNotRunning)$
10: $c \leftarrow 1$
11: **while** $c \leq nThreads$ **do**
12: $thread(schedule(macsLists[c], tasksLists[c]))$ ▷ O operador $[n]$ quando aplicado a uma lista retorna o n-ésimo elemento da lista. A função $thread(f)$ faz um chamado de função em uma nova thread de execução.
13: $c \leftarrow c + 1$
14: **end while**
15: $wait\_threads()$ ▷ Espera as threads criadas anteriormente finalizarem a execução.

Algorithm 5 Algoritmo chamado em cada thread.

procedure $schedule(macsList, tasksList)$
for $m \in macsList$ **do**
$tasksScheduled \leftarrow toyoda(m, tasksList)$ ▷ *toyoda* contém a implementação da heurística de Toyoda para duas dimensões.
$tasksList \leftarrow tasksList - tasksScheduled$
$S_m^t \leftarrow S_m^t \cup set(tasksScheduled)$ ▷ A função *set* retorna um conjunto formado pelo elementos da lista dada na entrada.
end for
end procedure

Algorithm 6 Algoritmo da função *divList*.

procedure $divList(list)$
$ret \leftarrow alloc(nThreads)$ ▷ A função *alloc* aloca espaço para os cabeçalhos das listas.
$i \leftarrow 1$
$div \leftarrow \frac{|list|}{nThreads}$ ▷ O operador $||$ retorna o número de elementos da lista.
while $i \leq nThreads$ **do**
$s \leftarrow div * (i - 1) + 1$
$e \leftarrow div * i$
$ret[i] \leftarrow list[s..e]$ ▷ $list[s..e]$ retorna uma nova lista contendo os elementos da posição s-ésima até a e-ésima de $list$.
$i \leftarrow i + 1$
end while
return ret
end procedure

Algorithm 7 Mapeamento das máquinas virtuais restantes

1: $macs \leftarrow list(Macs)$
2: $tasksNotRunning \leftarrow list(\{i \in Tasks | \forall j \in Macs.i \notin S_i^t\})$
3: $tasksNotRunning \leftarrow sort(tasksNotRunning, task\_key\_sort)$
4: **for** $i \in tasksNotRunning$ **do**
5: $\quad k \leftarrow \min_{j \in macs} \{(C_i^t - C_j^m)^2 + (M_i^t - M_j^m)^2\}$
6: $\quad S_k^t \leftarrow S_k^t \cup \{i\}$
7: **end for**

SLA.

Mesmo que as máquinas virtuais restantes tenham SLA quebrado, ainda sim é necessário mapeá-las, uma vez que pode ocorrer de pelo menos uma delas estar executando um serviço crítico. Deve-se então procurar, para cada máquina virtual, qual servidor atenderá maior parte da demanda da máquina virtual por recurso. Como a demanda e a oferta de recursos aqui são representadas por vetores, procura-se então, para cada máquina virtual, qual servidor tem a menor distância para a máquina virtual.

É possível ver o que foi descrito anteriormente no Algoritmo 7. A lista $taskNotRunning$ contém todas as tarefas não mapeadas ordenadas de forma decrescente. A cada iteração do laço da linha 4, a máquina virtual corrente é mapeada para o servidor com menor distância euclidiana ao quadrado.

4.4 Experimentos

A fim de comparar o resultado do TASC com outro algoritmo utilizado em outras propostas, o algoritmo escolhido a ser implementado e ter seus resultados comparados com TASC foi FFD. O FFD é utilizado como base em algumas outras propostas de algoritmo, uma vez que é uma das heurísticas utilizadas para resolver o Problema do Bin Packing, que por sua vez é também usado como base para modelar o problema de alocação de servidores. Em propostas como a de Murtazaev et al. [MO+11], o problema de alocação é modelado com o *Vector Packing*, uma variação do Bin Packing.

Um simulador foi desenvolvido para implementar tanto TASC como o (FFD). O simulador recebe como entrada um trace disponibilizado pelo Google [tra13]. O trace basicamente é composto de eventos envolvendo tarefas e os servidores de um CPD do Google. Como dito anteriormente, a descrição do trace do Google usa o termo tarefa no lugar de máquinas virtuais, dessa forma para descrever o resultado dos experimentos o termo tarefa também será utilizado no lugar de máquina virtual.

Baseado no trace, o simulador tenta alocar as tarefas para os servidores de acordo com o algoritmo de alocação, ou FFD ou TASC.

4.4.1 Algoritmo do FFD implementado

O Algoritmo 8 contém o First Fit Decreasing usado nos experimentos. Nessa implementação não há ordenação dos servidores, como utilizado em algumas heurísticas. Somente as máquinas virtuais são ordenadas, e então essa lista ordenada de máquinas virtuais é iterada e para cada máquina virtual procura-se um servidor capaz de executá-la sem quebra de SLA. A máquina virtual da iteração é mapeada para o primeiro servidor encontrado capaz de executá-la.

Algorithm 8 Algoritmo FFD implementado para os experimentos.

1: $macs \leftarrow list(Macs)$
2: $tasksNotRunning \leftarrow list(\{i \in Tasks | \forall j \in Macs.i \notin S_i^t\})$ ▷ Pega todas as tarefas que não estejam mapeadas para qualquer servidor.
3: $tasksNotRunning \leftarrow sort(tasksNotRunning, task\_key\_sort)$ ▷ Ordena de forma decrescente as máquinas virtuais tendo como chave o valor retornado pela função $task\_key\_sort$.
4: **for** $i \in tasksNotRunning$ **do**
5: **for** $j \in macs$ **do**
6: **if** $can\_run(j, i)$ **then** ▷ Verifica se o servidor j pode executar a máquina virtual i.
7: $S_j^t \leftarrow S_j^t \cup \{i\}$ ▷ Coloca a máquina virtual j no servidor i.
8: **end if**
9: **end for**
10: **end for**

4.4.2 Simulador

O simulador criado[2] é de eventos discretos e tem como objetivo realizar experimentos com diferentes estratégias para consolidação de servidores e assim verificar seus resultados. Foi criado inicialmente para simular o trace do workload disponibilizado pelo Google, entretanto pode ser estendido para ser usado com outros workloads.

O simulador tem como entrada o trace do workload descrito anteriormente. Para o experimento do trabalho presente somente informações sobre o uso de recursos por parte da tarefas e os eventos envolvendo servidores foram considerados. É reconhecido na descrição do trace que há algumas informações faltando no trace, foi necessário ignorar alguns registros para manter a validade dos resultados. As seguintes tarefas estão sendo ignoradas na entrada: tarefas que não possuem valores para o consumo de CPU ou memória; tarefas que possuem valores nulos para consumo tanto de CPU como de memória; e tarefas que possuem valores inválidos no consumo de CPU e memória, ou seja, valores negativos ou valores acima de 1.

O tempo na simulação flui de acordo com o início das tarefas e dos eventos relacionados aos servidores. De acordo com a descrição do trace, as medições do uso de recurso ocorreram em um intervalo de 300 segundos. Baseado nisso, o simulador lê todos os registros de eventos que estejam dentro de intervalos de 300 segundos de trace. Cada intervalo de 300 segundos de trace é chamado aqui de rodada. Ou seja, a rodada 0 contém todos os eventos do trace no intervalo de 0 a 300 segundos. Logo depois de ler todos os registros de uma rodada, o simulador chama o algoritmo de alocação para fazer o mapeamento das tarefas para os servidores.

4.4.3 Trace do Google

A fim de verificar o algoritmo proposto em um ambiente o mais próximo possível dos CPDs, utilizou-se o trace disponibilizado pelo Google. O Google disponibilizou um trace de um mês de uso dos servidores em seu CPD. O trace está dividido em servidores, jobs e tarefas. Os jobs são compostos de várias tarefas e essas tarefas são mapeadas para os servidores. Os arquivos do trace informam o uso dos recursos de cada tarefa, em qual servidor aquela tarefa foi executada, e os eventos que ocorreram tanto com os servidores como com as tarefas. Para construção dos experimentos somente os eventos envolvendo servidores e tarefas foram utilizadaos.

Os valores numéricos disponibilizados no trace estão normalizados e não é fornecida qualquer informação sobre os valores reais do hardware dos servidores. Os simuladores como CloudSim[clo13],

[2]Código-fonte disponível em https://github.com/maxrosan/Balancer-Simulator

RealCloudSim[rea13] e ICanCloud[NVPC+12] pedem valores detalhados, como por exemplo MIPS (milhões de instruções por segundo), memória RAM e armazenamento em bytes. Como aqui o algoritmo objetiva resolver somente o problema da alocação, muitas dessas informações são desnecessárias ou podem ser considerados valores ideais. A partir desses fatores, foi desenvolvido um simulador próprio para verificar o desempenho do algoritmo.

Os arquivos do trace comprimidos possuem aproximadamente 40Gb [tra13] de tamanho. Para executar os experimentos, foram utilizados todos os instantes de tempo do trace, dessa forma todos os arquivos de eventos de tarefas e servidores do trace foram utilizados.

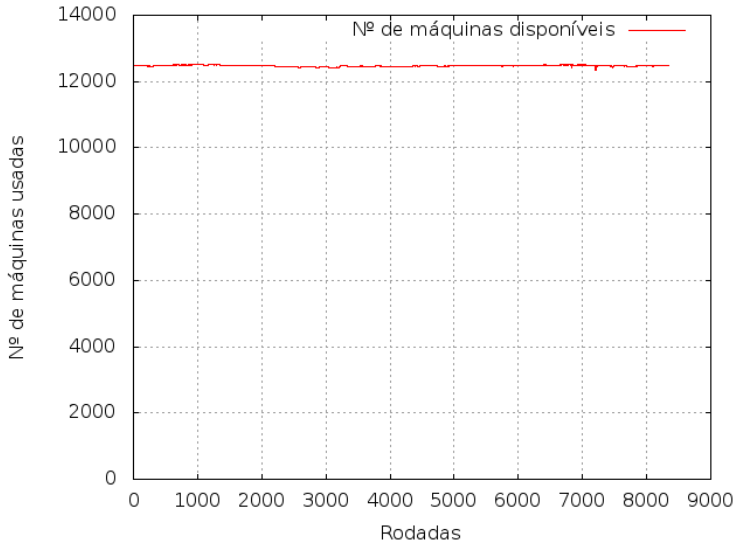

Figura 4.3: *Número de servidores em função do tempo.*

Antes de analisar os gráficos com os resultados da simulação, o comportamento do workload será analisado para então ser analisado e comparado o comportamento dos algoritmos de alocação em determinados pontos do trace. O gráfico na Figura 4.3 mostra a variação no número de máquinas ao longo do tempo. Como é possível ter uma noção a partir do gráfico, o número de máquinas varia em torno de 12500. Não se espera então que a variação no número de máquinas no trace do workload influencie os resultados dos algoritmos implementados.

A Figura 4.4 contém o gráfico do total de tarefas que estão executando e que chegaram. Já o gráfico na Figura 4.6 mostra o fluxo de novas tarefas. Na média, o simulador tem 119131 tarefas mapeadas em cada rodada.

A Figura 4.6 contém o gráfico de novas tarefas ao longo do tempo. Em média, 3756 tarefas novas chegam em cada rodada.

4.4.4 Ambiente

Para rodar os experimentos foi utilizada uma máquina com as seguintes características:

- CPU: Intel Core i7-2700K Quad-Core Processor 3.5 GHz 8 MB Cache
- Memória RAM: 16 GB
- Sistema Operacional: Debian Squeeze 6.0.5

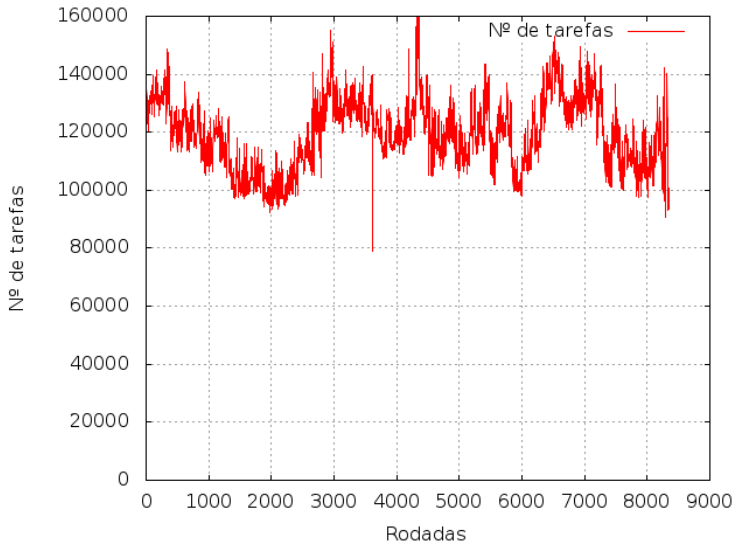

Figura 4.4: *Total de tarefas em função do tempo.*

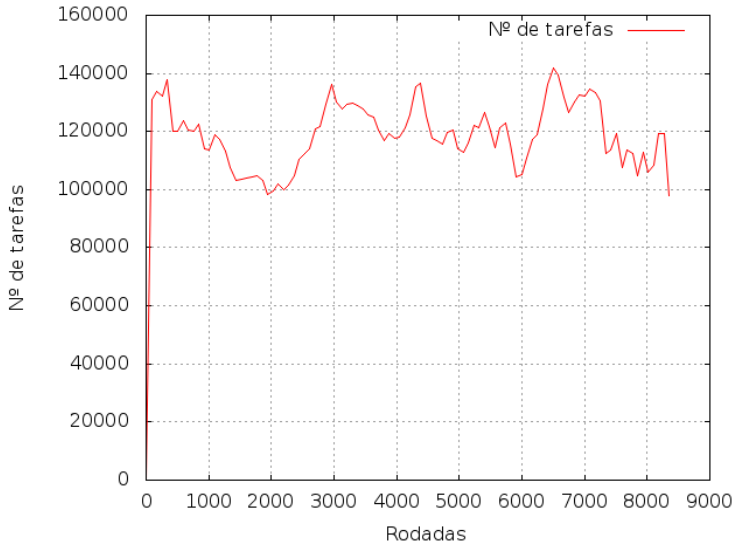

Figura 4.5: *Total de tarefas em função do tempo.*

4.4.5 Implementação

O simulador foi implementado na linguagem Python. O interpretador utilizado para rodar os experimentos foi o Pypy na versão 2.0.0-beta1.

4.4.6 Resultados

Para verificar o desempenho das duas propostas foram analisados os seguintes valores: consumo de energia, número de migrações e o número de máquinas utilizadas. O consumo de energia para cada rodada é calculado de acordo com o modelo desenvolvido e apresentado na Seção 4.1. Tanto para SLA,

A Figura 4.7 contém um gráfico comparativo para consumo de energia dos servidores de acordo com a alocação de cada algoritmo. Para melhor visualizar o comportamento do consumo em função da rodada, os valores foram recalculados usando a curva de Bezier. É possível notar, em ambos

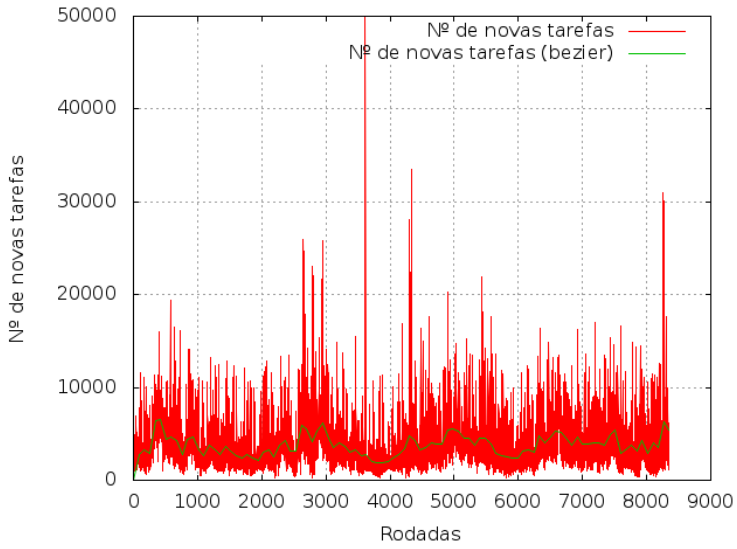

Figura 4.6: *Número de novas tarefas em função do tempo.*

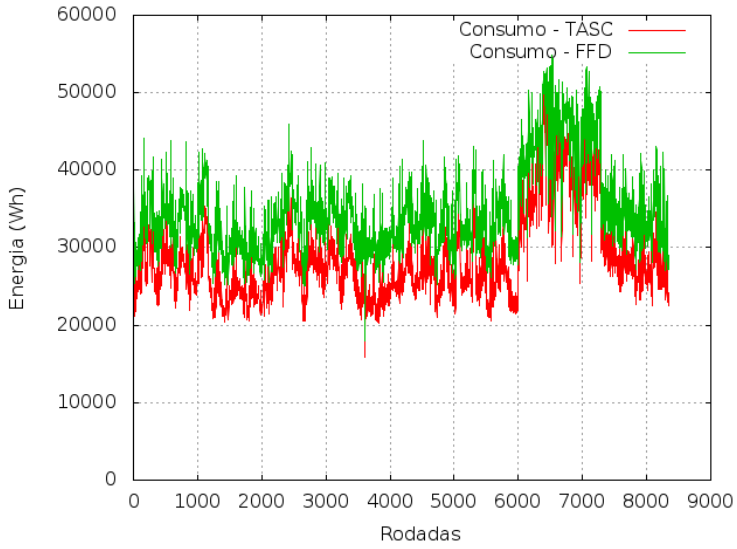

Figura 4.7: *Gráfico comparativo com consumos com Knapsack e FFD.*

os gráficos, que o TASC faz uma alocação na qual o consumo de energia na rodada é menor que na alocação realizada pelo FFD. O consumo maior por parte do FFD deve-se também ao maior número de migrações e o maior número de servidores alocados, como é possível ver nos gráficos de número de migrações e de servidores alocados. Em média, o TASC faz o consumo por rodada ser 6125 Wh a menos que o FFD.

O gráfico comparativo na Figura 4.9 contém o número de migrações realizadas nos dois algoritmos. Também a fim de melhor visualizar o comportamento do gráfico, foi construido um outro gráfico usando a curva de Bezier, como pode ser visto em 4.10. Vale ressaltar que a política de migração utilizada foi a mesma para ambos os algoritmos: a migração somente ocorre quando a máquina não tem mais recursos para executar uma determinada máquina virtual, o servidor vai ser desligado ou os recursos disponíveis no servidor são alterados. No gráfico é possível ver que TASC faz com que ocorra um número menor de migrações do que em relação ao FFD. Em média, TASC provocou 5053 migrações a menos que FFD.

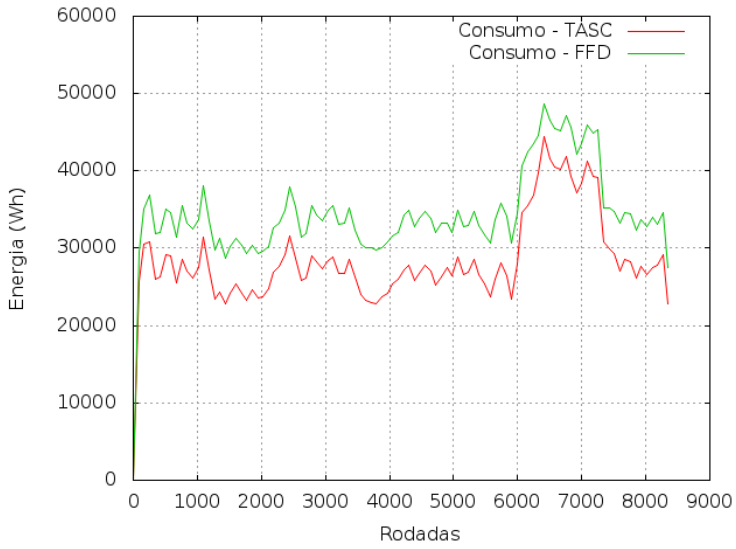

Figura 4.8: *Gráfico comparativo com consumos com Knapsack e FFD.*

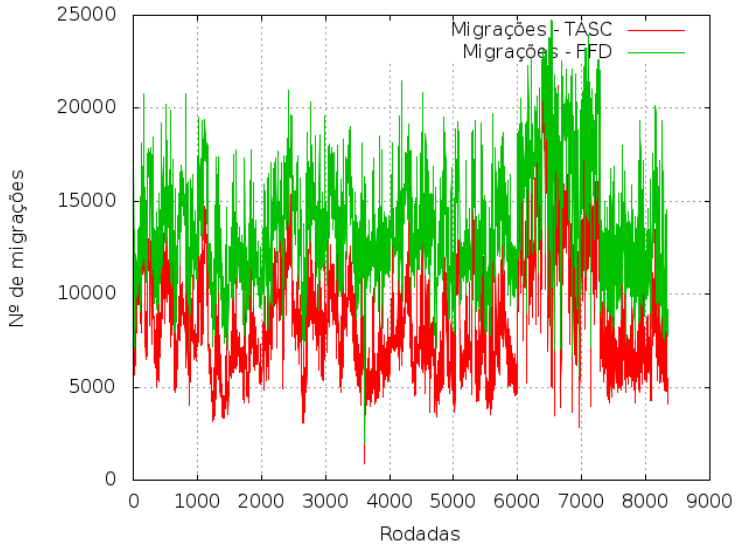

Figura 4.9: *Número de migrações.*

Na Figura 4.11 tem-se o gráfico do número de máquinas usadas. É possível notar que o FFD faz com que um maior número de máquinas sejam utilizadas que o TASC. Considerando todas as rodadas apresentados no gráfico da Figura 4.11, TASC usa em média 916 máquinas a menos que FFD. Em termos absolutos, a diferença máxima entre os dois algoritmos foi de 2353 servidores e a diferença mínima nula.

Concluindo, é possível notar a partir dos resultados da simulação que o TASC aloca os servidores de forma a consumir menos energia do que o FFD, um menor número de migrações, e um menor número de servidores utilizados. Por usar menos servidores e menos migrações, o TASC poderia causar um menor consumo de energia em um CPD, por exemplo, se comparado ao uso do FFD.

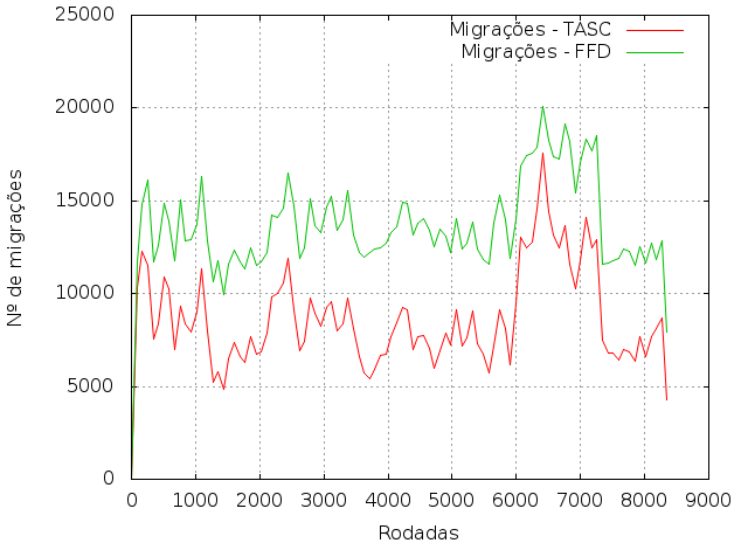

Figura 4.10: *Número de migrações.*

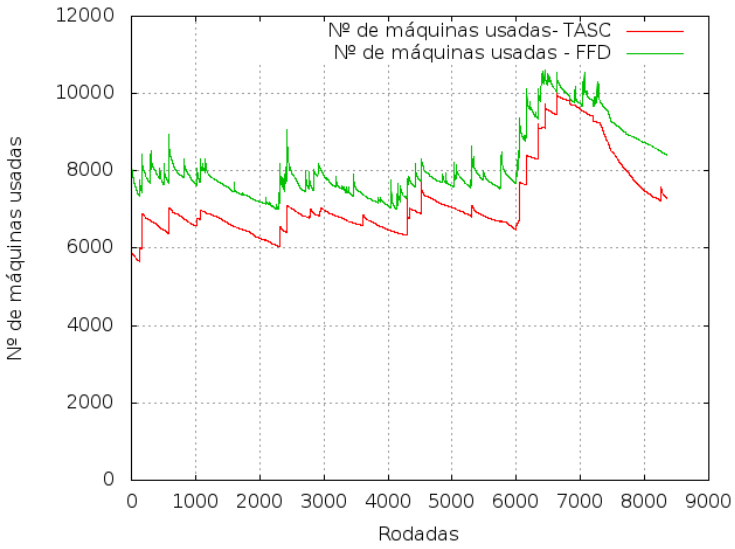

Figura 4.11: *Servidores utilizados ao longo do tempo.*

Capítulo 5

Trabalhos Relacionados

Aqui são apresentados alguns trabalhos relacionados ao tema da pesquisa. Na Seção 5.1 são discutidos algumas propostas de algoritmos para a realização de consolidação de servidores. Já na Seção 5.2 são discutidos alguns trabalhos sobre o consumo de energia em um servidor com virtualização.

5.1 Consolidação de servidores por meio da virtualização

Murtazaev et al. [MO+11] apresentam uma proposta para realizar a consolidação de servidores por meio da virtualização. O problema nesse trabalho é modelado como um *Vector-Packing Problem* e o algoritmo proposto, o Sercon, é beaseado em dois outros algoritmos gulosos para aquele problema: *First Fit Decreasing* e *Best Fit Decreasing*. Um dos objetivos do algoritmo, além de realizar a consolidação de servidores, é realizar o menor número possível de migrações. O algoritmo ordena os servidores de forma decrescente e pega o último servidor da lista ordenada, marcando as máquinas virtuais como candidatas à migração. Logo após atribui pesos a essas MVs e então tenta mapeá-las para os primeiros servidores na lista. Passa então a cadidatar à migração as MVs no penúltimo servidor da lista e assim por diante. Na simulação realizada no trabalho o Sercon apresentou melhores resultados em termos de número de migrações e número de máquinas utilizadas às heurísticas do *Bin Packing*.

Bobroff, Kochut e Beaty [BKB07] também apresentam um algoritmo dinâmico para consolidação por meio da virtualização, *Measure-Forecast-Remap*. Além de propor um algoritmo de alocação de servidores para máquinas virtuais, sendo esse também baseado no FFD, é proposto também um método de predição do uso de recursos. O MFR também recebe como parâmetro de sobrecarga que define o intervalo no qual um servidor pode ficar sobrecarregado, ou seja, não atendendo a todas as demandas das MVs nele sendo executadas. Para realizar as predições do uso de recursos, o MFR armazena durante um intervalo o quanto de cada recurso cada máquina virtual utilizou, baseado então nesses valores faz-se a predição e assim com os valores da predição e com o parâmetro de sobrecarga, o algoritmo realiza um mapeamento. Essa operação repete-se periodicamente. Nos resultados da simulação, o MFR conseguiu reduzir a quantidade de recurso utilizado em relação à alocação estática e o número de quebras de SLAs.

Na tentativa de reduzir o consumo de energia em um servidor pouco afetando sua eficiência, trabalha-se também na camada de hardware, como é o caso do *Dynamic Voltage/Frequency Scaling* (DVFS). Em abordagens do tipo, a partir do conhecimento de deadline de tarefas ou momentos

de pico do workload, pode-se regular o consumo ou a frequência do processador de acordo com as tarefas que o servidor está executando. No entanto é um assunto ainda estudado, como na proposta de formalização da técnica no trabalho de Wu et al. [WJM+05], e exige certo suporte do hardware para que a abordagem seja implementada.

Algumas outras soluções como *PowerNap* de Meisner, Gold e Wenisch [MGW09] também propõem soluções ainda na camada de hardware. PowerNap é uma abordagem onde somente partes de um servidor que recebem as mensagens via rede contendo informações sobre trabalho devem permanecer ligadas ou com o *full-clock*, as demais podem ser desligadas ou ter o clock reduzido. Assim que uma mensagem chega com trabalho, a parte que recebeu a mensagem envia um sinal para as demais serem religadas e então o trabalho é executado no servidor. Por mais que possa ter um desempenho melhor que o DVFS, essa abordagem só funciona caso o servidor tenha hardware capaz de fazer um rápida transição de clock, do contrário atrasaria a execução do trabalho e tornaria a solução inviável. Essa solução para ser implantada, como sugerido pelos autores, exige hardware com peças e características específicas, o que seria algo bastante custoso em termos financeiros para um CPD se adaptar a essa tecnologia caso não a tenha.

No trabalho de Sekhar, Jeba e Durga [SJD] é realizado um *survey* no assunto consolidação de servidores com migração. São apresentadas algumas arquiteturas já propostas para permitir a consolidação de servidores, assim como alguns algoritmos propostos. Também são comentadas algumas técnicas para realizar a *live migration*, que é uma técnica fundamental para os algoritmos de alocação dinâmica. Clark et al. [CFH+05] mostram em seu trabalho como é possível realizar a *live migration* em um ambiente virtualizado com *Linux*, propondo em seu trabalho um método de *live migration* baseado na abordagem de pré-cópia de páginas de memória.

5.2 Análises e modelos para consumo de energia de servidores com virtualização

Berl et al. [BG+10] revisam parte da literatura no que concerne ao uso eficiente de energia nos ambientes de nuvem. No trabalho é dito que grande parte do consumo de energia em um CPD vem dos servidores e outra considerável fatia do consumo vem de equipamentos dos quais os servidores são dependentes, como *cooler*. São também revisadas algumas das técnicas utilizadas para redução do consumo de energia por parte dos servidores. Um delas é a redução de consumo na camada de hardware, através do uso de equipamentos que consomem menos energia; por exemplo, pode-se usar os SSDs (*Solid-State Discs*), reduzir o frequência dos processadores quando houver baixa demanda ou desligar partes do hardware. Dessa forma é possível desligar parte do sistema de *cooling* e economizar tanto energia dos *coolers* como dos servidores. Outra forma citada no trabalho é dispor as tarefas, ou máquinas virtuais no caso do presente trabalho, de forma a reduzir a comunicação e ser ciente de certas propriedades das tarefas a ponto de saber quando e onde escalonar a tarefa de forma que não afete o desempenho e economize energia.

No trabalho de Kommeri, Niemi e Helin [KNH12] são também analisados alguns cenários de consumo de energia em um servidor com máquinas virtuais. As medições foram realizadas usando Xen e KVM. Foi concluido nesse trabalho que o consumo do KVM chega a ser próximo ao do hardware. No entanto, foi ressaltado o overhead que a virtualização possui. À medida que o número de máquinas virtuais vai aumentando e as tarefas são divididas entre essas máquinas, o consumo de

energia aumenta significativamente e o tempo de resposta diminui. Conclui que é melhor dedicar vários recursos de hardware a poucas máquinas virtuais com uma carga significativa de tarefas, do que dedicar esses mesmos recursos de hardware a muitas máquinas virtuais com tarefas leves.

Já trabalhos relacionados à modelagem do consumo de energia em um CPD, Basmadjian, R. and Niedermeier, F. and De Meer, H. [BNDM12] propõem um modelo mais genérico para estimar o consumo de energia de um servidor sem carga. Os autores consideram, além da CPU, diversas outras partes de um servidor como disco rígido, memória e a alimentação. Embora considerar outras partes de um servidor além da CPU permita uma estimativa melhor do consumo, exige informações nem sempre disponíveis pelos fabricantes dos equipamentos ou mesmo pelos CPDs, como é o caso do trace liberado pelo Google.

Capítulo 6

Conclusões

No presente trabalho foi apresentada uma proposta de algoritmo de alocação de máquinas virtuais em servidores em ambientes como CPDs. O presente capítulo revisa os resultados obtidos ao longo do trabalho, assim como também a conclusão obtida a partir desses resultados.

6.1 Considerações Finais

Foi proposto aqui um algoritmo para realizar alocação de servidores para máquinas virtuais objetivando usar o menor número de servidores e mapear todas as máquinas virtuais, ou seja, realizar a consolidação de servidores por meio da virtualização. Para isso, usou-se como base o Problema da Mochila Multidimensional, duas dimensões foram consideradas aqui, e a heurística proposta por Toyoda para resolver tal problema. Foi também implementado o algoritmo FFD para comparar os resultados da proposta, uma vez que o FFD é utilizado como base para diversas outras soluções propostas para realizar a consolidação de servidores por meio da virtualização.

Como mostrado Capítulo 3, alguns experimentos foram realizados para mostrar que por meio da virtualização é possível haver uma maior economia de energia sem afetar em demasia o desempenho dos serviços. Os resultados dos experimentos mostraram que, de fato, a disputa por recursos entre as máquinas virtuais em um mesmo servidor pode fazer com que haja uma leve piora no desempenho de processos *CPU-bound* e *IO-bound*, quando comparado com o desempenho em um servidor dedicado. No entanto a redução no consumo também mostrada nos experimentos faz da consolidação com virtualização uma solução bastante viável. É possível concluir desses experimentos que, havendo um casamento entre os recursos disponíveis por um servidor e os requisitados pelas máquinas virtuais, a consolidação por meio da virtualização pode promover de fato uma redução no consumo de energia sem afetar o desempenho dos serviços.

No Capítulo 4 foi apresentado o algoritmo proposto para a alocação, além do modelo para estimar o consumo de energia em um CPD. Foi visto que, para instâncias pequenas e para o modelo desenvolvido, um algoritmo exato de Programação Linear Inteira apresenta melhores soluções que as heurísticas em termos energéticos, uma vez que busca o valor ótimo. No entanto, como o tempo de resposta explode com o aumento do número de servidores ou máquinas virtuais, a heurística passa a ser a única solução viável para uma alocação dinâmica.

Ainda no Capítulo 4, além da descrição do algoritmo, foram apresentados resultados dos experimentos realizados com o algoritmo e o FFD, comparando os resultados dos dois algoritmos. Esses experimentos utilizada o trace do workload do Google disponibilizado publicamente. É possível ver,

a partir dos resultados apresentados, que o TASC provoca um menor consumo de energia em sua alocação que o FFD, de acordo com o modelo utilizado para calcular o consumo de energia. O TASC também faz com que haja um menor número de migrações e com que um número menor de servidores seja utilizado em relação ao FFD.

6.2 Sugestões para Pesquisas Futuras

Nesta seção são enumerados alguns pontos para possíveis pesquisas futuras.

- Verificar e comparar o desempenho do FFD e do TASC com diferentes políticas de migração;
- Verificar o FFD e o TASC com outros workloads de diferentes CPDs;
- Considerar como parte do problema a clusterização do CPD. Associar servidores a clusters e tentar mapear máquinas virtuais relacionadas de forma a ficarem no mesmo servidor ou em servidores de um mesmo cluster.

Bibliografia

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica e Matei Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, Abril 2010. 1

[AGPR13] D. Adami, S. Giordano, M. Pagano e S. Roma. Virtual Machines Migration in a Cloud Data Center Scenario: An Experimental Analysis. Em *IEEE International Conference on Communications (ICC)*, páginas 2578–2582, June 2013. 1

[ama13] Amazon EC2 SLA. http://aws.amazon.com/pt/ec2-sla/, 2013. Acessado em 18/09/2013. 2

[BB10] A. Beloglazov e R. Buyya. Energy Efficient Allocation of Virtual Machines in Cloud Data Centers. Em *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, páginas 577–578, May 2010. 1

[BBA10] Rajkumar Buyya, Anton Beloglazov e Jemal Abawajy. Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges. Em *Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010)*, páginas 6–20, 2010. 1

[BCH13] Luiz André Barroso, Jimmy Clidaras e Urs Hölzle. *The Datacenter as a Computer – An Introduction to the Design of Warehouse-Scale Machines*, chapter Chapter 5 – Energy and Power Efficiency. Morgan & Claypool, 2a. edição, 2013. 1

[BG+10] A. Berl, E. Gelenbe et al. Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051, 2010. 42

[BH07] L. A. Barroso e U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007. 16

[BKB07] N. Bobroff, A. Kochut e K. Beaty. Dynamic placement of virtual machines for managing SLA violations. Em *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, páginas 119–128. IEEE, 2007. 2, 6, 7, 41

[BNDM12] R. Basmadjian, F. Niedermeier e H. De Meer. Modelling and analysing the power consumption of idle servers. Em *Sustainable Internet and ICT for Sustainability (SustainIT), 2012*, páginas 1–9. IEEE, 2012. 24, 43

[CFH+05] C. Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt e Andrew Warfield. Live migration of virtual machines. Em *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, páginas 273–286. USENIX Association, 2005. 23, 42

[CFM+14] Gustavo Callou, João Ferreira, Paulo Maciel, Dietmar Tutsch e Rafael Souza. An Integrated Modeling Approach to Evaluate and Optimize Data Center Sustainability, Dependability and Cost. *Energies*, 7:238–277, 2014. 3

[clo13] The CLOUDS Lab: Flagship Projects - Gridbus and Cloudbus. http://www.cloudbus.org/cloudsim/, 2013. Acessado em 28/07/2013. 34

[Col11] David Cole. Data Center Energy Efficiency – Looking Beyond PUE, 2011. No Limits Software White Paper Number 4. http://www.nolimitssoftware.com/docs/DataCenterEnergyEfficiency_LookingBeyond.pdf. Acessado em 12/3/2014. 1

[coo14] Is There a Liquid Fix for the Cloudś Heavy Energy Footprint ? http://goo.gl/00I0g5, 2014. Acessado em 19/01/2014. 3

[Cre81] Robert J. Creasy. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490, 1981. 5

[DMT11] K. Dahbur, B. Mohammad e A. B. Tarakji. A survey of risks, threats and vulnerabilities in cloud computing. Em *Proceedings of the 2011 International conference on intelligent semantic Web-services and applications*, página 12. ACM, 2011. 2

[Dro] Dropbox. Dropbox - Sobre o Dropbox. https://www.dropbox.com/about. Acessado em 12/3/2014. 1

[Fré04] A. Fréville. The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004. 8

[Goo14] Google. Google Docs, Planilhas e Apresentacoes do Google - Ajuda do Drive, 2014. https://support.google.com/drive/answer/49008?hl=pt-BR. Acessado em 12/3/2014. 1

[hps13] Service level agreements for all products | HP public cloud. http://www.hpcloud.com/sla/, 2013. Acessado em 18/09/2013. 2

[HWO97] P. Hicks, M. Walnock e R. M. Owens. Analysis of power consumption in memory hierarchies. Em *Proceedings of the 1997 international symposium on Low power electronics and design*, páginas 239–242. ACM, 1997. 17

[jpc14] JPC. http://jpc.sourceforge.net/home_home.html, 2014. Acessado em 13/03/2014. 6

[Kan13] Michael Kanellos. Google Says: Save Energy, Ditch Your Data Center, 2013. http://www.forbes.com/sites/michaelkanellos/2013/06/06/google-says-save-energy-ditch-your-data-center/. Acessado em 12/3/2014. 1, 2

[KNH12] J. Kommeri, T. Niemi e O. Helin. Energy efficiency of server virtualization. Em *ENERGY 2012, The Second International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, páginas 90–95, 2012. 42

[KPP04] H. Kellerer, U. Pferschy e D. Pisinger. *Knapsack Problems*. Springer-Verlag, 1º edição, 2004. 7

[LAAA14] R. Latif, H. Abbas, S. Assar e Q. Ali. Cloud Computing Risk Assessment: A Systematic Literature Review. Em *Future Information Technology*, páginas 285–295. Springer, 2014. 2

[LMM02] Andrea Lodi, Silvano Martello e Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002. 9

[lpc13] IBM ILOG CPLEX Optimization Studio Preview Edition Trial. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, 2013. Acessado em 30/12/2013. 7

[lpg13] GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/, 2013. Acessado em 30/12/2013. 7

[lpm13] Solve linear programming problems - MATLAB. http://www.mathworks.com/help/optim/ug/linprog.html, 2013. Acessado em 30/12/2013. 7

[MGW09] D. Meisner, B. Gold e T. F. Wenisch. PowerNap: eliminating server idle power. Em *ACM Sigplan Notices*, volume 44, páginas 205–216. ACM, 2009. 42

[MO+11] A. Murtazaev, S. Oh et al. Sercon: Server consolidation algorithm using live migration of virtual machines for green computing. *IETE Technical Review*, 28(3):212, 2011. 2, 6, 10, 33, 41

[NVPC+12] Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero e Ignacio M Llorente. iCanCloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012. 35

[OnL13] OnLive. OnLive Background, 2013. https://www.onlive.com/about. Acessado em 12/3/2014. 1

[qem13] QEMU Emulator User Documentation. http://qemu.weilnetz.de/qemu-doc.html, 2013. Acessado em 23/09/2013. 6

[ran13] random - Generate pseudo-random numbers. http://docs.python.org/2/library/random.html, 2013. Acessado em 30/12/2013. 27

[rea13] RealCloudSim. http://sourceforge.net/projects/realcloudsim/, 2013. Acessado em 28/07/2013. 35

[SJD] J. Sekhar, G. Jeba e S. Durga. A survey on energy efficient server consolidation through VM live migration. *International Journal*, 5. 42

[spe14] Frequently asked questions for Enhanced Intel SpeedStep® Technology on mobile. http://www.intel.com/support/processors/sb/CS-028855.htm, 2014. Acessado em 04/01/2014. 16

[SS89] James K Strayer e JK Stryer. *Linear programming and its applications*. Springer New York, 1989. 7

[ST68] Shizuo Senju e Yoshiaki Toyoda. An approach to linear programming with 0-1 variables. *Management Science*, 15(4):B–196, 1968. 24

[SVLN13] Kristian Sandstrom, Aneta Vulgarakis, Markus Lindgren e Thomas Nolte. Virtualization technologies in embedded real-time systems. Em *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, páginas 1–8. IEEE, 2013. 5

[sys13] System performance benchmark. http://sysbench.sourceforge.net/, 2013. Acessado em 23/09/2013. 12

[TB05] T. Trygar e G. Bain. A framework for service level agreement management. Em *Military Communications Conference, 2005. MILCOM 2005. IEEE*, páginas 331–337. IEEE, 2005. 6

[Toy75] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, 21(12):1417–1427, 1975. 3, 8, 23

[tra13] Second format of cluster-usage traces - Traces of Google workloads - Google Project Hosting. http://code.google.com/p/googleclusterdata/wiki/TraceVersion2, 2013. Acessado em 28/07/2013. 30, 33, 35

[vir13] VirtualBox Documentation - Ch. 1. https://www.virtualbox.org/manual/ch01.html, 2013. Acessado em 23/09/2013. 6

[Vog08] W. Vogels. Beyond server consolidation. *Queue*, 6(1):20–26, 2008. 1, 2

[vsp13] Free VMware vSphere Hypervisor, Free Virtualization (ESXi). http://www.vmware.com/products/vsphere-hypervisor/, 2013. Acessado em 23/09/2013. 6

[WJM+05] Qiang Wu, Philo Juang, Margaret Martonosi, L-S Peh e Douglas W Clark. Formal control techniques for power-performance management. *Micro, IEEE*, 25(5):52–62, 2005. 42