PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Dynamic server consolidation with controlled reconfiguration delays

TIAGO COELHO FERRETO

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. César Augusto Fonticielha De Rose
Co-orientador: Prof. Dr. Hans-Ulrich Heiß

Porto Alegre
2010

Dados Internacionais de Catalogação na Publicação (CIP)

F387d Ferreto, Tiago Coelho
Dynamic server consolidation with controlled reconfiguration delays / Tiago Coelho Ferreto. – Porto Alegre, 2010.
75 f.

Tese (Doutorado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. César Augusto Fonticielha De Rose.

1. Informática. 2. Redes de Computadores.
3. Máquinas Virtuais. I. De Rose, César Augusto Fonticielha.
II. Título.

CDD 004.65

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Consolidação dinâmica de servidores com atrasos de reconfiguração controlados

RESUMO

A virtualização reapareceu nos últimos anos como solução para minimizar custos em data centers decorrentes da subutilização de recursos. A possibilidade de dinamicamente alterar a capacidade de máquinas virtuais e migrá-las de forma transparente entre servidores físicos resultou em maior flexibilidade para atender mudanças repentinas em demanda computacional, minimizando a quantidade de recursos físicos necessários. Este aumento em flexibilidade derivou novos modelos de computação como *utility computing* e *cloud computing*. Um dos principais mecanismos nestes data centers virtualizados é a consolidação dinâmica de servidores. Ele realiza periodicamente o mapeamento de máquinas virtuais para servidores físicos de forma a atender mudanças de demanda, enquanto utiliza um conjunto mínimo de recursos físicos para reduzir custos em consumo de energia. No entanto, redefinir o mapeamento pode exigir migrar máquinas virtuais entre servidores físicos, o que pode acabar atrasando a reconfiguração das máquinas virtuais para a nova capacidade. Este trabalho apresenta algoritmos normalmente utilizados para consolidação dinâmica de servidores e avalia o seu impacto na métrica de atraso de reconfiguração usando diferentes tipos de *workloads* reais e sintéticos. Um algoritmo para consolidação dinâmica de servidores que controla o atraso máximo de reconfiguração decorrente de um novo mapeamento é proposto. Os experimentos realizados com esse algoritmo demonstram que ele provê maior controle sobre atrasos de reconfiguração e possui baixo impacto na quantidade adicional de recursos necessários.

Palavras-chave: Virtualização; Consolidação Dinâmica de Servidores; Migração de Máquinas Virtuais.

Dynamic server consolidation with controlled reconfiguration delays

ABSTRACT

Virtualization has reemerged in the last years as a solution to minimize costs in data centers due to resources subutilization. The possibility of dynamically changing virtual machines capacities and live-migrate them between physical servers resulted in a higher flexibility to attend sudden changes in computing demand, minimizing the amount of physical resources required. This increase in flexibility derived novel computing models such as utility computing and cloud computing. One of the key mechanisms in these virtualized data centers is dynamic server consolidation. It performs periodically the mapping of virtual machines to physical servers in order to attend changes in demand, while using a minimal set of physical resources to restrain power consumption costs. However, redefining a new mapping can require migrating virtual machines between physical servers, which eventually delay the reconfiguration of virtual machines to a new capacity. This work presents traditional algorithms used for dynamic server consolidation and evaluate their impact on the reconfiguration delay metric using different types of real and synthetic workloads. An algorithm for dynamic server consolidation which controls the maximum reconfiguration delay incurred by a new mapping is proposed. The experiments performed using this algorithm show that it provides higher control over reconfiguration delays and has a small impact in the additional amount of resources required.

Keywords: Virtualization; Dynamic Server Consolidation; Virtual Machines Migration.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1. Introduction

In the last years, virtualization has returned to the computing mainstream, after a long period of inactivity since its first appearance in the late 60's, and it seems that it has definitely reappeared to be included as an essential component in traditional computing architectures. Several software and hardware enterprises are driving forces to optimize their products for virtualization, even in areas where it was previously discredited by most experts[1],[2] [SOR07]. Virtualization is currently present in different contexts, such as: enabling the portability of applications through different platforms [LI98], making it feasible to execute concurrently different operating systems on the same hardware [RAM99, BAR03], providing a controlled and secure environment to test and analyze risky applications [HOO08, JIA06], promoting the efficient management of resources over computational grids [KEA04, KEA05], among others. It is also the driving technology for new computing models such as utility computing [RAP04] and cloud computing [BUY09].

One of the areas that virtualization presents considerable benefits is in data center management. Data centers provide an adequate environment for hosting computing resources and critical applications, in order to guarantee higher security, fault tolerance and performance. Before the resurgence of virtualization, data centers were directed towards guaranteeing these requirements using a physical isolation model, where each application was deployed on a different physical server. This approach was used to ensure that any issue happening with an application would not affect others. However, this model also presented drawbacks. In order to guarantee good performance for applications, physical servers were usually overprovisioned to attend application's peak utilization. However, since most applications do not present a stabilized peak utilization during all of its execution time, the servers were most of the time underutilized. Each server represents a fixed cost, covering power consumption, cooling and floor space, even when it is not being fully utilized. Studies show that the average utilization of regular data centers are in the order of 10% to 15% [VOG08]. It meas that a big slice of the resources are simply wasted, in order to guarantee the efficient isolation between applications, and the proper performance when peak utilization is necessary.

The difference between traditional data centers and virtualized data centers is that applications are deployed on virtual machines, instead of physical servers. The data center's resource management system defines the mapping of virtual machines to physical servers, being able to map more than one virtual machine to each physical server. In each physical server, the virtualization software provides a logic isolation between virtual machines, guaranteeing that security problems, faults or performance variations in one virtual machine do not affect others located in the same physical server.

Another advantage provided by virtualization is regarding virtual machines configuration. Instead of configuring each virtual machine to handle application's peak demand, the dynamic configuration feature of virtualization can be used to reconfigure the resources during execution whenever neces-

[1] Oracle and Virtualization at http://www.oracle.com/us/technologies/virtualization/index.htm.

[2] SQL Server 2008 Virtualization at http://www.microsoft.com/sqlserver/2008/en/us/virtualization.aspx.

sary. Therefore, when an application demands more CPU or memory capacity, the virtual machine can be transparently reconfigured to the required capacity. It avoids the overprovisioning of resources that were common in traditional data centers.

In cases that the virtual machine can not have its capacity increased due to capacity limitations in its current physical server, the virtual machine can be dynamically migrated to another physical server with enough capacity using a technique called live-migration. This technique allows migrating active virtual machines during application's execution. This technique can also be used to aggregate virtual machines placed in different physical servers to a fewer number of physical servers, in order to reduce the number of active resources and minimize the corresponding costs.

In order to keep up with the dynamic behavior of the applications, the changes in virtual machines capacities are periodically reevaluated and a new mapping of virtual machines to physical servers is generated, aiming at minimizing the amount of physical servers required. This process is called dynamic server consolidation, and it provides a direct reduction in data centers costs, maintaining active only the required resources based on actual applications demands.

Despite the higher flexibility available in virtualized data centers, the dynamic server consolidation process can have a direct impact on applications' performance. In the previous model, with overprovisioning, applications could use the additional required capacity whenever wanted. The additional capacity was already available to the application in the physical server. In virtualized data centers, changes in capacity require reconfiguring virtual machines capacities, and possibly migrating them between physical servers to attend the new demands. This process can result in significant delays to complete the reconfiguration, which can result in degradation of application's performance during this time. Therefore, guaranteeing minimal reconfiguration delay is important to assure that applications will obtain the required additional capacity on time.

This work investigates how dynamic server consolidation algorithms affect the reconfiguration delay of virtual machines, and proposes an algorithm that performs dynamic server consolidation considering the impact on the reconfiguration delay. The objective of the algorithm is to provide higher control over the reconfiguration delay, in order to guarantee that, for each new mapping generated by the algorithm, the reconfiguration delay of all virtual machines stays below a given threshold. When it is not possible to guarantee it, the algorithm should minimize the number of virtual machines that have reconfiguration delays higher than the specified threshold. Current works that deal with similar problems focus on the minimization of migrations, using only the individual migration cost of each virtual machine, without considering the network behavior when performing several migrations simultaneously. In this work, the max-min fairness model is used to simulate the concurrent utilization of the network by several migrations and to generate a better approximation on migration cost. The migration cost is used eventually to obtain the reconfiguration delay of each virtual machine.

The text is divided as follows. Chapter 2 presents a background of the main topics addressed in the work, such as: virtualized data centers, virtualization, and server consolidation, jointly with a review of the main works related to dynamic server consolidation. Chapter 3 describes the work's motivation, presents a detailed formalization of the problem, and proposes an algorithm that performs dynamic server consolidation with controlled reconfiguration delays. Chapter 4 presents an evaluation of the proposed algorithm and comparison with other algorithms used for dynamic server consolidation. Finally, Chapter 5 presents the conclusions of the work.

2. Background

This chapter presents a review on virtualized data centers, describing its main benefits over traditional data centers. It also describes its importance in new computing models such as utility computing and cloud computing. Considering that virtualization is the driving technology in virtualized data centers, an overview of the history of virtualization is presented, along with a description of the main virtualization techniques. After that, two important features of virtualization are presented: dynamic configuration and live migration. These features correspond to the base techniques used for server consolidation. An overview of server consolidation benefits, types and challenges is presented. It is also described the relation of the server consolidation problem with bin packing, and some common implementations using heuristics and linear programming are presented. Finally, a set of related works that correspond to the state of the art in dynamic sever consolidation are described, jointly with their main contributions.

2.1 Virtualized data centers

In the last years significant changes happened in data center infrastructure and provisioning model. The data center infrastructure has changed from a model where each application was deployed on a unique and specific physical server, to a model where applications are deployed on virtual machines, vanishing the tight dependency that existed between software and hardware. This gives the freedom for the resource manager to place each virtual machine in any available physical server. Virtual machines can even be instantaneously migrated between physical servers when required with insignificant downtime using migration techniques. Provisioning virtual machines can be performed orders of magnitude faster than when dealing with physical servers. When necessary, the capacity of a virtual machine can be dynamically changed, without requiring to shut it down, as required using physical servers. Virtual machines can be aggregated in fewer physical servers, with guaranteed isolation regarding performance, fault tolerance, and security.

All these changes resulted in opportunities to decrease internal costs, maintaining a cost-effective infrastructure, and provide an agile provisioning mechanism, in which virtual machines can be dynamically provisioned on demand, and resource capacity can be modified during execution to attend changes in application demands. The costs with a virtualized infrastructure scale in direct relation to the effective utilization of its resources, i.e., the number of active physical servers is determined by the current demand. All these changes enabled a much more cost-oriented and agile resource management in data centers. Jointly with this changes, a new computing model has emerged, the utility computing model [RAP04], which has virtualization technology as one of its key components [BUN06].

In the same way that happened decades ago with water, gas and electricity, computing is also being transformed in an utility service. Instead of using the “one-application-per-server” allocation model to increase reliability, and over-provisioning resources based on peak-load conditions to assure quality of service, utility computing is based on a shared and flexible infrastructure, where utilization of resources is improved and clients can increase or decrease the amount of resources based on its applications demands [AND02]. Moreover, utility computing decreases operational costs with power consumption and cooling, since the workload can be easily aggregated using only the necessary resources, letting unused resources, for example, in a power saving mode.

In utility computing, clients use computing resources when, and as much as wanted, and pay only for the amount consumed, using the "pay-as-you-go" model. The goal of utility computing is to provide resources in a straightforward and fast way to clients, with low or even without initial costs, and charge clients per resources utilization. This model is employed by the recent Cloud Computing model, and virtualized data centers represent the infrastructure layer of this model. Currently there are several enterprises that deliver infrastructure on demand, using the utility computing model, to be used in Cloud Computing scenarios. Some examples include: Amazon EC2 (Elastic Computing Cloud)[1] and Sun Grid Computing Utility [SUN07]. Virtualization is considered the driving technology in all these novel approaches to provide, consume and manage computing.

2.2 Virtualization

Virtualization aims at extending, through an indirection layer, the functionalities of a hardware or software resource. Some of these functionalities are: enable the interoperability between resources that do not implement the same communication interface, provide the multiplicity of a resource or unification of several resources, simulate resources that do not exist, provide a more efficient utilization of resources in a transparent manner, among others [SMI05]. Considering this concept, there are several types of virtual components, virtual devices, or even virtual machines.

A virtual machine is an abstract environment provided by a virtualization software called hypervisor, or virtual machine monitor (VMM). The platform used by the hypervisor is named host machine, and the module that uses the virtual machine is named guest. The hypervisor aims at provisioning virtual machines and performing the connection between virtual machines and the host machine. Besides, the hypervisor also abstracts host machine resources to be used by the guest through the virtual machine. Another important function is to provide the isolation between virtual machines in the same host machine, guaranteeing its independence from each other [NAN05].

The utilization of virtual machines started in the late 60's. At that time, computers were rare and very expensive. Besides, they could handle only one task at a time from a single user. Due to the increase in demand to use computers, it was necessary to develop a mechanism to allow concurrent utilization of the computer by several users and also guarantee the isolation between users applications. At that time, IBM launched the first computer with virtualization support, the mainframe 360/67. The machine had an abstraction layer in software called virtual machine monitor (VMM), which partitioned the hardware in one or more virtual machines [GOL74]. This software

[1] http://aws.amazon.com/ec2

was the VMM CP-67, which was implemented as a time-sharing system presenting to each virtual machine a complete System/360 computer. It performed the multiplexing of computer resources between virtual machines. Each user received a unique system by the VMM, sharing the same hardware, and providing an illusion of exclusive access to the real hardware [CRE81]. Later on, the VM/370 (Virtual Machine Facility/370) was developed including several optimizations. It executed in the System/370 Extended Architecture (370-XA) [GUM83], which had specific instructions to optimize the virtualization performance. In this system, each virtual machine was a System/370 replica.

In the 70's and 80's the first operating systems with multitasking support started to appear. Besides, the hardware costs started to decrease, facilitating the acquisition of new machines and reducing the need to share resources. Therefore, the need of a virtualization layer started to disappear, incurring in a simplification in the hardware regarding virtualization support.

In the 90's, due to the appearance of the personal computer (PC), there was once again a significant reduction in hardware costs, facilitating the acquisition of a huge amount of computers by enterprises [ROS04] and the utilization of the "one-application-per-machine" model to ensure higher fault tolerance, security, performance and availability. In the late 90's, some of the virtualization ideas reappeared, but with different goals. The development of the Java technology by Sun Microsystems, aimed in developing a virtual machine that could be executed over different platforms. Besides portability, the project also included security requirements, obtained through isolation between the application and the host machine. In the University of Stanford, researchers analyzed the utilization of virtual machines to facilitate the utilization of traditional operating systems in massively parallel processing (MPP) machines, in order to facilitate the programming of these machines. The goal using virtualization was to transform a MPP machine into a machine with traditional architecture. This project originated the company VMware Inc..

Currently, the utilization of virtualization is becoming widespread again [ROS05]. Several companies are emerging to develop virtualization technology (e.g., Citrix Systems, Inc. [2], VMware Inc. [3], Parallels [4]), and the main hardware companies are modifying their products to enable the efficient support of virtualization (e.g., Intel-VT [NEI06, UHL05], AMD Pacifica [AMD05]).

One of the main reasons for the reappearance of virtualization is due to the low utilization of resources. The decrease in hardware costs leveraged the acquisition of several resources, enabling the utilization of one application per machine. However, most of the time these resources are underutilized. Studies indicate that data centers have an average utilization of 10% to 15%. The resulting problems of this approach are the inefficient utilization of floor space, waste in power utilization and high management cost. Virtualization aims at reducing this problem, consolidating servers in a lower number of machines, reducing the required floor space and power costs, while guaranteeing efficient isolation between the virtual machines in the same host. Other advantages include: facili-

[2] http://www.citrix.com

[3] http://www.vmware.com

[4] http://www.parallels.com

ties in hardware maintenance, dynamically reconfiguration of virtual machines on demand, and high availability support. It also provides higher flexibility to manage virtual machines (virtual machines can be easily paused, resumed, reconfigured, cloned, migrated, etc).

There are different techniques available to implement virtualization. The technique considered in this work is called platform or system virtualization, which is the same type of virtualization provided in the 60's by the first virtualization software. It aims at providing a complete environment in which is possible to execute a complete operating system, as well as all processes executed on the operating system. The Virtual Machine Monitor (VMM) or hypervisor enables the access to all hardware resources by the guest operating system. The VMM stays right on top of the hardware and is executed in privileged mode, while the guest operating system executes in lower priority. It intercepts the interaction between the guest operating system and the hardware, and provides controlled access to the resources provided by the hardware.

One of the main goals of this virtualization type is to provide platform replication, enabling the concurrent execution of different operating systems in the same host. Virtual machines stay active during the operating system execution and can be turned on or off without interfering in the physical server functioning, or with the other virtual machines executing in the same host.

The VMM must be able to intercept all privileged instructions performed by the virtual machine in order to give the appropriate treatment. Therefore it requires processor support in order to result in efficient functioning [POP74]. There are basically 2 approaches to perform this interception: full virtualization and paravirtualization.

In full virtualization [ROS04] all instructions performed in the virtual machine are analyzed by the VMM to give the proper treatment when a privileged operation is executed. This type of virtualization enables the utilization of unmodified operating systems. An example of software that uses full virtualization is the VMware ESX Server[5].

In paravirtualization [ROS04], the guest operating system is modified in order to perform calls directly to the VMM when a privileged instruction needs to be executed. This restriction prevents the utilization of unmodified operating systems. Although the operating system is modified, the libraries and applications running on top of the operating system do not need to be modified. An example of software that uses paravirtualization is the Xen hypervisor [BAR03].

Paravirtualization was first developed in order to provide much better performance than full virtualization, using the hardware available at that time. However, with the advances in processor development with virtualization support from Intel (Intel-VT [UHL05, NEI06]) and AMD (AMD Virtualization [AMD05]), a hardware-assisted virtualization model is taking place, which aims at decreasing the performance lag between paravirtualization and full virtualization, enabling the efficient execution of non-modified operating systems.

Another common type of virtualization is the operating system-level virtualization [NAN05]. It provides a replication of the operating system using the same kernel. Each virtual machine is called a partition and has an isolated set of resources from the system. The kernel manages the

[5] http://www.vmware.com

virtual isolation between partitions. The drawback of this approach is the restriction of each virtual machine being a replica of the same kernel. However, it results in performance very similar to the non-virtualized system, and also provides higher scalability due to the low cost of partitioning. Examples of this virtualization type include Linux VServer [DL05], Parallels Virtuozzo Containers [PAR10], FreeBSD Jails [NN05], OpenVZ [KOL06], and OpenSolaris Zones [TUC04].

2.2.1 Dynamic configuration

The possibility to dynamically change the capacity of a virtual machine is one of the great benefits of virtualization. In the past, each machine had a fixed capacity. The only way to attend changes in resources demands by an application located in a physical server, was to change the physical server's capacity including additional resources (e.g., memory, disk, network) or reinstalling the application in a new physical server with higher capacity (usually when higher CPU capacity was required). In both cases, it was necessary to shutdown the application, perform the change, and turn it up again hoping that everything works as expected. Due to this low flexibility to attend changes in computing demand, most physical servers were overprovisioned to applications, in order to attend a estimated peak demand with acceptable performance. Since this estimated peak demand was only achieved in rarely occasions, most of the time the machine was underutilized.

With virtualization, the capacity of a virtual machine can be dynamically changed to attend increases in computing demand, and decreased again when low utilization is presented. If the physical server hosting the virtual machine can not handle an increase in capacity, the virtual machine can be easily migrated to another physical server with available capacity, with negligible downtime using the live migration technique. The approach used by virtualization software to enable dynamic configuration varies with each resource type. Some typical approaches used are:

Processor configuration The virtual machines are assigned Virtual CPUs, or VCPUs for short. The VMM manages the creation, capacity configuration and mapping of VCPUs to physical CPUs (or CPU cores). VCPUs can be created and configured dynamically. The dynamic configuration affects the functioning of the VCPU scheduler (e.g., BVT [DUD99], SEDF or Credit), inside the VMM, which provides the sharing of physical CPUs to the VCPUs according to each VCPU current configuration. VCPUs can also be associated exclusively to use one or more physical CPUs. This process is called pinning and provides higher performance to the virtual machine, since it won't be preempted by other virtual machines having VCPUs assigned to the same CPU.

Memory configuration Memory configuration is usually performed using a technique called ballooning [WAL02]. This technique enables the dynamic increase or decrease of the memory assigned to a virtual machine in order to attend new demands. When more memory is required, the "balloon driver" inside the virtual machine is instructed to "inflate" allocating more physical pages to the virtual machine. In the same way, the "balloon driver" can deflate to deallocate pages.

Network configuration Since the configuration of virtual network interfaces in the virtual machine is performed using regular network tools, the configuration of network capacity is also performed using regular tools for traffic shaping and packet filtering such as htb and iptables.

Storage configuration Configuring storage is usually performed using mechanisms such as the Logical Volume Management (LVM). It permits the utilization of logic volumes as virtual machine storage. This mechanism enables the dynamic increase or decrease of a volume's size, besides the support for snapshots and creation of copy-on-write volumes.

The configuration of a virtual machine directly affects its performance. There are several works that use the dynamic configuration feature of virtual machines to attend a determined quality of service in specific applications. These works are based on a feedback control model [LIU04, WAN05, ZHU06, PAD07], which analyzes application performance metrics and change virtual machines capacities to cope with application demands, or a prediction model [XU06], forecasting the future application behavior and adjusting virtual machines capacity accordingly. However, they perform only local configuration, and migration is not considered during dynamic configuration.

2.2.2 Live migration

There are basically two types of migration: stop and copy, and live migration. In the stop and copy approach, the virtual machine is paused, a copy of its current state is saved to disk, the current state is transferred to another physical server, and the virtual machine is resumed in the new physical server using the transferred state. Live migration is a powerful feature provided by most virtualization solutions [NEL05]. It enables moving a running virtual machine from one physical server to another with negligible downtime.

The live migration technique also copies the virtual machine current state to the destination physical server, but the virtual machine stays online during most of the time, resulting in a downtime of tens of milliseconds (e.g., downtime using the Xen hypervisor is from 30 to 60 milliseconds). In this case the unavailability of the virtual machine is not noticed by its users. The live-migration procedure is described by the following steps [CLA05]:

1. A destination physical server, previously chosen to host the virtual machine, reserves space to receive the virtual machine. In case any problem happens, the virtual machine is not migrated and keeps its execution in the source physical server.

2. During each iteration, memory pages from the virtual machine are transferred from source to destination physical server. In the first iteration all pages are transferred. In the subsequent iterations only the modified pages since the last iteration are transferred. This process repeats until there is only a small percentage of dirty pages to be transferred or the number of transfers attempts for this migration have reached a specified limit. These parameters - percentage of dirty pages and number of transfer attempts - are defined in the virtualization software.

3. In this stage the downtime period starts. The virtual machine is suspended in the source physical server and the network traffic is redirect to the destination physical server. The CPU state and remaining dirty pages are transferred to the new physical server. At this point, both physical servers can continue to execute the virtual machine, i.e., in case a fault happens, the source physical server can still keep the virtual machine running.

4. The destination physical server acknowledges the complete virtual machine reception to the source. The source accepts and releases its allocation of the virtual machine. From this point, the destination physical server is the current host for the virtual machine.

5. The virtual machine is activated. The device drivers and network are reconfigured in the new host.

When a virtual machine migrates, its MAC and IP addresses migrate with the virtual machine. Therefore, it is only possible to migrate virtual machines using the same level 2 network and IP subnet. Currently there is no support to provide automatic remote access to the file system stored in the source physical server when a virtual machine is migrated. Hence, a centralized storage solution (e.g., SAN - Storage Area Network, or NAS - Network Attached Storage) is currently required to guarantee that the file system is available on all physical servers.

2.3 Server Consolidation

Server consolidation is a key feature in current virtualized data centers. It uses the dynamic configuration and live migration features of virtualization to assign virtual machines to physical servers minimizing the number of physical servers required. In order to reduce costs, unused resources can be turned off, or stay in a low power consumption mode. Server consolidation improves resources utilization and decreases operational costs. It is currently one of the main reasons for the widespread utilization of virtualization, due to its directly relation to costs reduction in enterprises' data centers. Besides, current studies indicate that resources in data centers are usually underutilized [AND02], favoring the utilization of consolidation. The dynamic configuration feature enables that virtual machines be configured on demand to the required capacity. Live migration enables that virtual machines be transferred between physical machines with negligible downtime. It can be used when a physical server can not handle the increase in capacity of a virtual machine, or when emptying a physical server, moving its virtual machines to other physical servers, in order to reduce the number of physical servers required to handle all virtual machines. The main benefits of server consolidation are: improvement in resource efficiency and utilization, reduction in data center footprint, reduction in power consumption and environmental impact for an IT organization. Server consolidation can be classified in two different types [BOB07]:

Static server consolidation Static server consolidation considers that each virtual machine has a pre-defined fixed capacity, and aims to map the virtual machines to physical machines in order to not overload physical machines' capacities and minimize the number of physical servers used. Virtual machines capacities are usually defined through observations of historical average resource utilization. After initial static server consolidation the mapping may not be recomputed for long periods of time, such as several months. The mapping change process is usually performed off-line, i.e., all virtual machines are suspended, reassigned to new physical servers, and resumed.

Dynamic server consolidation In the dynamic server consolidation, the changes in virtual machines demands are used to periodically remap virtual machines to physical servers in order to reduce the number of physical servers. Unused physical servers are turned off or put in a low power consumption mode. Each consolidation event happens on shorter timescales, preferably shorter than the periods of significant variability of the virtual machines demands. The virtual machines are dynamically migrated between physical servers when required using the live-migration technique, in order to avoid service interruption.

Some of the challenges in server consolidation include: defining the correct periodicity to remap virtual machines, performing an accurate forecasting of virtual machine capacity during the period till the next consolidation event, decrease the intrusiveness of remapping virtual machines, attend specific constraints, such as: avoid mapping a group of virtual machines to the same physical server due to security issues, or try to map together virtual machines that communicate with each other in order to decrease network latency.

The server consolidation problem, which consists in mapping a set of virtual machines with different capacities to a set of physical servers in order to minimize the number of physical servers required, is directly related to the classic problem of bin packing. There are several approaches in the literature to solve this problem, in which the most common use heuristics and linear programming. In the next sections the bin packing problem is reviewed, and its relation to the server consolidation problem is presented. Common heuristics used to solve the problem are described, along with a linear programming formulation.

2.3.1 Bin packing problem

Given a sequence $L = (a_1, a_2, \ldots, a_n)$ of items, each with a size $s(a_i) \in (0, 1]$, and subsets $B_1, B_2, \ldots, B_m$ of unit-capacity bins, the bin-packing problem aims at finding the best packing of items into bins such that the number of required bins is minimal (guaranteeing that the sum of the items sizes allocated to each bin is less than or equal the bins capacity, i.e., $\sum_{a_i \in B_j} s(a_i) \leq 1$ for $1 \leq j \leq m$) . The bin-packing problem is classified as an NP-hard problem [COF96].

The bin-packing problem is used in several areas, such as: determining the rectangular components that will be cut from material sheets in wood and glass industries, placing goods on shelves in warehousing contexts, arranging articles and advertisements into newspaper pages, among others. The server consolidation problem can also be seem as an application of the bin-packing problem, where the virtual machines represent the items and the physical servers, the bins.

The bin-packing problem presents several variations. One of these variations is called the multi-dimensional bin packing problem. This variation represents items and bins sizes as a tuple of values. This variation fits into a common virtualized data center scenario since each physical server capacity and virtual machine demand can be represented by a tuple of resources quantities, such as CPU, memory, network, storage, etc.

The server consolidation problem can also be classified as an offline variation of the bin-packing problem since all virtual machines demands are know before start packing into the physical servers. On the contrary, the on-line variation of the bin-packing problem states that each item should be sequentially packed, with no knowledge about the subsequent items.

Common implementations of the bin-packing problem use techniques such as heuristics and linear programming.

Heuristics

Heuristics is one of the most common techniques used to solve the bin-packing problem, and consequently can also be used to solve the server consolidation problem. The main goal of heuristics is to find good solutions at a reasonable computational cost, however it does not guarantee optimality.

There are several heuristics for the general bin packing problem. Some of the most commonly used are: next-fit, first-fit, best-fit, worst-fit and almost worst-fit. Each heuristic receives a set of bins and items with respective capacities and returns a feasible packing of items to bins. The strategy used by each heuristic is presented as follows [COF96, KOU77]:

Next-fit The next-fit heuristic keeps only one bin opened each time. Each item is packed to the open bin if it has enough spare capacity to hold the item, otherwise the bin is closed, and a new empty bin is opened to pack the item.

First-fit The first-fit heuristic searches available space through all bins and packs each item to the first bin with enough capacity to hold the item.

Best-fit In the best-fit heuristic, each item is packed to the bin that can hold the item leaving the lowest remaining space in the bin.

Worst-fit In the worst-fit heuristic, each item is packed to the non-empty bin that can hold the item leaving the highest remaining space in the bin. If there is no feasible bin, it allocates a new bin and packs the item to this bin.

Almost worst-fit Each item is packed to the non-empty bin that can hold the item and is the second bin that leaves the highest remaining space in the bin. If there is only one feasible bin, it packs the item to this bin. If there is no feasible bin, it allocates a new bin and packs the item to this bin.

These heuristics process each item without knowing anything about the other items, therefore they are usually used in the online bin packing problem. In the offline bin packing, when all items are know in advance, it is possible to perform a slight optimization. Before applying the strategy, the items are sorted according to its size in decreasing order before start packing. Considering that in the server consolidation problem the item and bin sizes are represented by tuples, it is necessary to define the ordering model that should be used. In this specific scenario, the sorting is based on the lexicographic order [KOU77], which stands that, given two ordered sets $A = \{a_1, a_2\}$ and $B = \{b_1, b_2\}$, $A < B$ if $a_1 < b_1$ or $a_1 = b_1$ and $a_2 < b_2$. The lexicographic order can be readily extended to sets of arbitrary length by recursively applying the same definition. After sorting the items, each heuristic applies its own strategy to pack the items to bins. This optimized version of the heuristics are called: next-fit decreasing, first-fit decreasing, best-fit decreasing, worst-fit decreasing, and almost worst-fit decreasing. They usually provide better results than their online versions.

Linear programming

Another common technique used to solve the bin-packing problem is linear programming. The main benefit of linear programming over heuristics is that it results in an optimal solution. However, it has much higher requirements in terms of computing power and time.

Linear programming is a technique used to solve optimization problems, which consists of minimizing or maximizing an objective function expressed in terms of a linear equation and a set of linear and non-negativity constraints. It finds the optimal set of values to be used in the decision variables (unknown values) which minimize or maximize the objective function.

Linear programming problems can be solved using different methods, such as the simplex, interior point, and branch and bound methods. Linear programming problems usually rely on a linear programming solver to be solved. Examples of academic and commercial solvers are: SCIP [ACH04] and CPLEX [ILO07].

In linear programming, the decision variables are real numbers. When dealing with only integer decision variables or a mix between real and integer decision variables, it is called integer linear programming (or integer programming), and mixed integer linear programming (or mixed integer programming) respectively. These variations usually present new obstacles and require different methods to be efficiently solved.

The liner programming formulation of the server consolidation problem requires only integer decision variables. Therefore it is classified as an integer linear programming problem. It has as parameters the sets P, V, and R, representing respectively the set of physical servers, virtual machines and resources; and arrays c_j^k and u_i^k, representing respectively the capacity of physical server j over resource k, and the demand of virtual machine i over resource k. The decision variables of the problem are: a binary array x_{ij} representing the mapping of virtual machines to physical servers (x_{ij} is equal to 1 if virtual machine i is mapped to physical server j, and 0 otherwise), and a binary array y_j representing the utilization of physical server j in the mapping. Figure 2.1 shows the formulation of the problem. The objective function aims at minimizing the total number of physical servers used. The first constraint guarantees that each virtual machine is mapped to only one physical server. The second constraint deals with physical server capacities, guaranteeing that the sum of the demands of each virtual machine mapped to a physical server for a given resource do not exceed physical server's capacity on this resource. The third constraint is used to set the variable y_j to 1 if the physical server is used, and 0 otherwise.

$$\boxed{\begin{array}{l}
minimize \sum_{j \in P} y_j \\
\text{s.t.} \\
\sum_{j \in P} x_{ij} = 1 \quad \forall i \in V \\
\sum_{i \in V} u_i^k * x_{ij} \leq c_j^k \quad \forall j \in P, \forall k \in R \\
\sum_{i \in V} x_{ij} \leq y_j * |V| \quad \forall j \in P \\
\\
x_{ij} \in \{0,1\} \quad \forall i \in V, \forall j \in P \\
y_j \in \{0,1\} \quad \forall j \in P
\end{array}}$$

Figure 2.1 – Dynamic server consolidation using integer programming.

2.4 Related Work

Server consolidation techniques have widespread adoption in virtualized data centers, however the process of mapping virtual machines to physical servers is not trivial. Depending on the application requirements and the resource provider goals, different strategies can be applied. A review of the main works regarding server consolidation and its main contributions is presented next.

In [AND02], the authors propose static and dynamic server consolidation algorithms. The algorithms are based on integer programming and genetic algorithms techniques. The algorithms are evaluated using a production workload containing traces from enterprise applications. The results present the benefits of using server consolidation, showing that the same workload could be allocated in a much smaller number of physical servers. The genetic algorithm resulted in solutions as good as with integer programming, with the benefit of reaching the solution much faster. In this work, migrations are considered to happen instantaneously, i.e., there is no migration cost included in the algorithms.

In [SPE07, BIC06], the authors present linear programming formulations for the static and dynamic server consolidation problems. Some extension constraints are also proposed, such as: limiting the number of virtual machines in a physical server, guaranteeing that some virtual machines are assigned to different physical servers, trying to aggregate a set of virtual machines in the same physical server, mapping some virtual machines to a specific set of physical servers that contain some unique attribute, and limiting the total number of migrations performed. Due to the high cost of solving the linear programming formulations, an LP-relaxation based heuristic is proposed. The heuristic works in two phases. In the first phase, the model is solved using a linear programming relaxation, enabling virtual machines to be fractionally assigned to servers. In the second phase, a branch and bound algorithm is used to assign the fractioned virtual machines in the first phase to physical servers. This approach reduces significantly the total time necessary to obtain solutions as good as the ones obtained using linear programming.

In [BOB07], the authors propose a dynamic server consolidation algorithm which focus on minimizing the cost of running the data center. The cost is measured using a penalty over underutilized and overloaded physical servers, and over service level agreements (SLA) violations, defined as CPU capacity guarantees. The algorithm is based on three phases: i) measuring historical data, ii) forecasting the future demand, and iii) remapping virtual machines to physical servers. The first phase is used to identify the workloads that can be most benefited from using the algorithm, i.e., workloads that can be efficiently predicted in order to provide a good forecasting of its behavior. The algorithm is executed periodically in successive intervals, and minimizes the number of required physical servers to support the virtual machines, guaranteeing that a defined percentage of physical servers do not become overloaded during the interval. The algorithm is evaluated using production workload traces.

In [KHA06], the authors propose a dynamic management algorithm, which is triggered when a physical server becomes overloaded or underloaded. When a physical server becomes overloaded, the algorithm migrates the virtual machine with lowest utilization to a physical server with least remaining capacity, big enough to hold the virtual machine. If there is no feasible physical server, a new one is allocated. This process repeats until the physical server is left in an average utilization rate. When a physical server becomes underloaded, it selects a virtual machine with lowest utilization across all physical servers, and tries to migrate it to a physical server with minimum remaining capacity, enough to hold the virtual machine. This process repeats until there is no more benefits in migrating virtual machines. The algorithm has the following objectives: i) guarantee that SLAs are not violated, ii) minimize migration cost, iii) optimize the residual capacity of the system, and iv) minimize the number of physical machines used. The SLAs are specified in a relation between application metrics, such as response time and throughput, and resource utilization.

In [WOO07], the authors propose a management algorithm that monitors virtual machines utilization and detect hotspots using time-series prediction techniques. When an overload situation is discovered, it determines the required capacity of each virtual machine and generates a new mapping of virtual machines to physical servers. The new mapping aims at minimizing the total migration time, choosing to migrate virtual machines that reduce the amount of data copied over the network from overloaded servers to underutilized servers. The work focus on overloaded situations, but does not act when the system becomes underutilized.

In [VER07], the authors propose an architecture, called pMapper, with server consolidation algorithms that takes into account power and migration costs in addition to the performance benefit when placing applications to physical servers. The consolidation is performed on a heterogeneous virtualized server cluster. Migration costs are defined according to virtual machine characteristics.

The results obtained from the related works show that dynamic server consolidation provides several gains when used to consolidate virtual machines in data centers. Most of the related works also deal with other constraints, such as: application SLAs, migration costs, and power consumption costs. These additional constraints aim at guaranteeing good performance to the applications executing in the virtual machines, while also minimizing the overall costs required to maintain the virtual machines running.

3. Dynamic server consolidation with controlled reconfiguration delays

This chapter presents the proposed work of dynamic server consolidation with controlled reconfiguration delays. Initially the motivation of the work is presented. An overview of common approaches used by related works is presented, together with a description of the different perspective analyzed by the proposed work, regarding controlling the costs due to reconfiguration delays in dynamic server consolidation. After this, a detailed description of the dynamic server consolidation with controlled reconfiguration delays problem is presented. Finally, a proposal of an algorithm that solves the problem is described.

3.1 Motivation

Dynamic server consolidation is an essential procedure in efficient virtualized data centers. It provides several benefits including: decrease in power consumption, efficient use of floor space, higher resources utilization, and adaptability to variations in workloads demands. Current works propose different alternatives to perform the best mapping of virtual machines to physical servers in order to minimize the amount of physical servers required. However, other important goals are also considered, such as: i) predicting workloads demand in order to pro-actively change virtual machines configuration [KHA06, WOO07, BOB07]; ii) minimizing migration costs [KHA06, WOO07, VER07]; and iii) minimizing power consumption costs [VER07].

Despite the clear benefits of dynamic server consolidation in overall costs reduction, it also presents a direct impact on applications performance. It is responsible for adequating virtual machines capacity, so applications can perform accordingly to the quality of service expected, without underutilizing or overloading resources. However, in order to attend the changes in capacity demand, migrating some virtual machines might be required.

Live-migration provides an efficient and transparent method to relocate virtual machines, since the virtual machine remains available all the time. However, the amount of time required for migration completion can not be ignored. It has a direct influence on the time it takes to reconfigure virtual machines capacity, and, consequently, attend changes in application demands. A large migration delay can represent a significant penalty in application's performance. For instance, considering the example in Figure 3.1 which represents a graph with an application's demand over time, and its corresponding virtual machine's requested and provided capacities. It shows that an increase in virtual machines capacity is requested in order to attend an increase in demand at t_1, which corresponds to the beginning of a consolidation event. However the change occurs only at time t_2. This time is defined as reconfiguration delay. During this time the virtual machine stays in an overload mode, resulting in application performance degradation. The gray area in the graph show the amount of resource demanded but not available to the virtual machine.

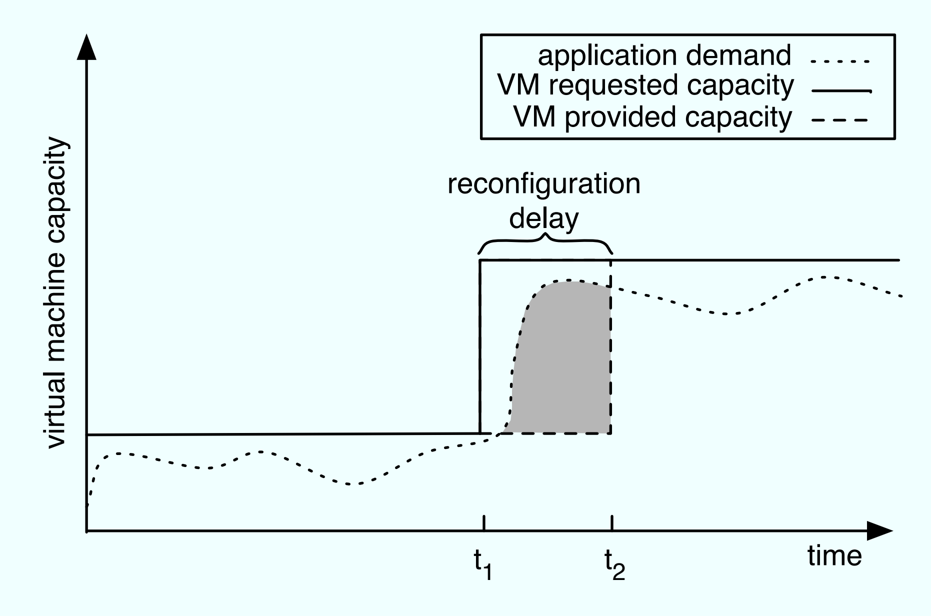


Figure 3.1 – Impact of reconfiguration delay.

Despite of being directly related to the migration cost, the reconfiguration delay also applies for virtual machines that are changing their capacities in their local physical server. For example, in order to increase the capacity of a virtual machine locally, the physical server might need to dump some of its virtual machines, in order to release some space for the increase of virtual machine's capacity. Therefore, the reconfiguration delay consists in the time it takes to migrate the virtual machines out. As observed, in either way the reconfiguration delay depends directly on the migration cost. The migration cost represents the time to complete a migration, and depends on several parameters, such as: network characteristics and capacity, amount of data required to migrate the virtual machine, and other transfers occurring at the same time.

Consequently, the reconfiguration delay has a direct impact on applications' performance, since it represents the duration of time that it will need to wait in order to have the requested capacity attended. The reconfiguration delay depends directly on the amount of time required to perform migrations in the virtualized data center, which depends, besides virtual machines capacities, on the local network infrastructure and the available capacity. Therefore, controlling the maximum reconfiguration delay in dynamic server consolidation can result in better applications performance and a more efficient virtualized data center.

Related works that address similar problems, focus only on the migration costs, minimizing the amount of memory transferred or the amount of migrations [KHA06, WOO07, VER07], but without any analysis regarding migrations completion time. Based on the studies analyzed, there is currently no work that considers network costs and aim at providing guarantees over the reconfiguration time for virtual machines. Therefore, this work aims to investigate how current approaches behave in relation to the reconfiguration delay, and propose a dynamic server consolidation algorithm which provides guarantees regarding maximum reconfiguration delays in a virtualized data center scenario.

3.2 Problem formalization

The scenario proposed is a virtualized data center composed by a set of virtual machines, which are mapped to a set of physical servers. Periodically a new consolidation event is performed. Changes in virtual machines demands are analyzed and a new mapping of virtual machines to physical servers is derived. Based on this new mapping, virtual machines are migrated between physical servers. The network topology used to connect the physical servers is a star. Each physical server is connected to a crossbar switch using bidirectional links.

In each consolidation event, the goal is to generate a new mapping which attends virtual machines demands, minimizes the number of required physical servers, and guarantees that the reconfiguration delay for each virtual machine is below a given threshold. When the last constraint is not possible, the new mapping should minimize the number of virtual machines which have their reconfiguration delays above the given threshold. Table 3.1 presents a description of each term used in the problem formalization.

Let P be the set of physical servers in the virtualized data center. The capacity of each physical server j is given by c_j^k, where $k \in R$. Set R represents the resources used to identify physical servers' capacity and virtual machines' demand. In this scenario $R = \{cpu, mem\}$, i.e., CPU and memory resources are used to characterize physical servers and virtual machines. Let V represent the set of virtual machines, and u_i^k be the new demand of resource $k \in R$ for virtual machine $i \in V$. Each consolidation event might use different values of u_i^k for the virtual machines representing the variations in virtual machines demand.

The current mapping of virtual machines to physical servers is given by a binary matrix x'_{ij} which is equal to 1 if virtual machine i is mapped to physical server j, and 0 otherwise. The new mapping that should be generated for the actual consolidation event, and which considers the changes in virtual machines demands, is represented by the binary matrix x_{ij}, being equal to 1 if virtual machine i is mapped to physical server j, and 0 otherwise. The mapping x_{ij} should respect the following constraints: i) each virtual machine i is mapped to a single physical server j, i.e., $\sum_{j \in P} x_{ij} = 1$ for all $i \in V$, and ii) the sum of the demands u_i^k of virtual machines i mapped to a physical server j do not overload its capacity c_j^k, i.e., $\sum_{i \in V} x_{ij} * u_i^k \leq c_j^k$ for all $j \in P$, and for all $k \in R$.

When the physical server mapped to a virtual machine in the new mapping is different from the physical server defined in previous mapping, the virtual machine should be migrated from the source physical server to the destination physical server. Let $M = \{i \mid x_{ij} \neq x'_{ij} \, and \, i \in V \, and \, j \in P\}$ be the set of virtual machines that should be migrated. For each virtual machine $i \in M$, source and destination physical servers are represented by src_i and dst_i respectively.

The network is represented by a directed graph $G = (N, A)$, where set N is a set of nodes composed by the set P of physical servers and the crossbar switch, represented by sw, which is connected to all machines to form the star topology, i.e., $N = P \cup \{sw\}$. Set A contains tuples (s, d, bw), in which each tuple represents a unidirectional link between a source s and a destination

Term	Description
P	set of physical servers
V	set of virtual machines
R	set of resources (e.g., CPU, memory, network)
c_j^k	capacity of resource $k \in R$ for physical server $j \in P$
u_i^k	demand of resource $k \in R$ by virtual machine $i \in V$
x'_{ij}	current mapping, equal to 1 if virtual machine $i \in V$ is mapped to physical server $j \in P$, and 0 otherwise
x_{ij}	mapping to be defined for the current consolidation event, equal to 1 if virtual machine $i \in V$ is mapped to physical server $j \in P$, and 0 otherwise
M	set of virtual machines that should be migrated according to the mapping defined by x_{ij}
src_i, dst_i	source and destination physical servers for virtual machine $i \in M$
$G = (N, A)$	graph representing the network composed by N nodes and A arcs
sw	represents a switch that interconnects all physical servers and is one of the elements of N
(s, d, bw)	represents of a directed link $\in A$, from physical server $s \in P$ to physical server $d \in P$ with bandwidth bw
$migbw_i$	available bandwidth to migrate virtual machine $i \in V$ from src_i to dst_i based on the max-min fairness model
$u_i'^{mem}$	current amount of memory allocated for virtual machine $i \in V$
$migdelay_i$	migration delay of virtual machine $i \in V$
$recdelay_i$	reconfiguration delay of virtual machine $i \in V$
y_j	indicates the utilization of physical server $j \in P$ in the new mapping, equal to 1 if j is used, and 0 otherwise
σ	threshold defined as the maximum reconfiguration delay
$rdbreak_i$	indicates a reconfiguration delay break for $i \in V$, equal to 1 if $recdelay_i > \sigma$, and 0 otherwise
α	penalty included for each reconfiguration delay break

Table 3.1 – Terms used in the problem formalization.

d with bandwidth capacity bw. Since the network topology proposed considers bidirectional links, for each physical server there are two tuples, one from the physical server to the crossbar switch and other in the opposite direction, i.e., $A = \{(j, sw, bw) \mid j \in P \, and \, bw \in \mathbb{R}\} \cup \{(sw, j, bw) \mid j \in P \, and \, bw \in \mathbb{R}\}$.

The flow control used to model the transfers (migrations) in the network is the max-min fairness model [IOA05, NAC08]. This model is often considered in the context of IP networks carrying elastic traffic. It presents the following properties: i) all transfers have the same priority over the available bandwidth, ii) link bandwidths are fairly shared among transfers being allocated in order of increasing demand, iii) no transfer gets a capacity larger than its demand, and iv) transfers with unsatisfied demands get an equal share of the link bandwidth.

Given a set of network links with respective bandwidths and the links used by each migration, it is possible to obtain the available bandwidth for each migration using a progressive filling algorithm which respects the MMF model properties. The algorithm used is the one presented by [BER92]. The algorithm initializes the bandwidth available for each transfer with 0. It increases the bandwidth for all transfers equally, until one link becomes saturated. The saturated links serve as a bottleneck for all transfers using them. The bandwidth for all transfers not using these saturated links are incremented equally until one or more new links become saturated. The algorithm continues, always equally incrementing all transfer bandwidths not passing through any saturated link. When all transfers pass through at least one saturated link, the algorithm stops. Figure 3.2 presents an example of the MMF model. The example presents a scenario with five nodes (A, B, C, D and SW). Each node is connected to SW using two links, each one with unitary bandwidth capacity. There are 5 transfers that are taking place: V1 from A to B, V2 from A to C, V3 and V4 from D to C, and V5 from B to D. In this example, transfers V2, V3 and V4 will use each one 1/3 of the available capacity, all having link l8 as bottleneck. Transfer V1 will use 2/3 of the available capacity, since it shares link l1 with V2 which is using 1/3 of the capacity. Finally, transfer V5 uses the full capacity since it does not share links l3 and l6 with any other transfer.

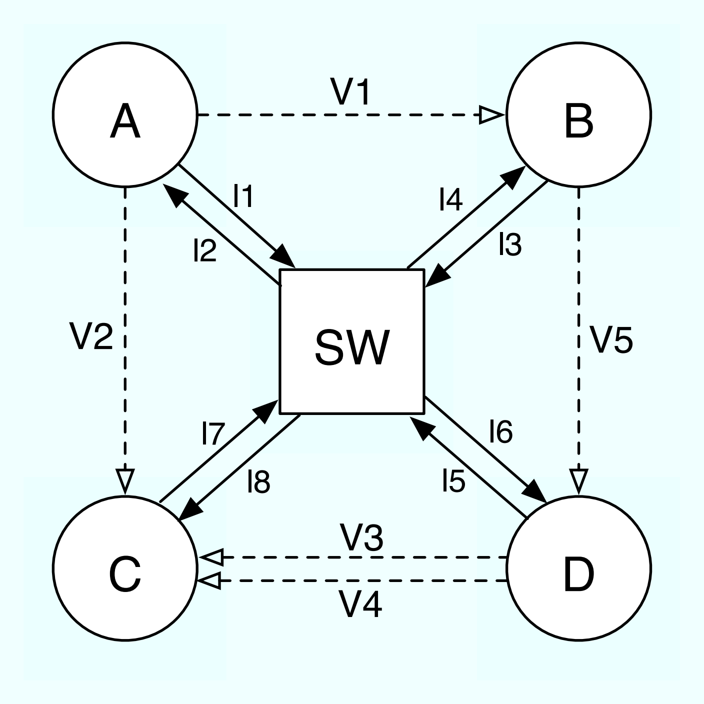


Figure 3.2 – Example of MMF flow model

The migration of a virtual machine $i \in M$ is performed using a single-path routing, which is composed by two links, one from the source physical server to the switch, $(src_i, sw, bw) \in A$, and other from the switch to the destination physical server $(sw, dst_i, bw) \in A$. Applying the available links with bandwidths A and the used links for each migration in the MMF model, it is obtained for each migrating virtual machine i the available bandwidth to perform the migration which is defined as $migbw_i$.

The migration delay is measured as the time it takes to transfer the current memory allocation of virtual machine i, which is given by $u_i'^{mem}$ [1] using the available bandwidth $migbw_i$. However, it is not possible to simply divide one by the other. Considering that some migrations finish before others, the available bandwidth for each migration can change, and the remaining amount of memory to be transferred should take into account the new available bandwidth. Therefore, the migration delay is measured in incremental steps.

Algorithm 1 presents the measurement of migration delays. After each migration finishes, the amount of time passed is added to the migration delay of all virtual machines, and the amount of memory transferred during this time is decreased from the total amount of memory to transfer. If there is no more memory to transfer, the migration is removed from the set of running migrations. The algorithm continues until there are no more running migrations.

Algorithm 1 Migration delay measurement

```
$migdelay_i \leftarrow 0$    for all $i \in M$ {initialize the migration delay}
$mem_i \leftarrow u_i'^{mem}$    for all $i \in M$ {initialize the amount of memory to be transfered}
$RM \leftarrow M$ {initialize the set of active running migrations}
while $RM \neq \emptyset$ {while there are running migrations} do
    $\{mb_i \mid i \in RM\} \leftarrow MMF(RM, A)$ {obtain the available bandwidth to transfer each migration according to the MMF model}
    $minmigdelay \leftarrow min(\{mem_i/migbw_i \mid i \in RM\})$ {get the minimum migration delay}
    $mem_i \leftarrow mem_i - (minmigdelay * migbw_i)$    for all $i \in RM$ {update memory transfered according to minimum migration time}
    $migdelay_i \leftarrow migdelay_i + minmigdelay$    for all $i \in RM$ {update migration time}
    $RM \leftarrow \{i \mid i \in RM \, and \, mem_i \neq 0\}$ {remove completed migrations}
end while
```

In the end, for each migrating virtual machine $i \in M$ there is a corresponding migration delay defined as $migdelay_i$ using the current mapping x_{ij}.

Given the migration delay $migdelay_i$ for each virtual machine $i \in M$, it is possible to measure the reconfiguration delay for each $i \in V$ which depends if the virtual machine is migrating or not.

If virtual machine $i \in V$ is migrating ($i \in M$), then the reconfiguration delay, defined as $recdelay_i$, is measured as the maximum of $\{migdelay_i\} \cup \{migdelay_j \mid j \in M \, and \, src_j = dst_i\}$. The reconfiguration is only complete when the virtual machine has been migrated and the destination physical server dst_i already finished migrating its virtual machines, if dst_i had virtual machines to be migrated. The migration delays of the virtual machines leaving the destination physical server are considered since the virtual machine i may require the capacity that will be released by these virtual machines before completing the reconfiguration process.

If virtual machine $i \in V$ is not migrating ($i \notin M$), then the reconfiguration delay $recdelay_i$ is measured as the maximum of $\{migdelay_j \mid j \in M \, and \, src_i = src_j\}$. The reconfiguration is only complete when all migrations from the local physical server have already finished.

[1] $u_i'^{mem}$ shouldn't be confused with u_i^{mem} which represents the current demand of memory, i.e., how much of memory will be allocated to virtual machine after the consolidation event

In the cases that there is no migration, the reconfiguration delay is admitted to be zero.

The goal of the dynamic server consolidation with controlled reconfiguration delays problem is to generate a mapping of virtual machines to physical servers x_{ij} that minimizes the number of physical servers used and also, given a maximum reconfiguration delay threshold σ, decreases the number of occurrences where $recdelay_i > \sigma$ for all $i \in V$. The objective function can be defined as

$$minimize \sum_{j \in P} y_j + \alpha * \sum_{i \in V} rdbreak_i$$

where y_j is a binary array equal to 1 if physical server $j \in P$ is used, and 0 otherwise; $rdbreak_i$ is a binary array equal to 1 if reconfiguration delay $recdelay_i$ of virtual machine i is greater than the threshold σ, and 0 otherwise; and α is a penalty included for each event where the maximum reconfiguration delay threshold is broken. The value of α should be greater than $|P|$ in order to ensure that the solution focus on reducing the number of breaks in the reconfiguration delay.

3.3 Proposed algorithm

The dynamic server consolidation with controlled reconfiguration delays problem includes additional constraints to the dynamic server consolidation problem. The goal is to minimize the number of physical servers required to map a list of virtual machines, but it also should guarantee that the cost of changing from the previous mapping to a new one is below a given threshold. This cost is the reconfiguration delay, which depends highly on how the migration of virtual machines is performed in the system. On the other hand, the migration depends on the cost to migrate each virtual machine, which depends on its size (more specifically memory allocation), and the network infrastructure that interconnects all physical servers.

Due to the considerable increase of complexity in the problem, in relation to the dynamic server consolidation problem, the utilization of optimization techniques that result in an optimal solution, such as linear programming, becomes prohibitive. The number of additional constraints and variables increases considerably. Therefore, the utilization of other methods is required.

The proposed algorithm is divided in two distinct phases. The first phase aims at finding a feasible mapping of virtual machines to physical machines that minimizes the maximum reconfiguration delay of all virtual machines. In the second phase, the feasible mapping is iteratively modified in order to produce solutions using a smaller number of physical servers, but also respecting the maximum reconfiguration delay threshold. In cases where it is not possible to guarantee delays smaller than the maximum reconfiguration delay threshold, the algorithm finds a solution that minimizes the number of virtual machines that have their reconfiguration delays higher than the specified threshold.

3.3.1 First phase

The first phase is based on the traditional descent method for local neighborhood search. Based on an initial solution, a set of small modifications to this solution is derived. The result obtained with each modification is analyzed and the best one is chosen. If this modification optimizes the current solution, then it is applied and the process repeats, otherwise the algorithm stops.

The algorithm starts repeating the last mapping as the current solution. This operation results in zero migrations, however the physical servers can become overloaded, i.e., the physical server capacity can not be able to handle the changes of the virtual machines demands mapped to it. The strategy is to remove each overloaded physical server from this overload state, migrating some of its virtual machines to other physical servers, choosing every time the alternative that results in minimal reconfiguration delays. In each step, migration alternatives are generated for each overloaded physical server. The physical server that performs the migrations which results in minimal reconfiguration delay is chosen. The migrations are included in the current solution, and the process repeats, until there are no more overloaded physical servers.

The generation of the migration alternatives is executed as follows. For each overloaded physical server, combinations of its virtual machines that remove the physical server from the overload state are generated. The generation of the alternatives ensures that each alternative contains only the required migrations that remove the physical server from the overload state. For example, given a physical server with capacity 100, packing virtual machines: v_1 with demand 50, v_2 with demand 40, v_3 with demand 30, and v_4 with demand 20, the combinations of virtual machines generated are: (v_1), (v_2), and $(v_3,\ v_4)$.

Each combination of virtual machines is mapped to the current solution using the best-fit decreasing heuristic and its cost is evaluated. The best-fit decreasing heuristic used has a slight modification in order to reduce the number of migrations using the same network links, and consequently the reconfiguration delay. The eligible physical servers to receive a virtual machine are only the ones that didn't receive any virtual machines yet, i.e., each physical server receives only one virtual machine migration. The cost function is defined as:

$$\begin{gathered} 1000 * \text{number of overloaded physical servers} + \\ 100 * \text{maximum reconfiguration delay} + \\ \text{sum of reconfiguration delays} \end{gathered}$$

The first term gives a big penalty (1000) when there are overloaded physical servers. The heuristic is directed to reduce the number of overloaded physical servers to zero, canceling this term. The second term is based on the maximum reconfiguration delay between all virtual machines. The goal is to select the alternative that minimizes the reconfiguration delay. When there is more than one alternative that gives the same maximum reconfiguration delay, the last term is used to choose the one that has minimum aggregate reconfiguration delay.

The process repeats until there are no more overloaded physical servers. In the end, the result generated will contain the same, or probably, more physical servers than the previous mapping. Algorithm 2 presents the algorithm for the first phase. The second phase is eventually used to decrease the number of physical servers, while guaranteeing that the reconfiguration delay stays under a given threshold.

Algorithm 2 First phase of the dynamic server consolidation with controlled reconfiguration delays algorithm

```
$current\_solution \leftarrow$ get-current-mapping()
while there are overloaded physical servers in $current\_solution$ do
    for all $p$ in overloaded physical servers do
        $alternatives \leftarrow$ generate-migration-alternatives($p$)
        for all $alt$ in $alternatives$ do
            $sol \leftarrow$ best-fit-decreasing( $alt$ ) {each virtual machine receives at most one virtual machine migration}
            $cost \leftarrow$ cost-function( $sol$ )
        end for
    end for
    apply alternative with lowest cost to $current\_solution$ {the physical server is automatically removed from the overload state}
end while
```

3.3.2 Second phase

In the second phase, it is used an implementation of the server consolidation problem using a method called tabu search. Tabu search is a common metaheuristic used to solve optimization problems. The main advantage of using metaheuristics is that it overcomes the problem of being trapped in a local optima, common to local search approaches as the ones used by heuristics. Besides, it presents lower computational costs than linear programming methods. Although it usually results in better solutions than heuristics, it can not guarantee the optimality of the result. Other examples of metaheuristics include: genetic algorithms and simulated annealing [FLO06].

The main idea of tabu search is to maintain a memory about previous local searches, in order to avoid performing repeatedly the same moves, returning to the same solution and staying confined into a local optima. The tabu search method is based on a repetition of steps that explores the possible solutions for the problem. At first, a feasible initial solution is generated for the problem and set as the current solution. This initial solution is usually far from optimal, but is simple to be generated. The current solution is initially set as the best solution found so far.

With the current solution set, it starts evaluating possible changes in the current solution and evaluating each alternative. In order to do that, it generates a set of moves given the current solution using simple transformations in its local neighborhood. Some of the moves are forbidden since they are in the tabu list. The tabu list represents the memory of the algorithm and contains a list of moves used in the previous iterations. All feasible moves are evaluated and their costs obtained using a cost function. The cost function is used to differentiate between good and bad solutions. The move which results in the solution with best cost is chosen to be applied, and the move is included in the tabu list. The tabu list is usually implemented as a fixed length list, dropping the oldest element when its size is exceeded. If the current solution has better cost than the best solution found so far, the current solution is set as the best solution. This process repeats until a termination condition is met. The termination condition can be a timeout, a specific number of repetitions, or a number of repetitions without any improvement in the best solution.

Given the tabu search general framework, the implementation of the second phase of the algorithm starts by using as initial solution, the resulting mapping from the first phase. The strategy is to select in each iteration one physical server to empty, reassigning its virtual machines to other physical servers. The selection of the physical server is based on a filling function, proposed by [LOD04], which gives a measure of easiness to empty the physical server. The filling function for physical server j is defined as:

$$\varphi(S_j) = \alpha \frac{\sum_{i \in S_j} u_i}{c_j} - \frac{|S_j|}{|V|}$$

where S_j is a set containing the virtual machines assigned to physical server j; u_i is the demand of virtual machine $i \in V$; c_j is the capacity of physical server j; α is a pre-specified positive value. The function gives higher priority to physical servers with low occupied capacity and more virtual machines packed on it. The physical server with the lowest filling index, according to the filling function, and that is not in the tabu list is chosen. The tabu list stores a list of previous physical servers chosen to get empty, and the goal of the tabu list is to avoid choosing repeatedly the same physical servers to empty.

After choosing the physical server using the filling function, the virtual machines mapped to it are retrieved and the physical server is set as unavailable during this iteration, in order to avoid remapping all virtual machines to it again. The list of virtual machines is used as input to a permutation function, which returns lists with these virtual machines in all possible orderings. Since the number of lists can be extremely large (it generates $n!$ lists, where n is the number of virtual machines), a smaller number of alternatives is randomly chosen. Each alternative is evaluated, applying the worst-fit heuristic to map the virtual machines to the physical servers. The alternative that provides a solution with best cost, according to the cost function, is selected and its solution is defined as the current solution. The cost function that should be minimized is defined as:

$$\begin{gathered}\text{number of physical servers } + \\ 1000 * \text{number of reconfiguration delay breaks}\end{gathered}$$

It includes the number of physical servers and a penalty for each virtual machine that has a reconfiguration delay greater than the maximum reconfiguration delay threshold, which indicates a reconfiguration delay break. The physical server chosen at first is included in the tabu list and set as available again to pack virtual machines in the next iterations. The tabu list size has a fixed capacity, and when this capacity is exceeded, the oldest entry is removed. The tabu list size should be smaller than the number of physical servers. If the cost of the current solution is smaller than the cost of the best solution, then the current solution is defined as the best solution. This process repeats until the best solution does not present any enhancements in a pre-specified number of iterations. The final solution can result in a situation that guaranteeing the maximum reconfiguration delay threshold is not possible, however it will minimize the number of occurrences of this situation. The algorithm is presented in Algorithm 3.

Algorithm 3 Tabu search algorithm for the dynamic server consolidation problem.

```
$current\_solution \leftarrow$ mapping from first phase {initial solution}
$best\_solution \leftarrow current\_solution$
repeat
  $p \leftarrow$ physical server with lowest index according to filling function and not in $tabu\_list$
  $vms \leftarrow$ get virtual machines mapped to physical server $p$ and set $p$ as unavailable
  $moves \leftarrow$ get $vms$ permutations
  for all $m$ in $moves$ do
    $sol \leftarrow$ worst-fit($m$)
    $cost \leftarrow$ cost-function($sol$)
  end for
  $current\_solution \leftarrow$ solution with smallest cost
  insert $p$ into $tabu\_list$ and set it as available again
  if cost-function($current\_solution$) $<$ cost-function($best\_solution$) then
    $best\_solution \leftarrow current\_solution$
  end if
until termination condition
```

4. Evaluation

This chapter presents an evaluation of the dynamic server consolidation algorithms and the proposed extension considering guarantees over the maximum reconfiguration delay. Initially, the workloads used for the experiments are described. The experiments performed are divided in three groups. Each group focus on evaluating the algorithms under different workload characteristics, which are based on workload variability, workload patterns, and workload utilization. All algorithms are analysed considering the average number of physical servers required, the number of migrations performed, the maximum reconfiguration delay, and the number of reconfiguration delay breaks, i.e., number of instances in which the reconfiguration delay is higher than a given threshold. Results indicate that the dynamic server consolidation algorithm proposed for controlled reconfiguration delays attends, in most cases, the maximum reconfiguration delay threshold required. When the threshold is not reached, the algorithm minimizes the number of reconfiguration delay breaks.

4.1 Workloads

In order to investigate the functioning of the dynamic server consolidation algorithms and evaluate the proposed algorithm for the dynamic server consolidation with controlled reconfiguration delays, it is required a workload composed by server traces with periodic samples of resources utilization (e.g., CPU, memory, etc). Each sample indicates the demand required in the next consolidation event. Several attempts were made in order to obtain workloads from real production data centers, however all attempts failed. This information is usually defined internally as classified by enterprises, since it provides hints about the data center efficiency. Therefore, it was generated a workload from available servers at TU-Berlin. The servers were monitored during a period of approximately six months. These servers are usually used by students and researchers to execute experiments and simple applications. Nonetheless, the generated workload enabled the identification of common utilization patterns, which are also present in production data centers. This analysis guided the implementation of a synthetic workload generator. This generator creates workloads that reproduce common utilization patterns found in data centers.

4.1.1 Real workload from TU-Berlin servers

The Technical University of Berlin provides to its students and researchers several servers that can be used to execute long-running experiments and perform tests that require more computing resources. In order to generate the workload, nine of these servers were monitored during a period of approximately 6 months. In each server, CPU and memory percentages of utilization were sampled each minute and this information was stored in a trace file. The group of servers is composed by different families of UltraSPARC servers running SunOS 5.10 and varying configurations, ranging from 1 to 20 CPUs and 16 to 48 GBytes of RAM.

Figure 4.1 presents CPU (black line) and memory (gray line) utilization for each server. The samples in which CPU and memory utilization are equal to 0 represent moments of server unavailability. The main difference between this workload and the ones found in production data centers is that these servers usually execute at the same time several users' applications. In a regular data center, each server would be usually assigned to execute a single application, and it would present more visible patterns. Nevertheless, it is still possible to identify some common application utilization patterns.

The utilization patterns identified were divided in three groups. The goal of establishing such classification is to serve as a model to implement a synthetic workload generator. The patterns are:

Steady The steady utilization pattern represents workloads that present low variation in its utilization in a medium to long term period. Server bonito in Figure 4.1 present several periods of steady utilization, with very little variation in resources utilization.

Periodic The periodic utilization pattern represents workloads that repeatedly present the same variations, and is relatively common in production data centers [BOB07]. This utilization pattern can be observed in the trace of servers bolero, bueno and fiesta in Figure 4.1.

Random The random utilization pattern represents workloads that do not fit in the other groups, and do not present any accurate pattern that can be used to predict its future behavior. The traces from servers pepino and pepita in Figure 4.1 represent a typical random pattern.

4.1.2 Synthetic workload generator

In order to be able to evaluate different workload types and due to the difficulty in obtaining these workloads, a synthetic workload generator was implemented. The generator was implemented using the R statistical computing environment and uses the utilization patterns defined, based on the analysis of the real workload.

The implementation details of each utilization pattern is described below. Figure 4.2 presents an example of traces generated using the synthetic workload generator with each pattern.

Steady The steady pattern represents an utilization with a clear mean value and low variation, and is implemented using a random number generator for the normal distribution (function *rnorm* from R). The implementation receives as parameters a mean and standard deviation, used as input to the number generator, and the limits in which the numbers should be encountered. Each number generated out of limits is substituted by the lower or upper limit accordingly. Figure 4.2(a) presents an example of a trace using the steady utilization pattern.

Periodic The periodic pattern represents a repeatable shape of variations in utilization. It is implemented using a simplified stationary time series model. Time series models can be decomposed in three parts, a trend part which identifies the time series progression in the long term, a cyclic part which identifies a shape that repeats periodically in the time series, and a noise

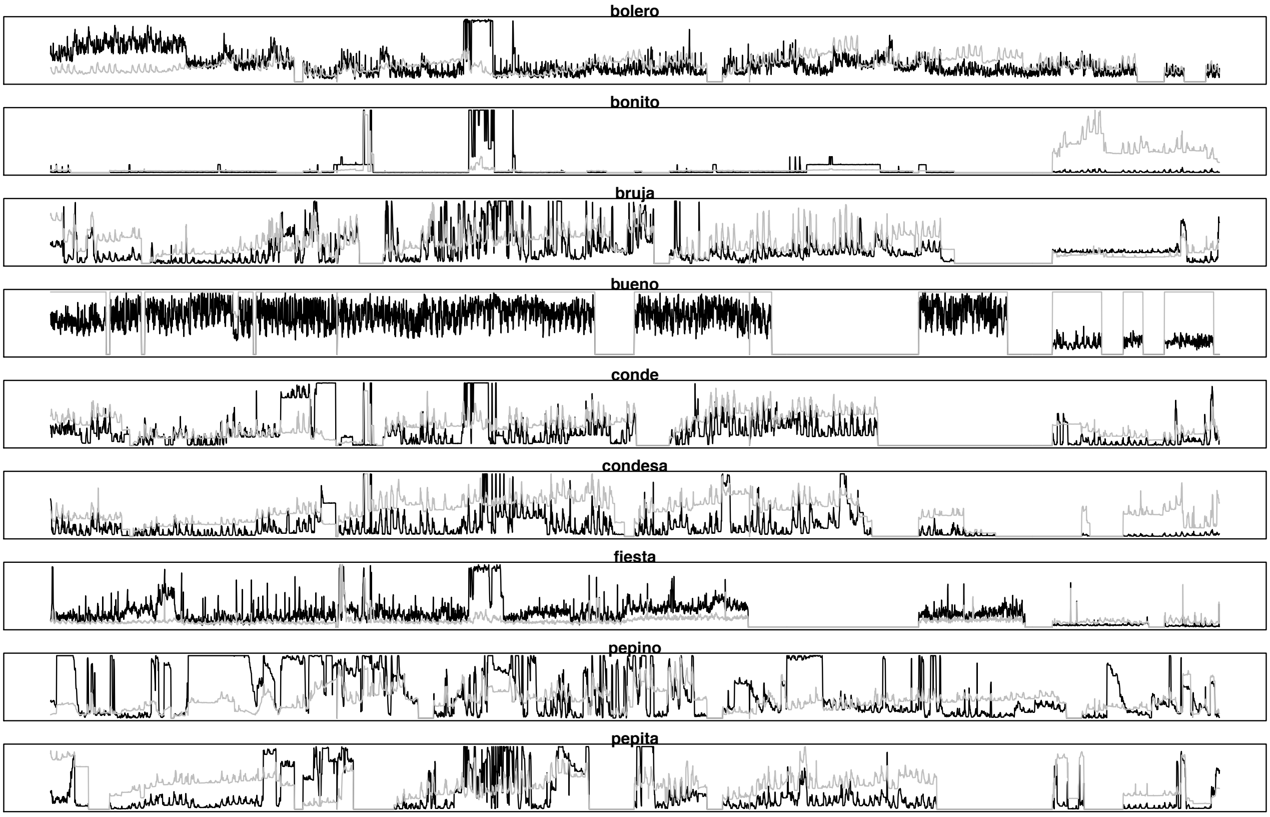
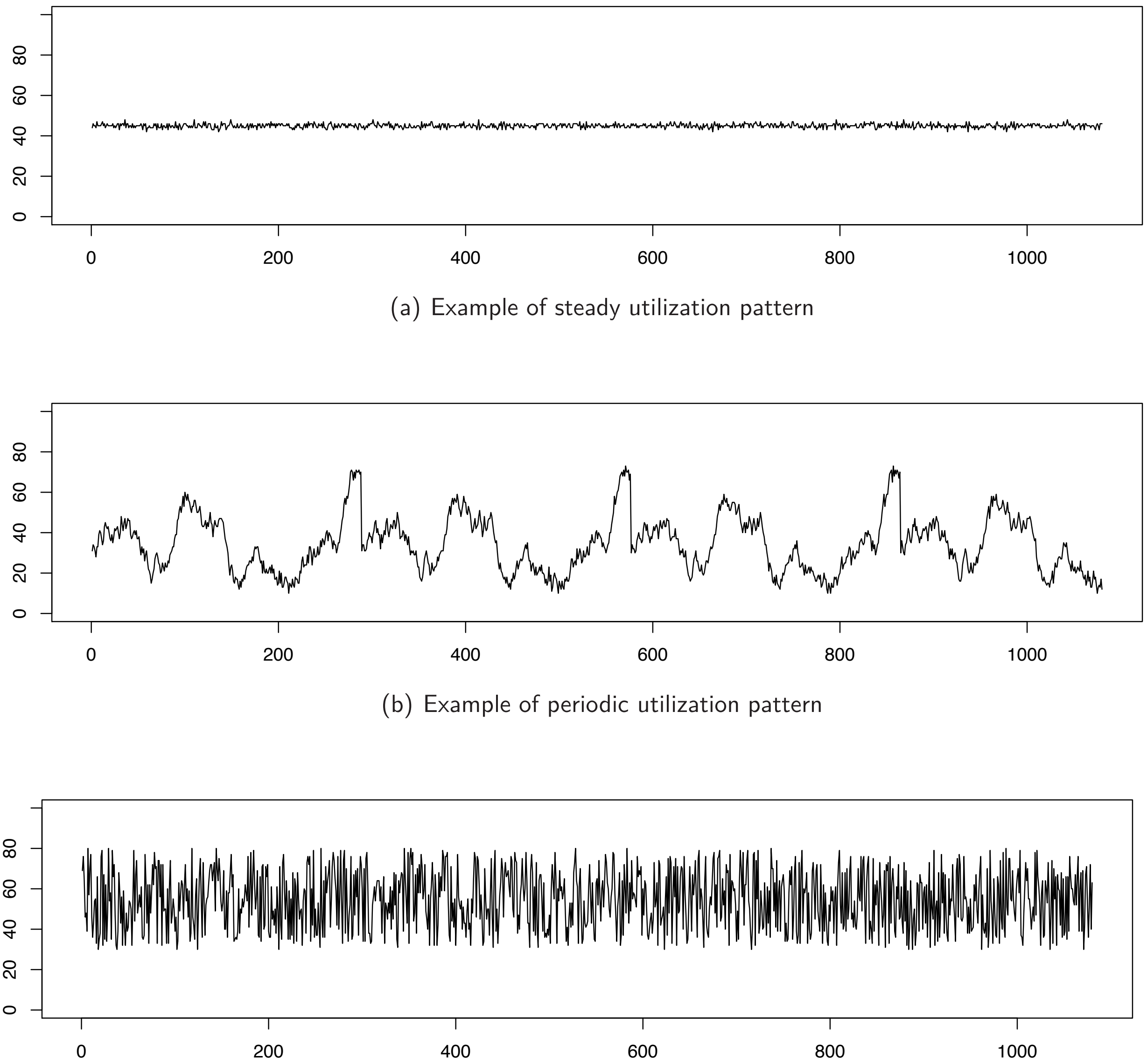


Figure 4.1 – CPU and memory utilization of TU-Berlin servers

part which identifies the irregular variations that can appear in the data. Considering that the time series is stationary, the trend part is not used. The shape used by the cyclic part is generated using a uniform distribution with minimum and maximum limits, and a δ parameter which identifies the maximum variation between two adjacent points in the shape. The idea of using the δ parameter is to force the existence of a continuity in the data. It is also defined a number of points to be generated for the shape, which identifies the periodicity of the cyclical part. After generating the time series using the cyclic part, a noise is included in each point based on a normal distribution (normal white noise), in which the mean is the point value itself, and the standard deviation is a given input. Figure 4.2(b) presents an example of a trace using the periodic utilization pattern.

Random The random pattern represents an utilization with unpredictable form. It is implemented using a random number generator for the uniform distribution (function *runif* from R). The implementation receives as parameters the minimal and maximal values, used as input to the number generator. Figure 4.2(c) presents an example of a trace using the random utilization pattern.

The synthetic workloads used for the experiments were composed by different groupings of the traces, obtained using the synthetic workload generator. The traces were generated using the parameters presented in Table 4.1. For each utilization pattern, 40 traces were generated. Each



(a) Example of steady utilization pattern

(b) Example of periodic utilization pattern

(c) Example of random utilization pattern

Figure 4.2 – Workloads generated using the synthetic workload generator

trace has 336 samples, corresponding to one consolidation event each 30 minutes during 1 week. For each sample generated, the same value is used for CPU and memory.

4.2 General description

The scenario used in all experiments is a virtualized data center infrastructure composed by 100 homogeneous physical servers with CPU and memory capacities equal to 100. Each server is connected to a single crossbar switch using bidirectional links forming a star network topology. Each link has a capacity of 100. It means that it takes 1 unit of time to transfer all memory from one physical server to another using full link capacity. The CPU, memory and network capacities can be considered as a percentage capacity, and the objective is to simplify the conversion of the relative

Utilization pattern	Number of traces	Parameters used
Steady	40 (2 traces per pair of mean and standard deviation)	limits=[10;90] (mean, standard deviation) = (15,1), (15,2), (15,3) (15,4), (15,5), (30,1), (30,2), (30,3), (30,4), (30,5), (45,1), (45,2), (45,3), (45,4), (45,5), (60,1), (60,2), (60,3), (60,4), (60,5)
Periodic	40 (10 traces for each number of points)	limits=[10;90] $\delta = 5$ number of points = (12, 96, 144, 288) standard deviation=1
Random	40 (2 traces per limits interval)	limits=[10;20], [10;30], [10;40], [10;50], [10;60], [20;30], [20;40], [20;50], [20;60], [20;70], [30;40], [30;50], [30;60], [30;70], [30;80], [40;50], [40;60], [40;70], [40;80], [40;90]

Table 4.1 – Parameters used to generate traces using the synthetic workload generator

measure of time to real scenarios. For example, given a scenario where each physical server contains 16 GBytes of memory and is connected to a 1 GbEthernet network, the relative measure of time of 1.0 would represent approximately 128 seconds, corresponding to the time to transfer 16 GBytes using 1Gbit/s. A relative time of 0.5 would represent half of the time, or 64 seconds, and so on. The experiments present the values according to the relative time. However, in order to facilitate the interpretation of the results, the converted time according to this hypothetical scenario will be presented when appropriate.

All solutions presented for the dynamic server consolidation problem were implemented for the evaluation. The implementation details of each algorithm are:

Heuristics The heuristics implemented were: next-fit (NF), first-fit (FF), best-fit (BF), worst-fit (WF), almost worst-fit (AWF), next-fit decreasing (NFD), first-fit decreasing (FFD), best-fit decreasing (BFD), worst-fit decreasing (WFD) and almost worst-fit decreasing (AWFD). They were implemented using the Python language.

Linear programming The linear programming (LP) solution was implemented using Zimpl [KOC04] and solved using the SCIP [ACH04] solver. The solver was configured with a timeout of 5 minutes, i.e., if the solver can not find the optimal result in 5 minutes, it returns the best result found so far. This approach is usually used since linear programming problems can take a long time to find an optimal solution.

Tabu search The tabu search (TS) algorithm corresponds to the second phase of the algorithm proposed to control reconfiguration delays. This version only considers in its cost function the number of physical servers used, excluding the reconfiguration delays. The initial solution is obtained using the first-fit algorithm. It was implemented using the Python language and configured with tabu list size equal to 5 and to terminate when the solution does not change during 10 iterations (termination condition).

The algorithm proposed for the dynamic server consolidation with controlled reconfiguration delays (DSCCRD) was implemented using the Python language. The second phase (using Tabu Search) is configured with tabu list size equal to 5 and to terminate when the solution does not present any changes in the last 10 iterations (termination condition). The first mapping is performed using the first-fit heuristic.

The goal of all experiments is to show how each algorithm behaves regarding the number of physical servers required to handle each workload, the required number of migrations, and the corresponding maximum reconfiguration delay. It is also desired to verify if the proposed algorithm (DSCCRD) attends the requirement of limiting the reconfiguration delay based on a defined threshold or minimizes the number of times when it can not attend the requirement (reconfiguration delay breaks), and the corresponding impact on the additional number of physical servers required. The values 1.0, 0.8, 0.6, 0.4 and 0.2 were used as input threshold for the DSCCRD algorithm. They represent the time of transferring 100%, 80%, 60,%, 40% and 20% of the physical server's memory to another server using full bandwidth. Considering the hypothetical scenario presented before, using physical servers with 16GBytes of memory and a 1 GbEthernet network, it would represent delays of 128, 102.4, 76.8, 51.2, and 25.6 seconds respectively. The number of reconfiguration delay breaks presented for the dynamic server consolidation algorithm implemented using heuristics, linear programming and tabu search considers 1.0 as the threshold for maximum reconfiguration delay.

The experiments were divided in three groups, each one focusing on analyzing the algorithms using workloads with different characteristics. The groups were divided based on workload variability, workload patterns and workload utilization. The real workload obtained from TU-Berlin servers and the workload generated using the synthetic workload generator were used in the experiments.

4.3 Analysis on workload variability

The goal of this analysis is to investigate the behavior of the algorithms when workloads with different variability are used. Workload variability is defined as the ratio of demand changes in each one of the traces that compose the workload. The workload obtained from TU-Berlin was used to perform this analysis.

One of the main problems of the workload is the low number of available traces (only 9). It is necessary to have more data in order to analyze the benefits and drawbacks of each consolidation algorithm. Therefore, each trace was divided in week periods, and considered each week period as belonging to a different trace. It resulted in a total of 140 traces, each one relative to a single virtual machine. The frequency of the measurements was also modified from one sample per minute to one per hour (using the peak demand in the corresponding period), resulting in 168 samples of CPU and memory demand per trace. Each sample is used in a different consolidation event. The CPU and memory values vary from 0 to 100, since the percentage was originally measured. Despite the

	Number of traces	Variability rate	Average CPU utilization	Average memory utilization
Group 1	43	0 - 30	25.2%	28.36%
Group 2	61	30 - 50	32.37%	39.39%
Group 3	36	50 - 100	47.28%	48.67%

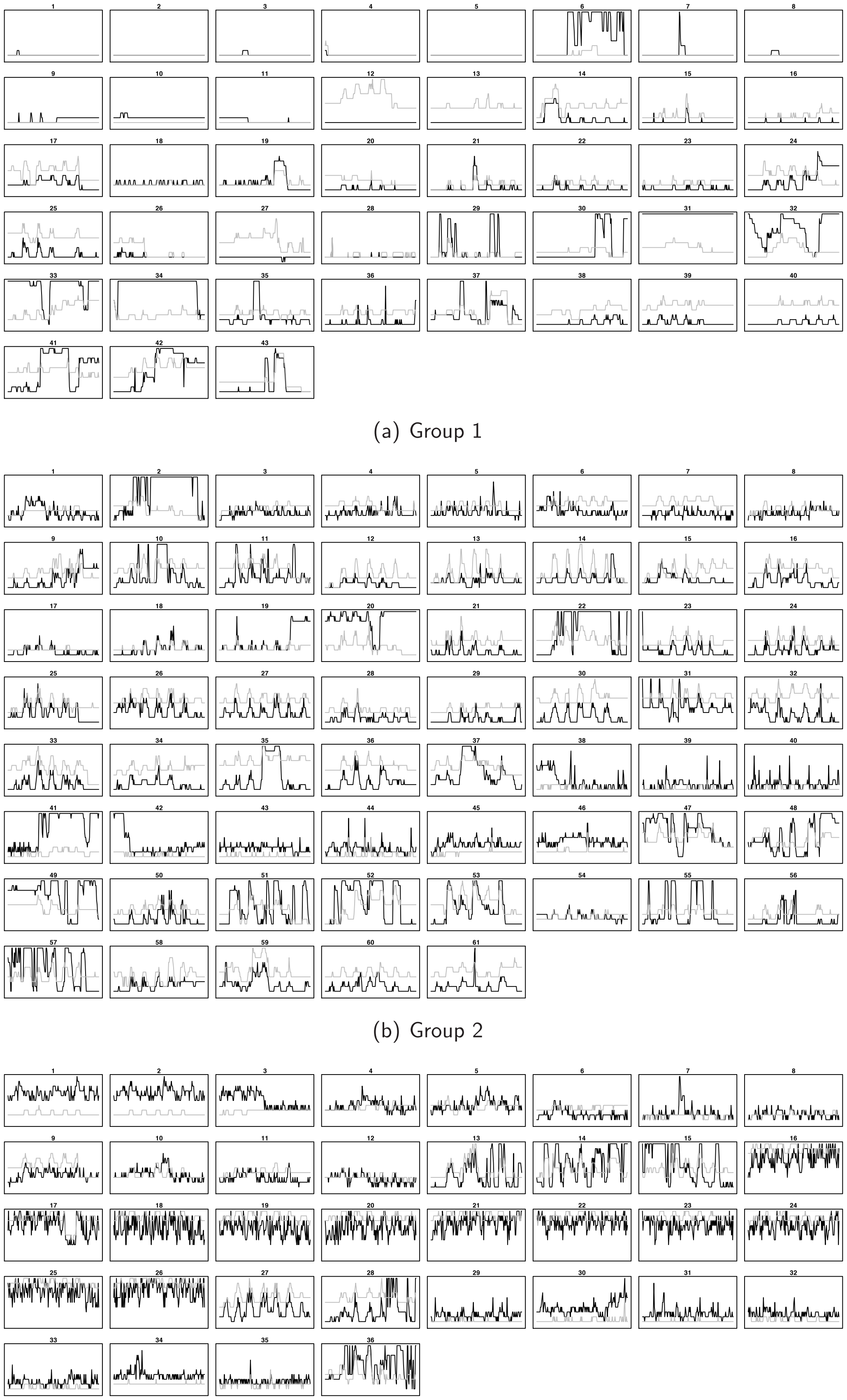
Table 4.2 – Characteristics of each group for workload variability analysis.

differences in CPU and memory capacity in each original server monitored, the data is considered to belong to a homogeneous environment.

In order to evaluate the problem with workloads with different variability, the 140 traces were divided in 3 groups, forming the workloads, according to the variability of each trace in terms of CPU and memory demand. Trace variability rate was measured as the number of demand changes by the total number of samples, in this case 168. Table 4.2 presents details about each group regarding the number of traces, variability range used to select the traces, and average CPU and memory utilization in percentage. Group 1 presents lower variability, between 0% (samples are the same during all time steps) and 30% consisting of a total of 43 traces. Group 2 is the larger group, containing 61 traces and variability between 30% and 50%. Group 3 is the smallest group with 36 traces, presenting variability between 50% and 100% (resource capacity requires changes in all time steps). Figure 4.3 shows the traces of each group. In each graph, the black line represents the CPU utilization and the gray line represents the memory utilization.

The results obtained executing each algorithm for each workload group are presented in Tables 4.3, 4.4 and 4.5. For each algorithm, it is presented the average number of physical servers used in all consolidation events, the total number of migrations performed, the maximum reconfiguration delay, and the number of reconfiguration delay breaks.

The workload used in group 1 is the one that has lower variability in its traces. The results presented in Table 4.3 show that LP presents the best consolidation, being able to pack all 43 traces in average to 15 physical servers. It results in a reduction of approximately 75%, when compared to having each trace in a single physical server. However, LP also presents the highest number of migrations, migrating approximately 38 virtual machines each consolidation event, which represents 88% of the virtual machines. It also presents the highest reconfiguration delay of 1.6 (204.8 seconds according to the hypothetical scenario), and highest number of reconfiguration delay breaks given threshold 1.0. Some of the heuristics - FFD, BFD, WFD, and AWFD - also presented good results regarding consolidation, with a still high number of migrations and high number of reconfiguration delay breaks. The results with TS shown in average good results for consolidation and reconfiguration delay, similar to the best heuristics. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8 and 0.6 with 0 breaks. Using thresholds 0.4 and 0.2, DSCCRD resulted in 6 and 46 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 0.5 and 1.0. This situation happens when a threshold can not be attended, since the algorithm tries to decrease the number of reconfiguration delay breaks, even if it requires increasing the maximum



(a) Group 1

(b) Group 2

(c) Group 3

Figure 4.3 – Workload groups for variability analysis

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	18.6	1739	1.3	65 (1.0)
FF	16	1940	1.3	120 (1.0)
BF	16.1	1888	1.3	84 (1.0)
WF	16.3	1909	1.3	98 (1.0)
AWF	16.1	1984	1.3	92 (1.0)
NFD	18	3916	1.5	400 (1.0)
FFD	15.5	3815	1.3	763 (1.0)
BFD	15.5	3847	1.3	778 (1.0)
WFD	15.5	4136	1.3	970 (1.0)
AWFD	15.5	3916	1.3	812 (1.0)
LP	15	6464	1.6	2904 (1.0)
Tabu search	15.8	2833	1.4	69 (1.0)
DSCCRD (1.0)	16	474	1	0 (1.0)
DSCCRD (0.8)	16.5	293	0.8	0 (0.8)
DSCCRD (0.6)	17.2	217	0.6	0 (0.6)
DSCCRD (0.4)	19	154	0.5	6 (0.4)
DSCCRD (0.2)	20.5	82	1	46 (0.2)

Table 4.3 – Workload variability analysis - results for group 1.

reconfiguration delay of some virtual machines. It is possible to observe that as the threshold becomes tighter, DSCCRD uses less migrations and more physical servers. In comparison to LP, DSCCRD uses between 6% to 15% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2, it uses 27% and 37% more physical servers in order to minimize the number of reconfiguration delay breaks.

In the results obtained for group 2, shown in Table 4.4, LP also presented the best consolidation, enabling a reduction in average of 55% of physical servers to pack all traces, in comparison to mapping one trace per physical server. It presented once more the highest number of migrations, requiring to migrate approximately 97% of the virtual machines in every consolidation event, and highest number of breaks. However, the highest reconfiguration delay was generated using heuristic WFD, with a value of 1.8 (230.4 seconds according to the hypothetical scenario). The BFD heuristic presented the best consolidation among heuristics. An interesting fact is that despite of the high number of migrations presented by NFD, which is similar to the other decreasing heuristics (FFD, BFD, WFD and AWFD), it shows considerably lower number of reconfiguration delay breaks than these other heuristics. TS presented again average results, showing good results regarding consolidation, migrations, maximum reconfiguration delay, and number of reconfiguration delay breaks, in comparison to the heuristics. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8 and 0.6 with 0 breaks. Using threshold 0.4 and 0.2, DSCCRD resulted in 34 and 135 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 1.0 in both cases. The same behavior using less migrations and more physical servers when the threshold gets tighter

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	36.6	5250	1.6	171 (1.0)
FF	28.6	5646	1.7	795 (1.0)
BF	28.6	5833	1.5	799 (1.0)
WF	29.6	5750	1.6	701 (1.0)
AWF	29	5897	1.5	738 (1.0)
NFD	34.4	8446	1.6	854 (1.0)
FFD	28	8346	1.6	3030 (1.0)
BFD	27.9	8361	1.6	3041 (1.0)
WFD	28.3	8442	1.8	2517 (1.0)
AWFD	28	8370	1.6	2819 (1.0)
LP	27.3	9908	1.6	4562 (1.0)
Tabu search	28.6	5375	1.4	135 (1.0)
DSCCRD (1.0)	30.2	856	1	0 (1.0)
DSCCRD (0.8)	31.2	652	0.8	0 (0.8)
DSCCRD (0.6)	33.2	475	0.6	0 (0.6)
DSCCRD (0.4)	37.9	245	1	34 (0.4)
DSCCRD (0.2)	41.1	137	1	135 (0.2)

Table 4.4 – Workload variability analysis - results for group 2.

was repeated. In comparison to LP, DSCCRD uses between 10% to 22% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2 it uses 39% and 51% more physical servers in order to minimize the number of reconfiguration delay breaks.

Group 3 presents the higher variability. The results presented in Table 4.5 show that LP resulted in the best consolidation, with a reduction of approximately 45% in the number of required physical servers to handle the workload. It does that requiring a huge number of migrations. Approximately 94% of the virtual machines are migrated in each consolidation event. It also presented the highest number of reconfiguration delay breaks. In relation to the maximum reconfiguration delay, the values obtained by LP, heuristics and TS were very similar. Once again, DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8 and 0.6 with 0 breaks. With threshold 0.4 and 0.2, DSCCRD resulted in 4 and 42 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 0.5 and 1.0 respectively. In comparison to LP, DSCCRD used between 7% to 13% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2 it uses 19% and 29% more physical servers in order to minimize the number of reconfiguration delay breaks.

As expected, LP presented the best consolidation in all groups. However, the high number of migrations required is considerably high. Almost all virtual machines are migrated in all consolidation events. The best consolidation results obtained with heuristics were quite similar to LP, being obtained in considerably less time. The average execution time using each algorithm was: 10 seconds

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	24.1	2547	1.3	40 (1.0)
FF	20.8	3190	1.4	275 (1.0)
BF	20.7	3248	1.5	341 (1.0)
WF	21.4	3160	1.4	203 (1.0)
AWF	20.9	3164	1.5	236 (1.0)
NFD	23.6	5040	1.4	227 (1.0)
FFD	19.9	5167	1.4	1000 (1.0)
BFD	20	5189	1.4	1025 (1.0)
WFD	20.2	5174	1.4	869 (1.0)
AWFD	20	5226	1.4	1068 (1.0)
LP	19.4	5758	1.4	1479 (1.0)
Tabu search	20.3	2944	1.5	89 (1.0)
DSCCRD (1.0)	20.9	814	1	0 (1.0)
DSCCRD (0.8)	21.4	597	0.8	0 (0.8)
DSCCRD (0.6)	22	445	0.6	0 (0.6)
DSCCRD (0.4)	23.1	298	0.5	4 (0.4)
DSCCRD (0.2)	25.1	138	1	42 (0.2)

Table 4.5 – Workload variability analysis - results for group 3.

for each heuristic, 12 hours for LP, 5 minutes for TS and 15 minutes for DSCCRD. TS resulted in average good results, providing good consolidation with relatively low number of migrations and maximum reconfiguration delay. In all groups, DSCCRD was able to avoid reconfiguration delays higher than 0.6 (76.8 seconds according to the hypothetical scenario). In all cases, decreasing the threshold used for maximum reconfiguration delay, resulted in a decrease in the number of migrations and increase in the average number of physical servers required. In group 2, it was possible to observe that having a huge number of migrations (LP) does not necessarily corresponds to a high maximum reconfiguration delay (heuristic WFD). Figure 4.4 presents the additional percentage of physical servers required by DSCCRD with different thresholds for groups 1, 2 and 3 in comparison to LP. Considering the variation in workload variability, it is possible to observe that, the number of migrations tend to change accordingly. As the variability gets higher, the algorithms tend to migrate more virtual machines and the difference of results obtained using LP and heuristics decreases. Nevertheless, the DSCCRD algorithm was still able to attend maximum reconfiguration delays down to 0.6 in all variability scenarios.

4.4 Analysis on workload patterns

The goal of this analysis is to evaluate the algorithms when using workloads with different utilization patterns. The traces generated using the synthetic workload generator were used in this

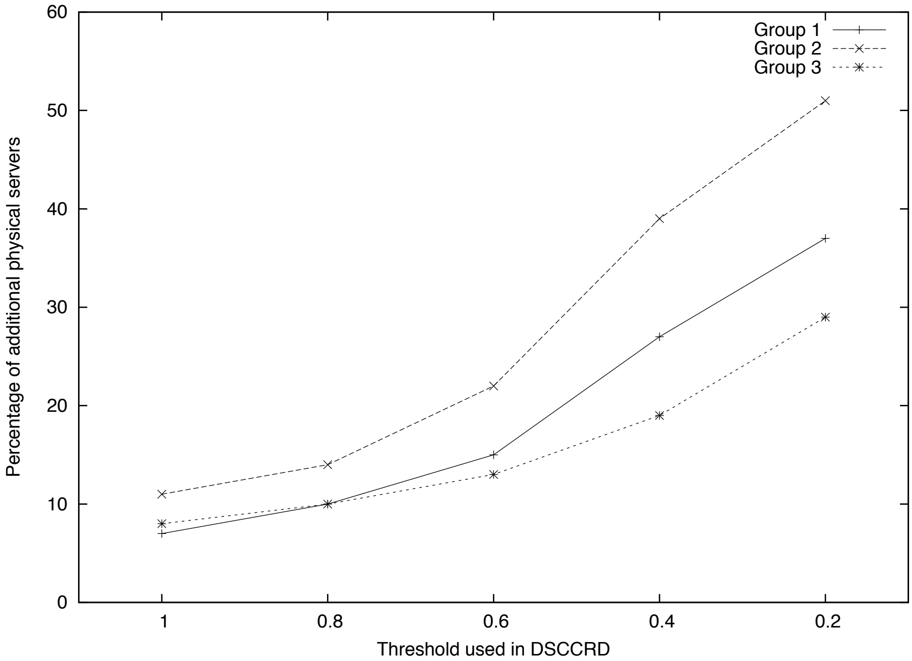


Figure 4.4 – Workload variability analysis - Percentage of additional physical servers required by DSCCRD

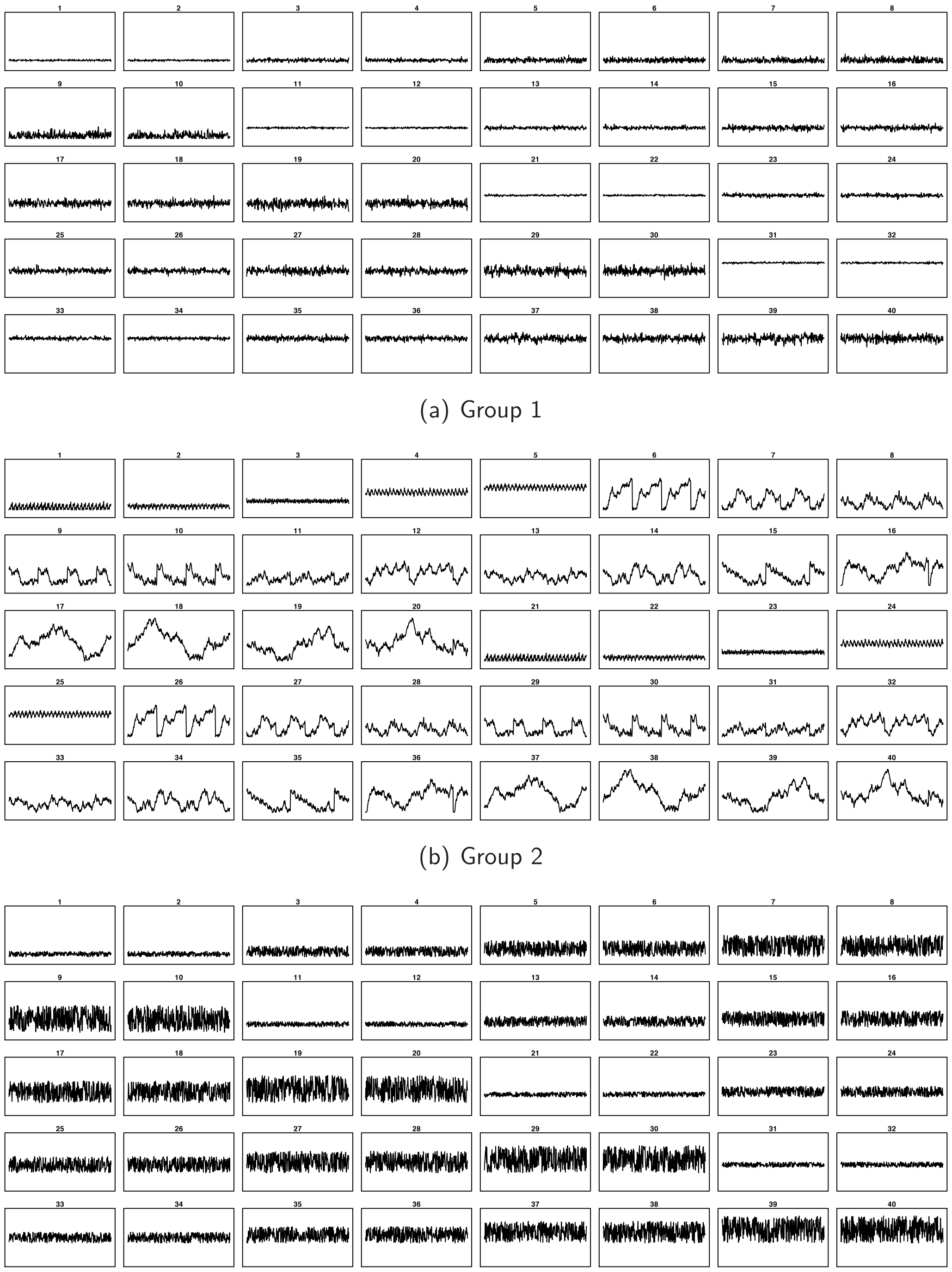
	Number of traces	Utilization pattern	Average CPU and memory utilization
Group 1	40	Steady	37.5%
Group 2	40	Periodic	32.7%
Group 3	40	Random	40.0%

Table 4.6 – Characteristics of each group for workload patterns analysis.

analysis. The traces were divided in three groups, based on its utilization pattern: steady, periodic, and random. The details of each group are presented in Table 4.6.

The results obtained executing each algorithm for each workload group are presented in Tables 4.7, 4.8 and 4.9. For each algorithm, it is presented the average number of physical servers used in all consolidation events, the total number of migrations performed, the maximum reconfiguration delay, and the number of reconfiguration delay breaks.

Group 1 represents a workload with steady utilization pattern. According to the results presented in Table 4.7, LP and some heuristics - FFD, BFD and AWFD - presented similar results for consolidation, being able to reduce in 57% the number of required physical servers in average, in relation to mapping each trace to a single physical server. They also presented similar results for maximum reconfiguration delay, of 1.4 (179.2 seconds according to the hypothetical scenario). LP required the highest number of migrations, migrating approximately 95% of the virtual machines each consolidation event, and presented the highest number of reconfiguration delay breaks. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8 and 0.6 with 0 breaks. It even got better consolidation results in comparison to other heuristics - FF, BF, WF and AWF - guaranteeing a



(a) Group 1

(b) Group 2

(c) Group 3

Figure 4.5 – Workload groups for workload utilization analysis

threshold of 0.6 (76.8 seconds according to the hypothetical scenario), instead of 1.3 (166.4 seconds according to the hypothetical scenario) provided by the heuristics, with considerably less migrations required. With threshold 0.4 and 0.2, DSCCRD resulted in 20 and 33 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 1.0 in both cases. In comparison to LP, DSCCRD uses between 9% to 21% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2, it uses 33% and 34% more physical servers in order to minimize the number of reconfiguration delay breaks.

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	21.6	4665	1.3	108 (1.0)
FF	21.2	4016	1.3	76 (1.0)
BF	21.2	4007	1.3	74 (1.0)
WF	21.3	4248	1.3	97 (1.0)
AWF	21.2	3987	1.3	74 (1.0)
NFD	21.6	9476	1.4	1168 (1.0)
FFD	17.3	9746	1.4	2974 (1.0)
BFD	17.3	9748	1.4	2958 (1.0)
WFD	17.4	9875	1.4	2582 (1.0)
AWFD	17.3	9847	1.4	2808 (1.0)
LP	17.2	12743	1.4	5335 (1.0)
Tabu search	18.6	7030	1.4	182 (1.0)
DSCCRD (1.0)	18.8	1030	1	0 (1.0)
DSCCRD (0.8)	19.7	430	0.8	0 (1.0)
DSCCRD (0.6)	20.8	90	0.6	0 (1.0)
DSCCRD (0.4)	22.9	36	1	20 (1.0)
DSCCRD (0.2)	23	21	1	33 (1.0)

Table 4.7 – Workload pattern analysis - results for group 1.

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	18.7	4995	1.7	490 (1.0)
FF	16.1	5325	1.7	385 (1.0)
BF	16.1	5314	2.1	349 (1.0)
WF	16.7	4934	2	365 (1.0)
AWF	16.2	5159	2.2	312 (1.0)
NFD	18.3	8943	2.9	1139 (1.0)
FFD	15.5	8590	2.8	2173 (1.0)
BFD	15.5	8589	2.8	2161 (1.0)
WFD	15.5	8810	2.8	2376 (1.0)
AWFD	15.5	8687	2.8	2154 (1.0)
LP	15.3	12436	2.2	5394 (1.0)
Tabu search	16.8	7487	2.3	304 (1.0)
DSCCRD (1.0)	16.2	1253	1	0 (1.0)
DSCCRD (0.8)	16.7	713	0.8	0 (0.8)
DSCCRD (0.6)	17.8	324	0.6	0 (0.6)
DSCCRD (0.4)	19.1	146	1	28 (0.4)
DSCCRD (0.2)	20.2	81	1	71 (0.2)

Table 4.8 – Workload pattern analysis - results for group 2.

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	23	7288	1.6	875 (1.0)
FF	21.4	7125	1.6	1271 (1.0)
BF	21.4	7128	1.6	1271 (1.0)
WF	21.6	7176	1.7	1211 (1.0)
AWF	21.4	7124	1.6	1288 (1.0)
NFD	22.4	11988	2.5	3736 (1.0)
FFD	18.4	12144	2.5	7202 (1.0)
BFD	18.4	12143	2.5	7202 (1.0)
WFD	18.5	12282	2.5	6942 (1.0)
AWFD	18.5	12249	2.5	7251 (1.0)
LP	18.4	12890	1.9	6297 (1.0)
Tabu search	19.9	8168	2	346 (1.0)
DSCCRD (1.0)	20.1	2128	1	0 (1.0)
DSCCRD (0.8)	21.4	1229	0.8	0 (0.8)
DSCCRD (0.6)	22.5	665	0.6	0 (0.6)
DSCCRD (0.4)	24	377	1	66 (0.4)
DSCCRD (0.2)	26	146	1	223 (0.2)

Table 4.9 – Workload pattern analysis - results for group 3.

Group 2 represents a workload with periodic behavior and the results obtained are presented in Table 4.8. LP got slightly better consolidation results than some heuristics - FFD, BFD, WFD, being able to reduce in 62% the number of required physical servers in average, and presented higher number of required migrations and reconfiguration delay breaks. In each consolidation event, LP requires migrating approximately 93% of the virtual machines. Despite of that, it presents considerably lower maximum reconfiguration delay in comparison to the heuristics that presented similar consolidation results (FFD, BFD, WFD, AWFD). TS did not present good results considering consolidation, but provides relatively low maximum reconfiguration delay and a reduced number of reconfiguration delay breaks. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8 and 0.6 with 0 breaks. With threshold 0.4 and 0.2, DSCCRD resulted in 28 and 71 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 1.0 in both cases. In comparison to LP, DSCCRD used between 6% to 16% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2 it uses 24% and 32% more physical servers in order to minimize the number of reconfiguration delay breaks. Despite of using TS in its second phase, DSCCRD was able to provide better consolidation results with thresholds 1.0 and 0.8. Therefore, TS results does not indicate a lower bound on DSCCRD consolidation efficiency.

Group 3 represents workloads with random utilization pattern. The results are presented in Table 4.9. The best consolidation was obtained both using LP and most decreasing heuristics - FFD, BFD, WFD and AWFD, resulting in a reduction of 54% in the required number of physical servers. They also presented similar results for the number of migrations and reconfiguration breaks. LP requires migrating approximately 95% of the virtual machines each consolidation event. Despite of that, LP presented considerably lower maximum reconfiguration delay in comparison to the other heuristics with similar consolidation results, 1.9 against 2.5 (243.2 and 320 seconds according to the hypothetical scenario respectively). TS does not present good results considering consolidation, but provides relatively low maximum reconfiguration delay and reduced number of reconfiguration delay breaks. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8 and 0.6 with 0 breaks. With threshold 0.4 and 0.2, DSCCRD resulted in 66 and 223 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 1.0 in both cases. In comparison to LP, DSCCRD used between 9% to 22% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2, it uses 30% and 41% more physical servers in order to minimize the number of reconfiguration delay breaks.

In general, the consolidation results obtained by LP and some heuristics (decreasing versions) were quite similar in all groups, but LP still requires the highest amount of migrations, migrating approximately all virtual machines every consolidation event. The approximate execution time for each implementation was: 20 seconds for each heuristic, between 12 and 24 hours for LP, 10 minutes for TS and between 15 to 30 minutes for DSCCRD. The results with TS considering consolidation were not satisfactory, but it usually resulted in a lower number of reconfiguration breaks and maximum reconfiguration time. In group 2 was possible to observe that TS does not necessarily represents a lower bound for DSCCRD, even using it in its second phase. In all groups, DSCCRD attended thresholds 1.0, 0.8 and 0.6, and when the threshold couldn't be attended, the number of breaks was minimized as the threshold gets smaller. The DSCCRD behavior of using more physical servers and decreasing the number of required migrations when the threshold gets lower was maintained in all groups. In order to counterpoint the idea that a huge number of migrations reflects in a high maximum reconfiguration delay, in groups 2 and 3, the LP maximum reconfiguration delay was considerably lower than other heuristics, even having higher number of migrations. DSCCRD presented lower number of migrations required and number of reconfiguration delay breaks when using a steady workload, and higher values with a random workload. Figure 4.6 presents the additional percentage of physical servers required by DSCCRD with different thresholds for groups 1, 2 and 3 in comparison to LP.

4.5 Analysis on workload utilization

The goal of this analysis is to evaluate the algorithms using workloads with variable average utilization ratios. The traces created using the synthetic workload generator were used for this analysis. They were divided in three workload groups according to its average utilization ratio. The details about each group regarding average CPU and memory utilization are presented in Table 4.10.

The results obtained executing each algorithm for each workload group are presented in Tables 4.11, 4.12 and 4.13. For each algorithm, it is presented the average number of physical servers

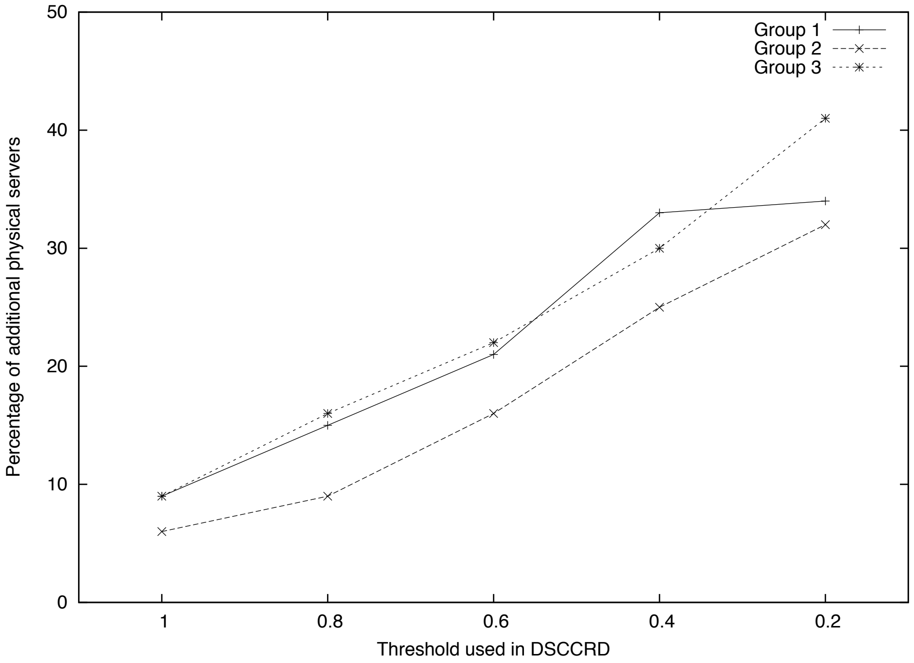


Figure 4.6 – Workload pattern analysis - Percentage of additional physical servers required by DSC-CRD

	Number of traces	Average CPU and memory utilization
Group 1	40	21.4%
Group 2	40	36.3%
Group 3	40	52.6%

Table 4.10 – Characteristics of each group for workload utilization analysis.

used in all consolidation events, the total number of migrations performed, the maximum reconfiguration delay, and the number of reconfiguration delay breaks.

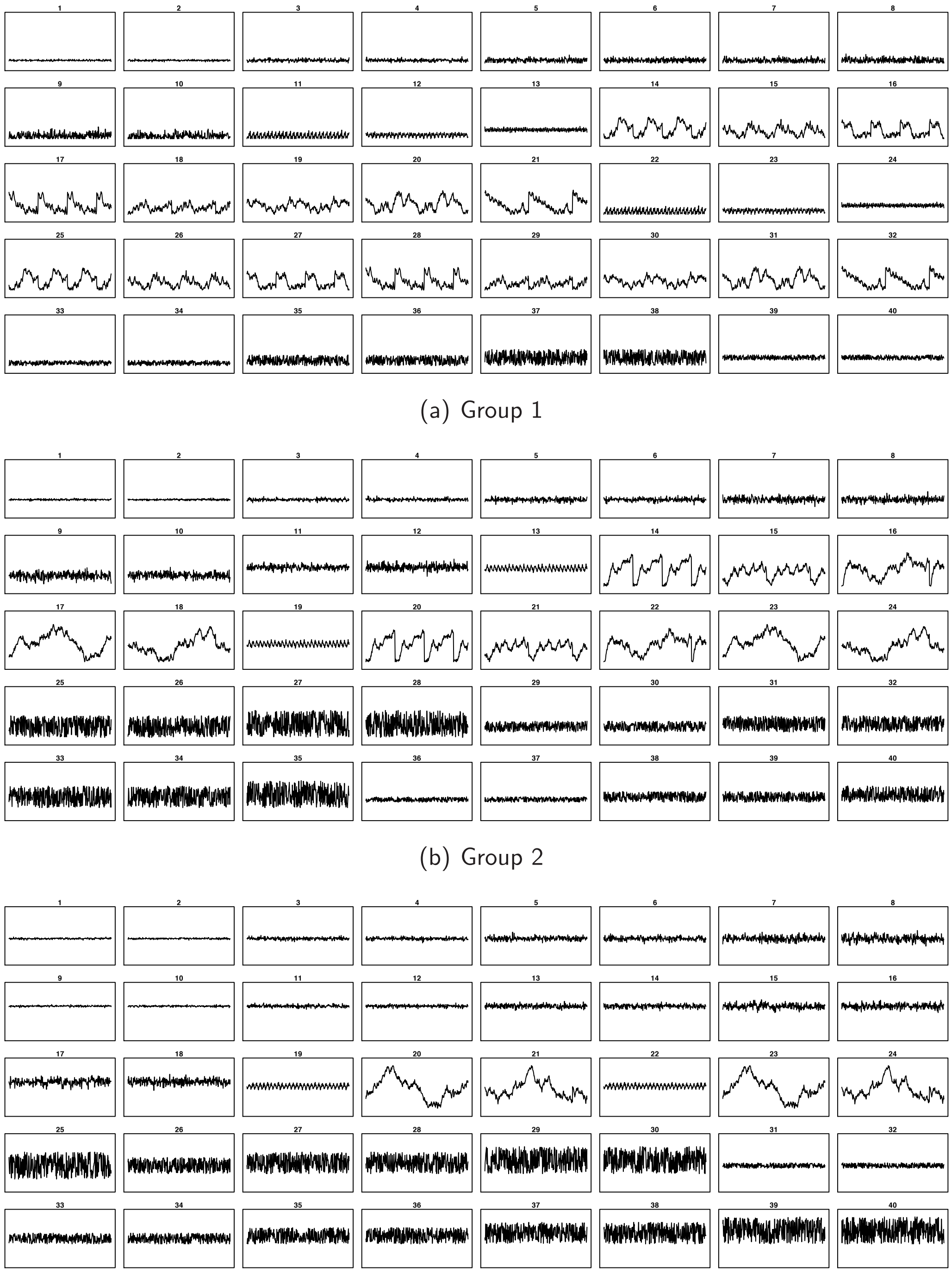
Group 1 represents a workload with low average utilization. Considering the results in Table 4.11, LP presented the best consolidation ratio, followed close by some heuristics - FFD, BFD, WFD and AWFD, resulting in a reduction of approximately 73% of physical servers, when compared with mapping each trace to a single physical server. It also presented the highest number of migrations, migrating approximately 90% of the virtual machines each consolidation event, and highest number of reconfiguration delay breaks. The highest maximum reconfiguration delay was presented by heuristic NFD, with a value of 1.7 (217.6 seconds according to the hypothetical scenario), but close to other heuristics such as FFD, BFD and WFD. TS resulted in good consolidation, but requiring a number of migrations higher than the heuristics. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8, 0.6 and 0.4 with 0 breaks. With threshold 0.2, DSCCRD resulted in 5 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 0.5. In comparison to LP, DSCCRD uses between 3% to 17% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.4. For threshold 0.2 it uses 31% more physical servers in order to minimize the number of reconfiguration delay breaks.

Algorithm	Average of number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	11.8	3548	1.3	143 (1.0)
FF	11.1	3811	1.4	48 (1.0)
BF	11.1	3827	1.4	66 (1.0)
WF	11.3	3700	1.3	100 (1.0)
AWF	11.1	3769	1.4	61 (1.0)
NFD	11.6	8822	1.7	1396 (1.0)
FFD	11	8585	1.5	1867 (1.0)
BFD	11	8585	1.5	1867 (1.0)
WFD	11	8900	1.6	1642 (1.0)
AWFD	11	8718	1.4	1885 (1.0)
LP	10.8	12083	1.5	4729 (1.0)
Tabu search	11.1	9376	1.4	697 (1.0)
DSCCRD (1.0)	11.1	1313	1	0 (1.0)
DSCCRD (0.8)	11.4	669	0.8	0 (0.8)
DSCCRD (0.6)	12	303	0.6	0 (0.6)
DSCCRD (0.4)	12.6	121	0.4	0 (0.4)
DSCCRD (0.2)	14.1	5	0.5	5 (0.2)

Table 4.11 – Workload utilization analysis - results for group 1.

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	19,.7	6483	2.3	736 (1.0)
FF	17.8	6450	1.9	792 (1.0)
BF	17.7	6366	1.9	766 (1.0)
WF	18.3	6468	2.3	813 (1.0)
AWF	17.8	6374	1.9	816 (1.0)
NFD	19.8	11244	2.8	3362 (1.0)
FFD	17.5	11170	2.1	4494 (1.0)
BFD	17.5	11177	1.8	4503 (1.0)
WFD	17.5	11241	2	4326 (1.0)
AWFD	17.5	11228	1.8	4547 (1.0)
LP	17	12930	2.3	5819 (1.0)
Tabu search	19.3	8427	1.8	385 (1.0)
DSCCRD (1.0)	18.2	1850	1	0 (1.0)
DSCCRD (0.8)	19	1060	0.8	0 (0.8)
DSCCRD (0.6)	20.1	481	0.6	0 (0.6)
DSCCRD (0.4)	20.9	219	1	30 (0.4)
DSCCRD (0.2)	21.8	122	1	144 (0.2)

Table 4.12 – Workload utilization analysis - results for group 2.



(a) Group 1

(b) Group 2

(c) Group 3

Figure 4.7 – Workload groups for workload utilization analysis

Group 2 represents a workload with middle average utilization and its results are shown in Table 4.12. LP presented the best consolidation, reducing the required number of physical servers in 58%, but requiring migrating approximately 95% of virtual machines each consolidation event. Despite of that, it presented lower maximum reconfiguration delay in comparison to heuristic NFD, but higher than the heuristics with best consolidation results - FFD, BFD, WFD and AWFD. LP also presented the highest number of reconfiguration delay breaks. TS did not present good results regarding consolidation, but was able to get a low maximum reconfiguration delay and fewer

Algorithm	Average number of physical servers used	Number of migrations	Maximum reconfiguration delay	Number of reconfiguration delay breaks (threshold)
NF	31.4	5980	1.6	611 (1.0)
FF	29.1	6681	1.7	1330 (1.0)
BF	29.1	6810	1.7	1363 (1.0)
WF	29.7	6529	1.7	1236 (1.0)
AWF	29.1	6843	1.7	1395 (1.0)
NFD	30.7	12055	1.7	2146 (1.0)
FFD	28.8	12132	1.7	3519 (1.0)
BFD	28.8	12132	1.7	3519 (1.0)
WFD	28.8	12168	1.7	3481 (1.0)
AWFD	28.8	12165	1.7	3567 (1.0)
LP	28.8	13206	1.7	2515 (1.0)
Tabu search	28.8	3875	1.5	117 (1.0)
DSCCRD (1.0)	30.2	1343	1	0 (1.0)
DSCCRD (0.8)	30.5	1262	0.8	0 (0.8)
DSCCRD (0.6)	31.5	926	0.6	0 (0.6)
DSCCRD (0.4)	34.3	312	1	102 (0.4)
DSCCRD (0.2)	36.4	56	1	96 (0.2)

Table 4.13 – Workload utilization analysis - results for group 3.

reconfiguration delay breaks. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8, and 0.6 with 0 breaks. With thresholds 0.4 and 0.2, DSCCRD resulted in 30 and 144 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 1.0 in both cases. In comparison to LP, DSCCRD uses between 7% to 18% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2, it uses 23% and 28% more physical servers in order to minimize the number of reconfiguration delay breaks.

Group 3 represents a workload with high average utilization. The results are presented in Table 4.13. In this experiment, LP, some heuristics - FFD, BFD, WFD and AWFD - and TS presented the best consolidation ratio, reducing the number of physical server required in 28%. LP presented the highest number of migrations, migrating approximately 98% of virtual machines each consolidation event. The maximum reconfiguration delay was the same for LP, with a value of 1.7 (217.6 seconds according to the hypothetical scenario), and almost all heuristics, and a little bit smaller for TS, which also presented lowest number of reconfiguration delay breaks. DSCCRD attended the maximum reconfiguration delay for thresholds 1.0, 0.8, and 0.6 with 0 breaks. With thresholds 0.4 and 0.2, DSCCRD resulted in 102 and 96 reconfiguration delay breaks, increasing the maximum reconfiguration delay to 1.0 in both cases. In comparison to LP, DSCCRD uses between 5% to 9% more physical servers in order to attend maximum reconfiguration delays between 1.0 and 0.6. For thresholds 0.4 and 0.2 it uses 19% and 26% more physical servers in order to minimize the number of reconfiguration delay breaks.

In general, the results obtained using LP, TS, and some heuristics (decreasing versions) were quite similar, specially in the workloads with low and high utilization, but LP still requires the highest amount of migrations, migrating approximately all virtual machines in every consolidation event. When the average utilization of the workload increases, the heuristics tend to increase the number of required migrations, approximating to LP. Besides, it seems that the higher the average utilization of the workload, the results become more similar between each other. Nevertheless, despite of having always the highest number of migrations, LP did not get always the highest maximum reconfiguration delay or number of reconfiguration delay breaks. The approximate execution time for each implementation was: 20 seconds for each heuristic, between 12 and 24 hours for LP, 10 minutes for TS and between 15 to 30 minutes for DSCCRD. TS presented best results regarding maximum reconfiguration delay and number of reconfiguration delay breaks in comparison to LP, and the heuristics with best consolidation results. In the group with smallest utilization (group 1), DSCCRD attended thresholds from 1.0 to 0.4, which indicates that the minimum threshold depends on the average utilization of the workload. In the other groups, DSCCRD attended thresholds from 1.0 to 0.6. In the cases that the threshold couldn't be attended, the number of breaks was minimized, with exception of group 3, which presented less breaks with 0.2 than 0.4. However, the decrease in number of migrations and increase in number of physical servers was maintained as the threshold gets smaller. Figure 4.8 presents the additional percentage of physical servers required by DSCCRD with different thresholds for groups 1, 2 and 3 in comparison to LP.

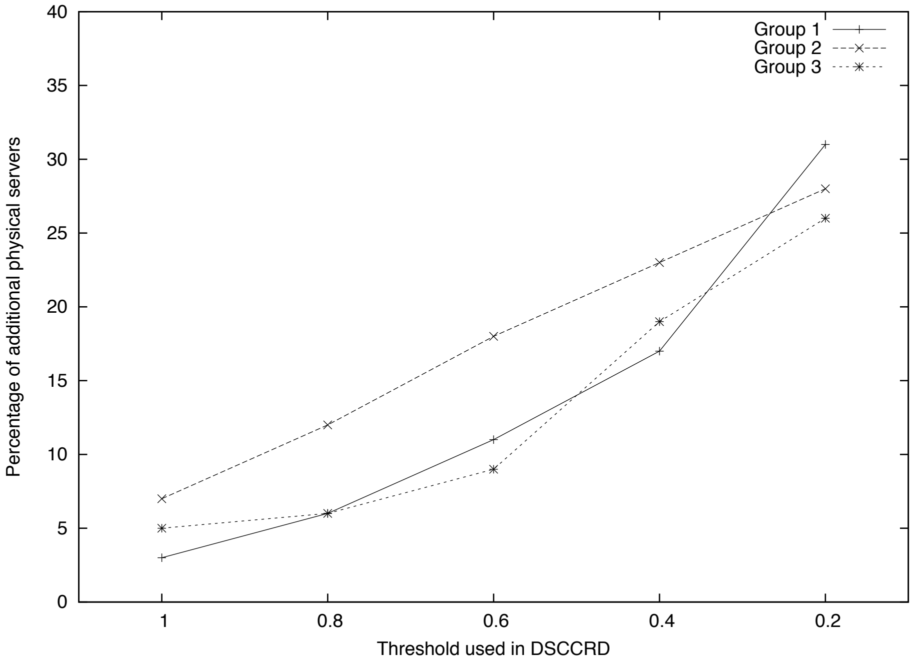


Figure 4.8 – Workload utilization analysis - Percentage of additional physical servers required by DSCCRD

4.6 Closing remarks

Considering all experiments performed, it is possible to verify that LP provides very good results regarding consolidation, however the resulting mapping requires migrating almost every virtual machine in each consolidation event, which can result in a considerable degradation in all virtual machines performance. In most cases, heuristics provided consolidation ratios comparable with LP, requiring less migrations. However, it does not direct reflect in a decrease in reconfiguration delays. The TS implementation presented interesting results. It usually does not provide the best results in consolidation, but usually achieves low reconfiguration delays.

The DSCCRD algorithm achieved the proposed requisites defined in the beginning of the work of providing higher control over reconfiguration delays. The algorithm was able to attend thresholds down to 0.6 in all cases, and 0.4 in a particular case which presented lower utilization. Possibly, using workloads with lower average utilization ratio, DSCCRD could attend even lower thresholds. When the threshold couldn't be achieved, the number of reconfiguration breaks was minimized, incurring in an increase in the maximum reconfiguration delay. The algorithm focus on harming some virtual machines with higher reconfiguration delays, in order to have a higher rate of reconfiguration delays below the given threshold. The DSCCRD algorithm presents a stable behavior, reducing the number of migrations and increasing the amount of physical servers required, when the threshold is decreased. The maximum percentage of additional physical servers required by DSCCRD on average was 20% in the experiments performed using 0.6 as threshold (76.8 seconds according to the hypothetical scenario).

5. Conclusion

Dynamic server consolidation presents high gains in virtualized data centers, with a significant reduction in the amount of required resources, and consequently a reduction on the average cost to maintain the data center. However, it can also have an impact on applications' performance. In order to attend changes in applications' demand, the mapping of virtual machines to physical servers might be modified, requiring migrating virtual machines between physical servers. The concurrent migration of virtual machines can result in an increase in the delay to complete the reconfiguration of each virtual machines to its new capacity. A high value of reconfiguration delay might harm applications' performance, since the application can require the additional capacity before the reconfiguration process is complete.

Related works tackle this problem focusing on minimizing the number of migrations performed, or the individual cost of each migration. However, they do not consider the underlying network connecting all physical servers, or the corresponding delay incurred to applications to complete the reconfiguration of virtual machines. Besides, it can be observed by the experiments performed that minimizing the number of migrations does not represent directly a reduction in the reconfiguration delay. Therefore, another approach should be considered.

This work analyzed algorithms for dynamic server consolidation, obtaining results regarding the number of physical servers used, the number of migrations required, and the corresponding maximum reconfiguration delay for each algorithm. The results indicate that the algorithms provide good consolidation ratio, minimizing considerably the number of required resources to attend the workload. However, in order to accomplish it, a significant amount of migrations is required, and there is no guarantees regarding the maximum delay to complete the reconfiguration of virtual machines. This elevated number of migrations indicate higher reconfiguration delays, but some results identified that solely minimizing the number of migrations does not guarantee a low reconfiguration delay.

An algorithm was proposed to enable higher control over the maximum reconfiguration delay, while also using the minimum amount of resources as possible. The algorithm was evaluated and presented results that attended the maximum reconfiguration delay threshold defined as input. When the threshold could not be attended, the algorithm minimizes the number of virtual machines that have a reconfiguration delay above the given threshold. The results indicate that the algorithm minimizes the number of migrations and uses more physical servers as the threshold gets smaller. The approximate percentage of additional machines required by the algorithm in comparison to LP (algorithm that presented best consolidation results), ranged between 3% and 22% in the experiments performed, when the algorithm was able to attend the specified threshold.

The contributions of this work are:

- modeling of the migration costs resulted from dynamic server consolidation using the max-min fairness model, simulating the behavior of concurrent migrations in the network;

- proposal of an algorithm for dynamic server consolidation using the tabu search metaheuristic;
- proposal of an algorithm for dynamic server consolidation that limits the maximum reconfiguration delay in virtual machines, and reduces the number of reconfiguration breaks;
- proposal of a generator of synthetic workloads, based on common data center utilization patterns;
- evaluation of typical dynamic server consolidation algorithms and the proposed algorithm for controlled reconfiguration delays regarding the average amount of physical servers used, number of required migrations, and reconfiguration delay of virtual machines.

The algorithm proposed provides higher guarantees in applications' performance, ensuring that changes in virtual machines capacity will be attended in a maximum delay. The definition of a maximum reconfiguration delay enables managers to more efficiently identify when to perform changes in virtual machines, in order to avoid applications' starvation for resources.

BIBLIOGRAPHY

[ACH04] Achterberg, T. "SCIP - a framework to integrate constraint and mixed integer programming", Technical Report 04-19, Zuse Institute Berlin, Germany, 2004, 28p.

[AMD05] AMD. "AMD64 virtualization codenamed pacifica technology: secure virtual machine architecture reference manual". White Paper, 2005, 124p.

[AND02] Andrzejak, A.; Arlitt, M.; Rolia, J. "Bounding the resource savings of utility computing models", Technical report HPL-2002-339, HP Laboratories Palo Alto, USA, 2002, 22p.

[BAR03] Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. "Xen and the art of virtualization". In: 19th ACM Symposium on Operating Systems Principles, 2003, pp. 164–177.

[BER92] Bertsekas, D.; Gallager, R. "Data networks". Prentice Hall, 1992, 592p.

[BIC06] Bichler, M.; Setzer, T.; Speitkamp, B. "Capacity planning for virtualized servers". In: Workshop on Information Technologies and Systems, Milwaukee, 2006, pp. 1–6.

[BOB07] Bobroff, N.; Kochut, A.; Beaty, K. "Dynamic placement of virtual machines for managing SLA violations". In: 10th IFIP/IEEE International Symposium on Integrated Network Management, Germany, 2007, pp. 119–128.

[BUN06] Bunker, G.; Thomson, D. "Delivering utility computing: business-driven IT optimization". Wiley, 2006, 370p.

[BUY09] Buyya, R.; et al. "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility". *Future Generation Computer Systems*, vol. 25-6, June 2009, pp. 599–616.

[CLA05] Clark, C.; Fraser, K.; Hand, S.; Hansen, J. G.; Jul, E.; Limpach, C.; Pratt, I.; Warfield, A. "Live migration of virtual machines". In: 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation, Boston, 2005, pp. 273–286.

[COF96] Coffman Jr, E.; Garey, M.; Johnson, D. "Approximation algorithms for bin packing - a survey". In: Approximation algorithms for NP-hard problems, chapter 2, PWS Publishing, Boston, 1996, pp. 46–93.

[CRE81] Creasy, R. J. "The origin of the VM/370 time-sharing system". *IBM Journal of Research and Development*, vol. 25-5, 1981, pp. 483–490.

[DL05] des Ligneris, B. “Virtualization of Linux based computers: the Linux-VServer project”. In: 19th International Symposium on High Performance Computing Systems and Applications, Canada, 2005, pp. 340–346.

[DUD99] Duda, K. J.; Cheriton, D. R. “Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular”. In: 17th ACM Symposium on Operating Systems Principles, USA, 1999, pp. 261–276.

[FLO06] Floudas, C. C. A.; Pardalos, P. M. “Encyclopedia of optimization”. Springer, 2006, 4626p.

[GOL74] Goldberg, R. P. “Survey of Virtual Machine Research”. *IEEE Computer*, vol. 7-6, 1974, pp. 34–45.

[GUM83] Gum, P. H. “System/370 extended architecture: facilities for virtual machines”. *IBM Journal of Research and Development*, vol. 27-6, 1983, pp. 530–544.

[HOO08] Hoopes, J. “Virtualization for security: including sandboxing, disaster recovery, high availability, forensic analysis, and honeypotting”. Syngress, 2008, 384p.

[ILO07] ILOG. “ILOG CPLEX 11.0”. User’s manual, 2007, 532p.

[IOA05] Ioannis, D. “New algorithm for the generalized max-min fairness policy based on linear programming”. *IEICE Transactions on Communications*, vol. 88-2, 2005, pp. 775–780.

[JIA06] Jiang, X. “Enabling internet worms and malware investigation and defense using virtualization”, PhD Thesis, Purdue University, USA, 2006, 139p.

[KEA04] Keahey, K.; Doering, K.; Foster, I. “From sandbox to playground: dynamic virtual environments in the grid”. In: 5th IEEE/ACM International Workshop on Grid Computing, USA, 2004, pp. 34–42.

[KEA05] Keahey, K.; Foster, I.; Freeman, T.; Zhang, X. “Virtual workspaces: achieving quality of service and quality of life in the grid”. *Scientific Programming*, vol. 13-4, 2005, pp. 265–275.

[KHA06] Khanna, G.; Beaty, K.; Kar, G.; Kochut, A. “Application performance management in virtualized server environments”. In: 10th IEEE/IFIP Network Operations and Management Symposium, 2006, pp. 373–381.

[KOC04] Koch, T. “Rapid mathematical programming”, PhD Thesis, Technische Universität Berlin, 2004, 217p.

[KOL06] Kolyshkin, K. “Virtualization in Linux”. Technical report, OpenVZ, available at http://ftp.openvz.org/doc/openvz-intro.pdf, 2006, 5p.

[KOU77] Kou, L.; Markowsky, G. “Multidimensional bin packing algorithms”. *IBM Journal of Research and development*, vol. 21-5, 1977, pp. 443–448.

[LI98] Li, Q. “Java virtual machine - present and near future”. In: 25th Technology of Object-Oriented Languages and Systems Conference, Washington, DC, USA, 1998, pp. 480–480.

[LIU04] Liu, X.; Zhu, X.; Singhal, S.; Arlitt, M. “Adaptive entitlement control of resource containers on shared servers”, Technical report HPL-2004-178, HP Laboratories Palo Alto, USA, 2004, 16p.

[LOD04] Lodi, A.; Martello, S.; Vigo, D. “TSpack: a unified tabu search code for multi-dimensional bin packing problems”. *Annals of Operations Research*, vol. 131-1, 2004, pp. 203–213.

[NAC08] Nace, D.; Nhatdoan, L.; Klopfenstein, O.; Bashllari, a. “Max-min fairness in multi-commodity flows”. *Computers & Operations Research*, vol. 35-2, 2008, pp. 557–573.

[NAN05] Nanda, S.; cker Chiueh, T. “A survey on virtualization technologies”, Technical report TR-179, Stony Brook University, USA, 2005, 42p.

[NEI06] Neiger, G.; Santoni, A.; Leung, F.; Rodgers, D.; Uhlig, R. “Intel virtualization technology: hardware support for efficient processor virtualization”. *Intel Technology Journal*, vol. 10-3, 2006, pp. 167–178.

[NEL05] Nelson, M.; Lim, B.; Hutchins, G. “Fast transparent migration for virtual machines”. In: USENIX Annual Technical Conference, USA, 2005, pp. 391–394.

[NN05] Neville-Neil, G. V.; McKusick, M. K. “The design and implementation of the FreeBSD operating system”. Addison-Wesley, 2005, 683p.

[PAD07] Padala, P.; Shin, K. G.; Zhu, X.; Uysal, M.; Wang, Z.; Singhal, S.; Merchant, A.; Salern, K. “Adaptive control of virtualized resources in utility computing environments”. *ACM SIGOPS Operating Systems Review*, vol. 41-3, 2007, pp. 302–315.

[PAR10] Parallels. “An introduction to OS virtualization and Parallels Virtuozzo containers”. White Paper, 2010, 10p.

[POP74] Popek, G. J.; Goldberg, R. P. “Formal requirements for virtualizable third generation architectures”. *Communications of the ACM*, vol. 17-7, 1974, pp. 412–421.

[RAM99] Ramm, D. “VMware: not your granddaddy’s virtual machine”. In: 3rd annual conference on Atlanta Linux Showcase, Berkeley, CA, USA, 1999, pp. 30–30.

[RAP04] Rappa, M. A. “The utility business model and the future of computing services”. *IBM Systems Journal*, vol. 43-1, 2004, pp. 32–42.

[ROS04] Rose, R. “Survey of system virtualization techniques”. Technical Report, Available at http://www.robertwrose.com/vita/rose-virtualization.pdf, 2004, 15p.

[ROS05] Rosenblum, T., M. Garfinkel. “Virtual machine monitors: current technology and future trends”. *IEEE Computer*, vol. 38-5, 2005, pp. 39–47.

[SMI05] Smith, J.; Nair, R. “Virtual machines: versatile platforms for systems and processes”. Morgan Kaufmann, 2005, 656p.

[SOR07] Soror, A. A.; Aboulnaga, A.; Salem, K. “Database virtualization: a new frontier for database tuning and physical design”. In: 23rd International Conference on Data Engineering Workshop, Washington, DC, USA, 2007, pp. 388–394.

[SPE07] Speitkamp, B.; Bichler, M. “Allocation problems in large scale server consolidation”, Technical report ISR-0001-1922.6, TU M§nchen, Germany, 2007, 56p.

[SUN07] Sun Microsystems. “Sun Grid Compute Utility”. User’s guide, 2007, 96p.

[TUC04] Tucker, A.; Comay, D. “Solaris zones: operating system support for server consolidation”. In: 3rd Virtual Machine Research and Technology Symposium Works-in-Progress, San Jose, CA, USA, 2004, pp. 1–2.

[UHL05] Uhlig, R.; Neiger, G.; Rodgers, D.; Santoni, A. L.; Martins, F. C.; Anderson, A. V.; Bennett, S. M.; Kägi, A.; Leung, F. H.; Smith, L. “Intel virtualization technology”. *IEEE Computer*, vol. 38-5, 2005, pp. 48–56.

[VER07] Verma, A.; Ahuja, P.; Neogi, A. “pmapper: power and migration cost aware application placement in virtualized systems”. In: 9th ACM/IFIP/USENIX International Conference on Middleware, Leuven, Belgium, 2008, pp. 243–264.

[VOG08] Vogels, W. “Beyond server consolidation”. *ACM Queue*, vol. 6-1, 2008, pp. 20–26.

[WAL02] Waldspurger, C. “Memory resource management in VMware ESX server”. In: 5th Symposium on Operating Systems Design and Implementation, Boston, MA, USA, 2002, pp. 181–194.

[WAN05] Wang, Z.; Zhu, X.; Singhal, S. “Utilization of SLO-based control for dynamic sizing of resource partitions”. In: 16th IFIP/IEEE Distributed Systems: Operations and Management, Barcelona, Spain, 2005, pp. 24–26.

[WOO07] Wood, T.; Shenoy, P.; Venkataramani, A.; Yousif, M. “Black-box and gray-box strategies for virtual machine migration”. In: 4th USENIX Symposium on Networked Systems Design & Implementation, Cambridge, MA, USA, 2007, pp. 229–242.

[XU06] Xu, W.; Zhu, X.; Singhal, S.; Wang, Z. "Predictive control for dynamic resource allocation in enterprise data centers". In: 10th IEEE/IFIP Network Operations and Management Symposium, Vancouver, Canada, 2006, pp. 115–126.

[ZHU06] Zhu, X.; Wang, Z.; Singhal, S. "Utility-driven workload management using nested control design". In: American Control Conference, Minneapolis, Minnesota, USA, 2006, pp. 1–6.