Pre-Print Version
R. Jhawar, V. Piuri, " Dependability-oriented resource management schemes for cloud computing data centers," in Handbook on Data Centers, S.U. Khan and A.Y. Zomaya Eds, Springer, 2015

# Dependability-Oriented Resource Management Schemes for Cloud Computing Data Centers

Ravi Jhawar and Vincenzo Piuri
Dipartimento di Informatica
Università degli Studi di Milano
Via Bramante 65, 26013 Crema, Italy
Email: ravi.jhawar@unimi.it, vincenzo.piuri@unimi.it

## 1 Introduction

A major factor in the growth of the Information and Communications Technology has been the widespread use of data centers for deploying and executing web services, business processes, and scientific and e-commerce applications [DFJ2013]. While some data centers are designed to operate a specific business (e.g., Google's search engine), others are used as the backbone infrastructure to deliver computing resources as services to hundreds of users (e.g., Amazon's EC2 service). Relying on data centers for running applications, particularly when resources are delivered to the users as a service, offer significant benefits [SD2010, DFS2012]. For example, applications can benefit significantly from the economy of scale, and users are relieved from buying expensive hardware and software licenses and from maintaining the computing infrastructure.
To ensure resource availability, Quality-of-Service (QoS) and dependability to hundreds of applications in the modern data centers under fluctuating workloads, server failures, and network congestion, dedicated servers are allocated to applications and server capacity is often over-provisioned. However, the use of dedicated hardware not only leads to poor energy usage, but also made it difficult to react to system changes. Furthermore, the growing number of under-utilized servers increases the data center's operating costs such as system management, energy consumption of servers, and network and cooling infrastructure costs.

In the last decade, the virtualization technology has emerged as a very effective approach to address these issues by de-coupling physical servers from the resources needed by applications. In particular, virtualization provides an efficient way to insulate and partition server's resources so that only a portion of them can be utilized by an application. It also provides a greater flexibility and control over resource management, allowing for dynamic adjustment of CPU and memory usage, and live migration of virtual machines among physical servers (e.g., [HDL2011, HLM2011]). Nowadays, Cloud computing is the most popular approach to create virtualized environments for application execution in distributed data centers.

Deploying virtualized services in data centers, including those based on Cloud computing, create new resource management problems, such as optimal placement of virtual machines. Existing solutions focus mainly on placing and adaptive managing virtual machines in order to (i) reduce energy consumption costs and maximize profits for the data center owner, and (ii) improve the performance and QoS of applications. Only a few approaches are available in the literature to improve dependability – fault tolerance and security – of applications (e.g., [AJJ2013, AJJ2013a, AJP2013, BBK2002, BCD2009, DDF2005, DFJ2008, DFJ2012DLP2012, DLP2013]). This chapter aims to investigate resource management schemes that improve dependability (both fault tolerance and security) and performance of applications in virtualized data centers. In particular, we mainly focus on initial virtual machine

placement and runtime adaption schemes developed for the Infrastructure-as-a-Service (IaaS) paradigm in Cloud computing data centers.

The remainder of this chapter is organized as follows. Section 2 briefly outlines the failure characteristics of data centers' components. Section 3 introduces the analysis of resource management in data centers by formalizing the virtual machine placement problem and by defining constraints for representing data center owner and users perspectives. In Section 4 we discuss initial virtual machine placement approaches that satisfy dependability and performance constraints at the activation of the applications. Finally, in Section 5 we discuss runtime adaption schemes that balance performance and availability of applications during system operation with possible occurrence of dependability threats. Section 6 provides some concluding remarks.

# 2 System model and failure behavior of data center components

To appreciate the resource management schemes which improve dependability and performance of applications deployed in Cloud computing data centers, in this Section we first review the typical architecture of modern data centers and, then, we analyze the behavior of the various data center components, the main causes of failures, and the impact of failures on applications. This is very useful to analyze the efficacy and efficiency of resource management schemes.

## 2.1 Overview of the data center architecture

A data center interconnects a large number of physical hosts $H$ to form a resource pool that is partitioned into a set of clusters [HB2009]. A cluster $C$ is formed by grouping the hosts that have identical resource characteristics or administrative parameters (e.g., hosts that belong to the same network latency class or geographical location). Each host or server contains multiple processors, storage disks, memory modules and network interfaces. The resource characteristics of each physical host $h \in H$ can be represented using a $D$ dimensional vector $\vec{h} = (h[1], \dots, h[d], \dots, h[D])$, where each dimension represents the amount of host's residual capacity corresponding to a distinct resource type (e.g., CPU, memory, storage, network bandwidth). For simplicity, the resource capacity of hosts can be denoted using normalized values between 0 and 1. For example, host $h$ characterized as $\vec{h} = (\text{CPU, Mem}) = (0.6, 0.5)$ implies that 60% of CPU, 50% of memory on $h$ is available for use. A *hypervisor* is deployed on each host to virtualize its resources: virtual machines of required size are created on appropriate hosts to allocate resources for users' applications.

Hosts are connected using several network switches and routers. The most common network topology of data centers is as follows [HB2009]. Physical hosts, servers or racks of servers are first connected to a Top of Rack switch (ToR), which is in turn connected to two (primary and backup) aggregation switches (AggS). The subsystem formed by the group of servers under an aggregate switch can be viewed as a cluster. An AggS then connects tens of ToRs to redundant access routers (AccR). Thus, each AccR handles traffic from thousands of servers and routes it to Core routers that connect different data centers to the Internet. All links in the data centers commonly use Ethernet as the link layer protocol and redundancy is applied to all network components at each layer in the network topology (except for ToRs). In addition, redundant pairs of load balancers (LBs) are connected to each AggS and mapping between static IP address presented to the users and dynamic IP addresses of internal servers that process user's requests is performed.

Servers and network components are subject to failures during their life, thus affecting the correct operation of applications. In this Section, we review their typical failure behavior in order to understand the redundancy that is needed to overcome critical situations and ensure successful completion of applications.

## 2.2 Failure behavior of servers

Let's consider first the servers' life and their possible failures. Vishwanath and Nagappan [VN2010] have studied server failures and hardware repair behavior using a large collection of servers (nearly 100,000 servers) and corresponding data on part replacement. For example, they mine data relevant to server configuration, when a hard disk has been marked for replacement and when it has been actually replaced, to generate statistical reports on failure behavior. The data repository for their study includes server collection spanning multiple data centers distributed across different countries. Some interesting outcomes of this study are as follows:

- The annual failure rate (AFR) of servers is approximately 8%, and the average number of repairs per each failure-prone server is 2. The remaining 92% of servers do not see any repair events (20 repair/replacement events have been identified in 9 machines over a 14 months period).
- For 8% AFR, repair costs that amount to 2.5 million dollars are approximately spent for 100,000 servers.
- Hard disks are the most failure-prone hardware components and the most significant reason behind server failures. In particular, about 78% of total faults/replacements have been detected on hard disks, 5% on RAID controllers, and 3% due to memory failures. 13% of replacements have been due to a collection of components (not particularly dominated by a single component failure).
- About 5% of servers experience a disk failure in less than one year from the date when it is commissioned (young servers), 12% when they are one year old, and 25% when they are two years old.
- Comparing the number of repairs per machine (RPM) against the number of disks per server in a group of servers (clusters) indicates that (i) there is a relationship in the failure characteristics of servers that have already experienced a failure, and (ii) the number of RPM has a correspondence to the total number of disks on that machine.

This analysis can be used to develop robust fault tolerance mechanisms (e.g., by improving the reliability of hard disks to substantially reduce the number of failures) and resource management schemes (e.g., to improve the availability of a given application, its tasks must not be allocated on the server whose hard disks have already experienced a failure).

## 2.3 Failure behavior of network components

Let's consider now the possible network failures. Gill et al. [GJN2011] have performed a large scale study in data centers. A link failure happens when the connection between two devices on a specific interface is down. A device failure happens when the device is not routing/forwarding packets correctly (e.g., due to power outage or hardware crash). Some interesting outcomes from this study are as follows:

- The overall data center network reliability is about 99.99% for 80% of the links and 60% of devices.

- Among all the network devices, ToRs are most reliable (with a failure rate of less than 5%) and load balancers are least reliable (with failure probability of 1 in 5). The root causes for failures in LBs are dominantly software bugs and configuration errors; moreover, LBs demonstrate frequent but short failures. This observation clearly indicates that low-cost commodity switches such as ToRs and AggS provide sufficient reliability.
- The links forwarding traffic from LBs have highest failure rates; links higher in the topology (e.g., connecting AccRs) and links connecting redundant devices have second highest failure rates.
- Network redundancy reduces the median impact of failures (in terms of number of lost bytes) by only 40%, as opposed to the common belief that redundancy almost completely masks failures from applications.
- The estimated median number of packets lost during a failure is 59K and median number of bytes is 25MB (average size of lost packets is 423Bytes). Based on prior measurement studies (that observe packet sizes to be bimodal with modes around 200Bytes and 1,400Bytes), it is estimated that most lost packets belong to the lower-level functions (e.g., ping messages or ACKs).

## 2.4 Analysis of the impact of failures on applications

A detailed analysis of the impact of various component failures on applications as well as the definition of the impact boundary of a given failure (e.g., fault region) are useful to improve the dependability of applications [GY2010, JPS2013, JP2013a, STT2008]. In [JPS2012] such analysis has been integrated within the resource management schemes to provide fault tolerance support to applications in the Cloud computing paradigm. For example, if a switch failure disconnects a rack of servers, then replicas of a given applications can be allocated in different clusters in order to increase the application's failure independence.

Jhawar and Piuri [JP2012] use the notion of fault trees and provide a hierarchical model to analyze the impact of component failures. As an example, let's consider the impact of power failures (other failures can similarly be analyzed in a straight-forward manner). Assume that a data center receives power via an uninterrupted power network, and a redundant distribution unit (DU) is deployed for each cluster within the data center. A DU hence provides power to all the servers within a cluster. In this context, a failure in the DU is independent of other DUs and the central power supply. The fault tree for power failures include the conditions $(Power1 \wedge Power2)$ for redundant power units of a server, $(DU1 \wedge DU2)$ for a cluster, and $(Power1 \wedge Power2) \vee (DU1 \wedge DU2) \vee (Central\ Power\ Supply)$ for the power failure of the whole system. An application failure happens when this latter condition evaluates *true.* We note that the boundary (server, cluster, data center) of the impact of each failure can also be identified using this approach.

The fault tree model can also be extended to understand the failure behavior of applications by integrating it with Markov chains, where the probability values permit to quantitatively analyze the fault tolerance of the given application in the envisioned data center. Building on the notion of fault trees and Markov chains, Jhawar and Piuri [JP2012] identify three main Deployment Levels $DL$ for applications. A deployment level is the smallest subsystem within the infrastructure where the replicas of a users' application are deployed to ensure dependability. Deployment levels correspond to fault regions in the data center, and describe how replicas of applications can be allocated, depending on its fault tolerance, performance, availability, and reliability goals. The three deployment levels are as follows.

- *Multiple machines within a cluster*: Replicas of an application can be placed on hosts that are connected by a ToR switch, that is, in a LAN. This deployment provides benefits in terms of low

latency and high bandwidth but offers least failure independence. Replicas cannot communicate and execute the dependability protocol upon a single switch failure, or a failure in the power distribution unit results in an outage of the entire application.

- *Multiple clusters within a data center*: Replicas of an application can be placed on hosts that belong to different clusters in the same data center, that is, connected via a ToR switch and AggS. This deployment still provides moderate benefits in terms of latency and bandwidth, and offers higher failure independence. The replicas are not bound to an outage with a single power distribution or switch failure.
- *Multiple data centers*: Replicas of an application can be placed on hosts that belong to different data centers, that is, connected via a ToR switch, AggS and AccR. This deployment has a drawback with respect to high latency and low bandwidth, but offers a very high level of failure independence. A single power failure has least effect on the availability of the application.

A partially-ordered hierarchy can be defined between the deployment levels. For example, a data center is a larger subsystem or deployment level when compared to a cluster. A transitive closure indicating that "contains-in" relationship also exists on the hierarchy of deployment levels. For example, a host can be part of a cluster that in turn exists in the data center. Intuitively, availability (failure independence) of an application increases with the increase of the deployment level; that is, the availability of an individual host is smaller than the availability of a cluster, which is still smaller than the availability of a data center. On the other hand, the network latency increases with the increase of the deployment level; that is, hosts in the same rack have lower network latency than hosts across different clusters. Hence, if $Lat(DL)$ denotes the maximum latency between two hosts in the deployment level $DL$, then the virtual machine placement algorithm can decide a suitable $DL$ based on users desired performance goals (e.g., in terms of the expected response time). In the next Sections, we discuss how to integrate this analysis within resource management schemes to improve fault tolerance, security, and performance of applications.

## 3 Resource management in data center environments

The problem of resource management in data center environments has gained a wide attention from the research community and the industry. In this Section, we model the resource management problem with specific reference to improving dependability and performance of users applications deployed in the data centers using virtual machines.

Resource management for Cloud-based data centers can be modeled as a placement problem in which virtual machines are allocated on the data center's hosts [HDL2011, HLM2011, JP2013], having been the applications mapped on the appropriate virtual machine templates available in the data center environment. Existing solutions, particularly for services that provision on-demand resources to users, primarily focus on making virtual machine *placement decisions* at two distinct levels: (i) *initial* virtual machine *allocation* and (ii) *runtime adaption* of current virtual machine allocation [MFD2011].

Based on user's requirements and failure characteristics of the envisioned data center, a set of dependability and performance constraints are specified (e.g., constraint specifying that replicas of the user's application be allocated on two different physical hosts to avoid single points of failure). The initial resource allocation process identifies the physical hosts on which the requested virtual machines can be allocated such that all the placement constraints are satisfied. Once the required virtual machines are created and delivered to the user, the runtime adaption process monitors the system and resizes virtual machines or migrates them to other physical hosts in order to meet the predefined goals (e.g., energy conservation), while satisfying the placement constraints. While the objective of most resource management algorithms in this context has been to maximize the service provider's goals (e.g., through

resource consolidation, load balancing, satisfaction of SLAs), we will provide a broader perspective which encompasses both the provider's and the users' views, balancing all needs in a comprehensive way.

In this Section, we formulate and categorize various placement constraints that can be used to specify dependability – fault tolerance and security – and performance related conditions during resource management. Within this framework, in Section 4 we will study resource management schemes for placing applications in the Cloud computing environment at the initial deployment. Then in Section 5 we will discuss dynamic adaptation of the applications placement to deal with changing working status of the architecture components, balancing dependability and performance of users' applications.

Let us start by defining a mapping function $p: V \rightarrow H$ that maps each virtual machine $v \in V$ on a physical host $h \in H$ in the data center. Notation $p(v) = h$ denotes that virtual machine $v$ has been allocated on physical host $h$.

Placement constraints can be distinguished in three main categories.

- *Global constraints* that must be satisfied across all the hosts and virtual machines in the data center. These constraints essentially specify the conditions necessary to maintain a consistent system state.
- *Infrastructure-oriented constraints* that are specified by the data center owners and service providers who build their services on top of the data center network (e.g., a Cloud-based IaaS provider). These constraints aim to ensure security and quality of service.
- *Application-oriented constraints* that are specified by the users who use data center's resources to execute their applications (e.g., application owners who use the IaaS service). These constraints allow users to specify conditions relevant to the specific configuration of their dependability mechanisms, and consequently, to improve availability, reliability, and response time of their applications.

These constraints have been introduced and formalized in [JP2012]. Other solutions also formulate the resource allocation problem by including one or more constraints discussed in this Section (e.g., [BBB2011, AAP2010, SBB2013]). Some additional constraints have also been considered in the past, depending on the specific formulation of the allocation problem. For example, Shi et al. [SBB2013] include FULL and SEC constraints: FULL specify that either all or none of the virtual machines in a given request must allocated, while SEC requires that a given physical host can be assigned only the virtual machines from the same request and no other request. Similarly, Zheng et al. [ZLX2013] include constraints relevant to the maintenance schedule of the physical hosts. In particular, they include (i) a constraint that confines each host to finish its maintenance activity before a specified deadline and (ii) a constraint specifying that any host does not execute any maintenance activity after its deadline.

For simplicity, in the rest of this chapter, we define constraints using virtual machine and host identifiers. However, these constraints can also be specified in a straight-forward manner for groups of virtual machines or hosts, clustered by means of some of their properties (e.g., all the hosts that belong to a given cluster or a deployment level).

## 3.1 Global constraints

Global constraints include the classical resource capacity constraints. Similarly to the representation of the resource characteristics $\vec{h} = (h[1], \dots, h[d], \dots, h[D])$ of a host $h \in H$, the amount of each resource type $d$ in a virtual machine $v \in V$ can be represented by $\vec{v} = (v[1], \dots, v[d], \dots, v[D])$. The resource capacity constraint states that the amount of resources consumed by all the virtual machines mapped on a

single physical host cannot exceed the total capacity of that host in any dimension $d$. In general, this constraint is formulated as follows: for all the virtual machines $v \in V$ and physical hosts $h \in H$ in the system, mapping function $p: V \rightarrow H$ must satisfy

$$\forall\, h \in H, d \in [1..D] \qquad \sum_{v \in V | p(v)=h} v[d] \leq\ h[d]$$

This constraint is taken into account by most solutions existing in the literature, and is necessary to ensure that the data center operates correctly. Some variants of this constraint have also been considered in the literature, depending on the problem context. For example, solutions for server consolidation require that the amount of resources consumed by a virtual machine when placed in isolation on a host or with other co-hosted virtual machines may be different. When multiple virtual machines are placed on a host, several memory pages can be shared between them thus reducing the overall memory requirements; conversely, the hypervisor or host operating system may consume additional CPU cycles or I/O bandwidth for resource scheduling. Similarly, virtual machines may interfere with each other and consume higher amounts of shared resources (e.g., the L2 cache during context switching).

To avoid performance degradation and inconsistent system state due to the above factors, the data center owner can define an upper bound on the resource capacity of each host that can be used by users' virtual machines. The resource capacity constraint then ensures that virtual machines are allocated on a host only if its capacity in any dimension does not exceed the upper bound or $threshold$ value, that is,

$$\forall\, h \in H, d \in [1..D] \qquad \sum_{v \in V | p(v)=h} v[d] \leq (h[d] * threshold[d])$$

The $threshold[d]$ can be specified using percentage or normalized values between 0 and 1.

## 3.2 Infrastructure-oriented constraints

The data center owner may need to impose restrictions on the mapping function to improve the security, operational performance and reliability of the data center. While a number of conditions can be introduced depending on the specific system architecture, here we discuss two representative infrastructure-oriented constraints: *forbid* and *count.*

**Forbid.** To improve security, the data center owner may need to dedicate a set of hosts only to execute system-level services (e.g., the access control engine or reference monitor) and, as a consequence, it may need to specify that the mapping function does not allocate users' virtual machines on those hosts. In this context, the forbid constraint prevents a virtual machine $v$ from being allocated on a physical host $h$. When the data center owner defines a set $Forbid = \{(v_i, h_j) |\ v_i \in V,\ h_j \in H\}$ specifying the virtual machines $v_i \in V$ that must be forbidden from being allocated on hosts $h_j \in H$, the allocation algorithm guides the mapping function $p: V \rightarrow H$ to satisfy the following condition:

$$\forall\, v \in V, h \in H \qquad (v, h) \in\ Forbid\ \Rightarrow\ p(v) \neq h$$

**Count.** The performance of a host degrades as the number of virtual machines co-hosted on it increases. For example, the performance of a storage disk decreases if the number of I/O intensive applications in the virtual machines increases; similarly, the network traffic from a host, virtual machine management costs, and CPU utilization costs gradually increase. To avoid such conditions, the count constraint allows

the data center owner to limit the number of virtual machines that can be allocated on a given host. When the data center owner defines $count_h$ as the maximum number of virtual machines allowed on host $h$, the mapping function $p: V \rightarrow H$ ensures that the following condition is satisfied:

$$\forall\, v \in V, h \in H \qquad |\{v \in V\}: p(v) = h\}| \;\leq\; count_h$$

## 3.3 Application-oriented constraints

Based on the dependability policy, users may need to impose a set of restrictions on the placement of their virtual machines in the data center. For example, suppose that a user applies a replication technique to increase the reliability and availability of her application. She may then need to impose a set of conditions on the system parameters and relative placement of her virtual machines to correctly implement the dependability policy while satisfying her performance goals. We discuss three representative application-oriented constraints that can be used to realize the aforementioned conditions: *restrict*, *distribute* and *latency*.

**Restrict.** To ensure survival and possible continuous operation of an application even in the case of failures, it can be replicated in the data center so as to have at least one replica able to proceed in the application activities. A user may wish to allocate each replica of her application in a specific region (e.g., cluster, availability zone) of the data center. To achieve this, the user may leverage the restrict constraint which limits a virtual machine $v \in V$ on being allocated only on a specified group of physical hosts $H' \in H$. We note that the analysis of the failure behavior of the data center components may be beneficial in this context since each replica can be placed in a different failure zone or deployment level (see Section 2.4), thus increasing the failure independence of the application replicas. When a user defines the set $Restr = \{(v_i, H'_j) | v_i \in V \wedge H'_j \subseteq H\}$, the mapping function $p: V \rightarrow H$ ensures the following condition:

$$\forall\, v_i \in V, H'_j \in 2^H \qquad \left(v_i, H'_j\right) \in Restr \;\Rightarrow\; p(v_i) \in H'_j$$

The restrict constraint is also beneficial in other scenarios such as enforcement of privacy policies and improvement of applications performance. For example, a user may leverage the restrict constraint to satisfy mandatory government enforced obligations (e.g., EU Data Protection 95/46/EC Directive) requiring virtual machines to be always located within a given community area (e.g., within EU countries). Similarly, to improve the application's performance, a user may require the mapping function to place her virtual machines on the hosts whose geographical location is closest to her customers.

**Distribute.** A replication-based fault tolerance scheme inherently requires that each replica be placed on different physical hosts in order to avoid single points of failure. If a user replicates her application on two virtual machines, and if both the virtual machines are allocated on the same host, then a failure in the host results in an unavailability of the user's application. To avoid such situations, the distribute constraint allows a user to specify that two virtual machines $v_i$ and $v_j$ must never be allocated on the same host at the same time. Given the set $Distr = \{(v_i, v_j) | v_i, v_j \in V' \subseteq V\}$ of pairs of virtual machines that cannot be deployed on the same host, the mapping function $p: V \rightarrow H$ ensures the following:

$$\forall\, v_i, v_j \in V' \subseteq V, h \in H \qquad (v_i, v_j) \in Distr \;\Rightarrow\; p(v_i) \neq p(v_j)$$

**Latency.** To ensure that the response time of an application does not exceed a maximum value, a user may want to specify the latency allowed between each application replica. For instance, in a checkpoint-based reliability mechanism, the state of the backup virtual machine instance must be frequently updated with that of the primary instance to maintain the system in a consistent state. This task involves high

amounts of message exchanges, and hence an upper bound in the network delay is essential; otherwise, the wait-time of the primary instance during which the state transfer to the backup takes place may increase significantly and the overall availability of the application may be reduced.
For simplicity's sake, let's assume that the network latency between two virtual machines is equal to the network latency between the physical hosts on which they are deployed. The latency constraint forces the mapping function to allocate two virtual machines $v_i, v_j \in V$ such that the network latency $latency(p(v_i), p(v_j))$ between them is less than a specified value $T_{max}$. Given the set $MaxLatency = \{(v_i, v_j, T_{max}) | v_i, v_j \in V\}$ that specifies the acceptable network latency $T_{max}$ between two virtual machine instances $v_i$ and $v_j$, the mapping function $p: V \rightarrow H$ ensures the following condition:

$$\forall\, v_i, v_j \in V, (v_i, v_j, T_{max}) \in MaxLatency \qquad latency(p(v_i), p(v_j)) \leq T_{max}$$

# 4 Initial allocation of virtual machines in data center environments

In this Section, we discuss resource allocation schemes that, for each user request, identify the physical hosts on which the requested virtual machines can be allocated while satisfying the placement constraints. The solutions discussed here perform initial allocation considering the fault tolerance, security, and performance constraints, and aiming at maximizing the data center efficiency. We first analyze the solution by Jhawar et al. [JPS2012] in detail (Section 4.1) since it satisfies all the constraints described in the previous Section, and then provide an overview of other state-of-art schemes (Section 4.2).

## 4.1 A comprehensive scheme for virtual machines allocation

To address the needs of optimum data center management in Cloud-based environments, while satisfying both data center owner's constraints and users' goals, the approach presented in [JPS2012] performs the initial placement of virtual machines by designing the mapping function $p: V \rightarrow H$ to meet the following objectives:

- reduce the energy consumption and operational costs by consolidating virtual machines on physical hosts such that the number of *free* hosts is maximized;
- reduce the load variance of physical hosts across all the clusters in the Cloud in order to improve the performance and resilience of the data center.

Besides, the mapping function is also structured to satisfy the resource capacity, forbid, count, restrict, distribute, and latency constraints, thus satisfying application's fault tolerance, security and performance goals.

The allocation algorithm works as follows. Each time a request to allocate new virtual machines arrives, the data center is analyzed to identify the clusters and the physical hosts that can be used for resource allocation. For each shortlisted physical host, the set of virtual machines that can be allocated on that host are identified. In particular, *to reduce the load variance between different clusters*, the new virtual machines are created in the cluster that has highest amount of available resources.
A priority queue $CL$ is built based on the resource availability in each cluster, and then, the cluster $C$ with maximum resource availability is extracted from $CL$. *To reduce the energy consumption costs*, each host within the selected cluster is analyzed to allocate as many virtual machines on that host as possible. This heuristics maps virtual machines on fewer physical hosts, thus leaving the remaining hosts to operate in the *cold-standby* mode.

A priority queue $VMreq$ is created based on the vector dot-product [SKM2008] value of virtual machines with respect to the current host and entries from $VMreq$ are extracted in the decreasing order of the dot-product values and analyzed for performing the final allocation.
Therefore, the algorithm can be viewed as a two-step process: (i) select the least-used cluster and (ii) allocate virtual machines on its hosts using the dot-product method, and satisfies both the aforementioned objectives.

In this allocation algorithm appropriate checks are introduced to guide the mapping of virtual machines on physical hosts as follows. Since the *forbid* and *restrict* constraints define conditions on the association between virtual machines and physical hosts, corresponding checks are applied mainly while building the $VMreq$ priority queue (i.e., when analyzing the suitability of allocating a given virtual machine on the current physical host). Hence, a temporary set $V'$ that contains all the virtual machines that must be allocated is created, and a virtual machine $v$ from set $V'$ is discarded if: (i) an entry $(v, h) \in Forbid$ exists or (ii) a set of hosts is specified in the $Restr$ set for the virtual machine $v$ but the considered host $h$ does not belong to that set. This check allows the allocation algorithm to enforce the forbid and restrict constraints.

The *capacity* and *count* constraints deal with the resource usage of the individual physical hosts. To enforce the capacity constraint, the residual resource capacity of each physical host is maintained, based on the threshold values specified by the data center owner. A virtual machine is allocated on a physical host if the resource requirements of the virtual machine are less than the residual capacity of that host in all the dimensions. Each time a virtual machine is allocated on a host, its residual capacity value is updated.
To ensure the count constraint, the vector representation of hosts and virtual machines is extended with respect to Section 2.1 by adding a new dimension on each physical host that denotes the number of virtual machines that can be allocated on that host $h[D+1] = count_h$. Similarly, the $[D+1]^{th}$ dimension of each virtual machine is initialized to 1, and the count control is enforced with the capacity constraint.

The *distribute* constraint is enforced by verifying whether a given host $h$ already contains a virtual machine $v_j$ for which the user has specified a condition $(v, v_j) \in Distr$. The algorithm simply does not consider the virtual machine $v$ when working on the allocation for the host $h$ if $p(v_j) = h$ is true.

Finally, the allocation algorithm enforces the *latency* constraint based on the notions of *forward allocation* and *reserve list.* If a virtual machine $v_i$ satisfies all other constraints when considered for allocation on a host $h$, but is related to other virtual machines $v_j \in V_i \subset V$ by latency constraints, then the virtual machine $v_i$ cannot be actually allocated on the host $h$ until an allocation for all $v_j \in V_i$ is found. Therefore, the allocation algorithm tentatively allocates the virtual machines by saving pair $(v_i, h)$ in the $Reserve\_list$, and calls the function $Forward\_Allocate$ to find an allocation for the other virtual machines $v_j \in V_i$. The $Forward\_Allocate$ function first determines the virtual machines $v_j \in V_i$ that are related to $v_i$ by the latency constraints and stores them in a priority queue. Each virtual machine from the priority queue is then extracted in the increasing order of $T_{max}$, and the set of hosts that can be reached from the current host $h$ within the specified network threshold time (and not conflicting w.r.t. the restrict and forbid constraints) is selected. The capacity, count and distribute constraints are then verified for each shortlisted host, and the corresponding allocation is saved in the $Reserve\_list$. The $Forward\_Allocate$ function is recursively called until an allocation for all the virtual machines is determined. If an allocation is not obtained, the entries from the $Reserve\_list$ are removed, and the algorithm resumes from another host.

## 4.2 Other schemes for virtual machines allocation

In the literature there are other approaches for the initial allocation of virtual machines in data centers. Existing solutions either use Constraints Programming (CP) solvers (e.g., [BBB2011]) or design heuristics (e.g., [SBB2013]) to obtain the placement solutions. In general, while the goal is to maximize the goals of the data center owner, each solution takes a different approach in modeling the context of the system and, consequently, defines different objective functions and placement constraints. In this sub-section, we discuss four representative solutions to understand different dimensions in which the overall problem has been studied.

Bin et al. [BBB2011] combine the Hardware Predicted Failure Analysis alerts (HwPFA) and live migration techniques to provide a high availability solution. On predicting hardware failure alerts, a trigger to the cluster management system is provided so as to move the virtual machines from the failing host to other working hosts. Depending on the allowed response time, either a complete live relocation of the virtual machine is performed so that continuous operation of the applications is ensured, or a cold relocation is performed by starting a new virtual machine on a working host with a small interruption.
The goal of their solution is to provide *k*-resiliency to users applications while reducing the resource consumption costs. We note that *k*-resiliency allows a given application or virtual machine to tolerate up to *k* host failures. In general, to ensure *k*-resiliency, a feasible solution should dedicate at least *k* hosts for the given virtual machine (in addition to the virtual machine itself). The proposed approach introduces the notion of *shadow virtual machines* that denotes the location or host where a virtual machine can be evacuated (i.e., a shadow serves as a placeholder) and aim to construct shadow placement constraints so to reduce the overall resource requirements to a value less than (*k*+1). To achieve this, they transform the placement problem with *k*-resiliency constraints into a constraint satisfaction problem including the notion of *shadow virtual machines*, and solve it using a constraint programming engine. All the shadows of a given virtual machine and the virtual machine itself are *anti-colocated* (equivalent to the Distribute constraint discussed in Section 3.3). In addition, they employ a scheme of numbering shadows and failures in a way that identifies the possible overlaps of actual virtual machine evacuations. In particular, each failing host is assigned a unique index (1 through *k*) and each shadow of a virtual machine is assigned a unique index. Upon failure of a host indexed *i*, the virtual machines on that host are evacuated to the location of their *i*-th shadow. The placement constraints are defined to specifically numbered shadows and virtual machines that may overlap following host failures, thus reducing the number of backup hosts required. For example, virtual machines that are placed on different hosts cannot be evacuated together to shadows with the same index (as each host would be assigned a different failure index); therefore, their shadows with same index can overlap.

Machida et al. [MKM2010] consider consolidated server systems and present a method to redundant configuration of virtual machines, in anticipation of host server failures for online applications. They estimate the requisite minimum number of virtual machines according to the performance requirements of the given application, and compute the virtual machine placement solution so that the configuration can survive *k* host server failures. The overall problem is defined as a combinatorial optimization problem and a greedy algorithm for determining the placement solution is provided with the aim of minimizing the number of required hosting servers. Their method performs better than the conventional *N+M* redundant configuration in terms of the number of hosting servers required.

Shi et al. [SBB2013] formulate the problem of virtual machine placement as an Integer Linear Programming (ILP) problem and provide a twofold solution. First, they use solvers to obtain optimal results. Second, since the scalability of this approach is limited, they also provide a modified version of the first fit decreasing heuristic to generate sub-optimal results. In particular, they classify the requests for virtual machine placements into different categories and satisfy the following constraints using the first fit

decreasing heuristic, in the form of a multidimensional vector packing problem: (i) the *full deployment* constraint that ensures either all the virtual machines requested by the user are allocated or none; (ii) the *anti-colocation* constraint requiring all the virtual machines to be placed on different physical hosts; and (iii) the *security* constraint requiring a physical host only be assigned virtual machines from the same user request and not be assigned any virtual machines from other requests.

The three aforementioned resource allocation techniques consider only fault tolerance and security constraints. The solution by Jayasinghe et al. [JPE2011] that also take into account various performance attributes while performing initial allocation of virtual machines. In particular, they propose a structural constraints-aware virtual machine placement approach to improve the performance and availability of applications deployed in the data centers. They integrate the structural information of users applications within the algorithm for initial placement of virtual machines by means of three constraints: (i) *demand constraint*, that defines the lower bound of resource allocations that each virtual machine requires from the service to meet its SLA; (ii) *availability constraint*, that improves the overall availability of given applications using a combination of anti-collocation/collocation constraints; and (iii) *communication constraint*, that represents the communication requirement between two virtual machines. The objective of the proposed algorithm is to minimize the communication cost while satisfying both the demand and availability constraints. Their solution uses the divide-and-conquer technique which involves the following steps: (1) the group of virtual machines requested by the user is divided into a set of smaller virtual machine groups and the upper bound of the virtual machine group size is determined by the average capacity of a server rack; (2) a suitable server rack is identified for each virtual machine group such that the mapping minimizes the total communication cost and guarantees the satisfaction of availability constraints; (3) a physical host satisfying the demand constraint is identified for each virtual machine.

# 5 Runtime adaption of virtual machine allocation in data center environments

Data centers are highly dynamic in terms of task activation, bandwidth availability, component failures and recovery. This implies that static deployment strategies for virtual machines that perform only initial allocation (such as the $p: V \rightarrow H$ function) may not provide satisfactory results at runtime and application's dependability and performance requirements may not be satisfied all along their lives. A naïve approach is to re-compute the allocation from *scratch* each time system changes affect an application. However, since this method may not scale well during runtime [JP2013], a number of solutions have been proposed in the literature to adapt the current allocation of applications using fewer actions. In this Section, we first discuss the solution that balances application's performance and availability goals at runtime (Section 5.1) and then present other state-of-art adaptive resource management schemes (Section 5.2).

## 5.1 Runtime adaption to balance availability and performance

A heuristics-based approach that minimizes the performance and availability degradation of applications due to various system changes has been presented in [JP2013]. To dynamically manage the adaptation of the allocation of applications' virtual machines on the hosts of a data center a dedicated monitoring process (called *online controller*) is introduced in the data center itself. The online controller uses the monitoring information (e.g., application workload, server's failure behavior, processor and bandwidth usage) to create a comprehensive view of the working status of the whole system. When events may violate the applications' availability or performance goals, the online controller re-deploys the applications so as to maintain the dependability and performance characteristics of the data center. For

example, when the availability of an application (i.e., the probability that the application is working correctly and is not affected by hardware failures or security threats) is below the desired value, the current allocation is adapted by deploying new application task replicas as a response to server failures and/or by migrating individual tasks on (other working hosts) across different deployment levels in the system. The initial allocation algorithm (see Section 4) is executed only if the output generated by the online controller is unfeasible for the users' applications.

The online controller is based on the notion of deployment levels discussed in Section 2.4 and on the following remarks. First, the availability of the application increases and the performance (response time) decreases as the number of replicas increase. Second, the availability of the application increases as the deployment level in which its replicas are placed increase. On the contrary, response time decreases as the deployment level increases. Using this analysis, the online controller changes the current allocation of a given application and redeploys it by applying the following allocation actions.

- $Launch(v, h)$: The online controller may have to create new application replicas when a system failure happens. To achieve this, it uses action $Launch(v, h)$ to create a new virtual machine $v$ on a physical host $h$, in which deploys a new replica of the failed application.
- $Migrate(v, h_i, h_j)$: The online controller may have to locate a subset of application's replicas on different physical hosts as a response to performance or availability degradation. For example, to respond to network congestion in cluster $C_1$, the online controller may want to move replicas initially allocated in $C_1$ to another cluster $C_2$. Action $Migrate(v, h_i, h_j)$ specifies that the virtual machine $v$ deployed on host $h_i$ must be moved to host $h_j$.
- $Delete(v, h)$: Due to performance overhead, the online controller may need to reduce the number of replicas of the application, even though dependability might be reduced. Action $Delete(v, h)$ removes virtual machine $v$, hosting application's replica, from host $h$.

On the bases of the current allocation, system status, application tasks and the sets specifying allocation constraints as input, the online controller generates the sequence of allocation actions that brings the system to a new allocation state in which the constraints are again satisfied. The online controller is invoked when a failure or performance degradation is identified for an application. It is worth noting that it is therefore useful for maintaining the availability and performance goals for long-running applications which are more exposed to possible component failures and overloads, while short-running applications can practically be managed by the initial deployment alone.

The algorithm for online adaptation of the applications' allocation consists of two main conditions, one concerning availability violation due to system failures and the other concerning performance degradation. If the real availability of an application is less than the minimum desired value, the online controller first identifies the task replica failures and tentatively launches new replicas at the same deployment level. This *Launch* action is performed only until the current replication level is same as that of the original level and as long as performance goals are not violated. If the addition of application's replicas does not satisfy the user's requirements, the online controller tries to move task replicas to a higher deployment level using the *Migrate* action. We note that the availability increases with increasing deployment levels. This action allows the online controller to generate the new allocation solution without increasing the resource consumption costs. If performance is degraded by moving tasks to higher deployment levels, additional replicas must be created to improve the availability. To create new replicas, the online controller starts from higher deployment levels and moves gradually to lower levels, creating the replicas at the level where availability and performance goals are fulfilled. These actions are realized using *Migrate* and *Launch* actions.
When real performance is less than desired minimum value, virtual machines are deleted instead of launching new replicas, and migration takes place to lower deployment levels instead of moving higher in

the hierarchy. We note that the response time of the application improves as the number of replicas and the deployment level decreases.

## 5.2 Other schemes for runtime virtual machines allocation adaption

In the literature other approaches for dynamic adaptation of virtual machines' allocation in data centers have been studied, even though often they focus only on some of the constraints discussed above. The placement of application replicas to achieve dependability becomes especially challenging when they consist of communicating components (e.g., multi-tier web applications). Recent works on performance optimization of such applications (e.g., [CAA2007, JJH2010]) address the performance impact of resource allocation, but does not combine performance modeling with availability requirements and dynamic regeneration of failed components.

The trade-off between availability and performance is considered in the literature on dependability since increasing availability (by using more redundancy) typically increases response time. In fact, the well-known Brewer's theorem states that consistency, availability, and partition tolerance are the three commonly desired properties by a distributed system, but it is impossible to achieve all three [GL2002]. Examples of work that explicitly address this issue include [SKL1989, KMT2009, JP2013, JJH2010]. Among these solutions, [SKL1989] considers the problem of when to invoke a (human) repair process to optimize various metrics of cost and availability defined on the system. The optimal policies that specify when the repair should be invoked (as a function of system state) are computed off-line via Markov decision process models of the system. Similarly, Jung et. al [JJH2010] study how virtualization can be used to provide enhanced solutions to the classic problem of ensuring high availability while maintaining performance of multi-tier web services. Software components are restored whenever failures occur and component placement is managed using information about application control flow and performance predictions.

Addis et al. [AAP2010] devise a resource allocation policy for virtualized Cloud computing environments aiming at identifying performance and energy trade-off, with a priori availability guarantees for the users. They model the problem as a mixed integer non-linear programming problem and propose a heuristic solution based on non-linear programming and local-search techniques. In particular, the availability requirements are introduced as constraints, and the objective is to determine a resource allocation that allows improving the total profit (the difference between the revenues from SLA contracts and the total costs). Their solution defines the following four actions to take into account availability constraints: (i) increase/decrease the working frequency for each server according to the application loads (a frequency change compatible with the new application loads does not affect availability); (ii) switch a server to low power sleep state and allocate all applications of that server on the other available servers (some applications from the overloaded servers can be duplicated and allocated on another active server to satisfy availability constraints); (iii) reallocation of class-tier applications on a server with sufficient availability assurance to satisfy the performance constraints; (iv) servers exchange is used to move all applications from a server which should be switched to low the power sleep state to a different active server if the availability of the new server guarantees the availability requests of all the moved applications.

Zheng et al. [ZLX2013] consider the lack of maintenance to be the root cause for downtime events in a Cloud computing data center. To address this issue, they first present a heuristics for resource provisioning under a given maintenance schedule and, then, build on the heuristics to solve the joint resource provisioning and maintenance scheduling problem. Yang et al. [YXT2011] propose an algorithm that uses Markov chain models to schedule tasks so that they get the best value of utility. A job being

executed on the Cloud possesses the following factors: deadline, data, and reward factor for completing it on time. The reward is assigned to each task depending on the time it takes to complete the job with respect to its deadline. The earlier it completes, the higher is the reward. If a task fails during execution, then the time required for the task to recover is also added to the total time it takes to complete. The proposed algorithm includes reliability while calculating the value of the reward for each task. The impact of the amount of time which is spent in recovery from a failure and the useless waiting time in the task queue are added in this model. The proposed rules are: execute a job as soon as possible when the resources are available, and pausing a job in the queue till a resource is available or assigning a new task to free resources. This approach has been tested against well-known task scheduling algorithms: if there are not system failures results have quality similar to other conventional scheduling techniques, while in the case of system failures the proposed algorithm produces better efficiency and stability.

## 6 Conclusions

This chapter has discussed the state-of-art resource management schemes that are designed to improve dependability and performance of applications deployed in data centers based on Cloud computing environments. First, we briefly discussed the failure characteristics of data center components and presented an approach to evaluate the impact of system failures on users' applications. We have formalized the applications placement problem and formulated various placement constraints, involving fault tolerance, security and performance conditions of both users and data center owners. Then, we discussed virtual machine placement algorithms that satisfy the placement constraints and reduce the management costs for the data center owner. In particular, the approaches to virtual machines placement discussed in this chapter apply to two different operating phases: the initial allocation of applications at their activation, and the runtime adaption of their placement to deal with dependability threats and performance changes. These strategies ensure dependable data center based on Cloud computing environment for users' applications.

## References


[AAP2010] B. Addis, D. Ardagna, B. Panicucci, and L. Zhang, "Autonomic management of cloud service centers with availability guarantees," in Proc. of 3rd International Conference on Cloud Computing, Miami, FL, USA, Jul 2010, pp. 220-227

[AJJ2013] M. Albanese, S. Jajodia, R. Jhawar, and V. Piuri, "Reliable Mission Deployment in Vulnerable Distributed Systems," in Proc. of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, Budapest, Hungary, June 24-27, 2013, pp. 1-8

[AJJ2013a] M. Albanese, S. Jajodia, R. Jhawar, and V. Piuri, "Securing Mission-Centric Operations in the Cloud," in *Secure Cloud Computing*, S. Jajodia, K. Kant, P. Samarati, V. Swarup, C. Wang (eds.), Springer, 2013

[AJP2013] C. A. Ardagna, R. Jhawar, and V. Piuri, "Dependability Certification of Services: A Model-Based Approach," in Computing, Springer, October, 2013, pp. 1-28

[BBK2002] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "On the propagation of faults and their detection in a hardware implementation of the Advanced Encryption Standard", in Proc. of the 2002 IEEE International Conference on Application-Specific Systems, Architectures and Processors, San Jose, CA, USA, July 2002, pp. 303-312

[BBB2011] E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in Proc. of 31st

International Conference on Distributed Computing Systems, Minneapolis, USA, Jun 2011, pp. 700-709

[BCD2009] C. Blundo, S. Cimato, S. De Capitani di Vimercati, A. De Santis, S. Foresti, S. Paraboschi, P. Samarati, "Efficient Key Management for Enforcing Access Control in Outsourced Scenarios," in Proc. of the 24th IFIP TC-11 International Information Security Conference (SEC 2009), Cyprus, Greece, May 2009

[CAA2007] I. Cunha, J. Almeida, V. Almeida, and M. Santos, "Self-adaptive capacity management for multi-tier virtualized environments," in Proc. of 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany, May 2007, pp. 129-138

[DDF2005] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, P. Samarati, "Key Management for Multiuser Encrypted Databases," in Proc. of the International Workshop on Storage Security and Survivability, Fairfax, Virginia, USA, November 2005

[DFJ2008] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, "Controlled Information Sharing in Collaborative Distributed Query Processing," in Proc. of the 28th International Conference on Distributed Computing Systems (ICDCS 2008), Beijing, China, June 2008

[DFJ2012] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and G. Livraga, "Enforcing Subscription-based Authorization Policies in Cloud Scenarios," in Proc. of the 26th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy, Paris, France, July 11-13, 2012

[DFJ2013] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Enforcing Dynamic Write Privileges in Data Outsourcing," in Computers & Security, 2013

[DFS2012] S. De Capitani di Vimercati, S. Foresti, P. Samarati, "Managing and Accessing Data in the Cloud: Privacy Risks and Approaches," in Proc. of the 7th International Conference on Risks and Security of Internet and Systems (CRiSIS 2012), Cork, Ireland, October 2012

[DLP2012] S. De Capitani di Vimercati, G. Livraga, V. Piuri, F. Scotti, "Privacy and Security in Environmental Monitoring Systems," in Proc. of the 1st IEEE-AESS Conference in Europe about Space and Satellite Communications (ESTEL 2012), Rome, Italy, October 2012

[DLP2013] S. De Capitani di Vimercati, A. Genovese, G. Livraga, V. Piuri, F. Scotti, "Privacy and Security in Environmental Monitoring Systems: Issues and Solutions", in Computer and Information Security Handbook, 2nd Edition, J. Vacca (ed.), Morgan Kaufmann, Boston, 2013, pp. 835-853

[GJN2011] P. Gill, N. Jain, and N. Nagappan, "Understanding Network Failures in Data Centers: Measurement, Analysis and Implications", ACM Computer Communication Review, vol. 41, no. 4, 2011, pp. 350-361

[GL2002] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of Consistent, Available, Partition-tolerant web services," SIGACT News, vol. 33, no. 2, Jun 2002, pp. 51-59

[GY2010] R. Guerraoui and M. Yabandeh, "Independent faults in the cloud," in Proc. of 4th International Workshop on Large Scale Distributed Systems and Middleware, Zurich, Switzerland: ACM, 2010, pp. 12-17

[HB2009] U. Helzle and L. A. Barroso, "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines", 1st ed. Morgan and Claypool Publishers, 2009

[HDL2011] F. Hermenier, S. Demassey, and X. Lorca, "Bin repacking scheduling in Virtualized datacenters", in Proc. of Constraints Programming, Perugia, Italy, 2011

[HLM2011] F. Hermenier, J. Lawall, J.M. Menaud, and G. Muller, "Dynamic Consolidation of Highly Available Web Applications", INRIA, Tech. Rep. RR-7545, 2011

[JJH2010] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu, "Performance and availability aware regeneration for cloud based multitier applications," in Proc. of 2010 IEEE/IFIP

International Conference on Dependable Systems and Networks, Chicago, IL, USA, July 2010, pp. 497-506

[JP2012] R. Jhawar and V. Piuri, “Fault tolerance management in IaaS Clouds,” in Proc. of the 1st IEEE-AESS Conference in Europe about Space and Satellite Telecommunications, Rome, Italy, Oct 2012, pp. 1-6

[JP2013] R. Jhawar and V. Piuri, “Adaptive resource management for balancing availability and performance in Cloud computing,” in Proc. of the 10th International Conference on Security and Cryptography, Reykjavik, Iceland, Jul 2013, pp. 254-264

[JP2013a] R. Jhawar and V. Piuri, "Fault Tolerance and Resilience in Cloud Computing Environments," in *Computer and Information Security Handbook, 2nd Edition*, J. Vacca (ed.), Morgan Kaufmann, 2013, pp. 125-141

[JPE2011] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, “Improving performance and availability of services hosted on IaaS Clouds with structural constraint-aware virtual machine placement,” in Proc. of IEEE International Conference on Services Computing, Washington, DC, USA, Jul 2011, pp. 72-79

[JPS2012] R. Jhawar, V. Piuri, and P. Samarati, “Supporting security requirements for resource management in cloud computing,” in Proc. of the 15th IEEE International Conference on Computational Science and Engineering, Paphos, Cyprus, Dec 2012, pp. 170-177

[JPS2013] R. Jhawar, V. Piuri, and M. Santambrogio, “Fault Tolerance Management in Cloud Computing: A System-Level Perspective,” in IEEE Systems Journal 7 (2), June, 2013, pp. 288-297

[KMT2009] S. Kim, F. Machida, and K. Trivedi, “Availability modeling and analysis of virtualized system,” in Proc. of 15th IEEE Pacific Rim International Symposium on Dependable Computing, Shanghai, China, Nov 2009, pp. 365-371

[MFD2011] K. Mills, J. Filliben, and C. Dabrowski, “Comparing VM-Placement Algorithms for on-demand Clouds”, in Proc. of CLOUD’11, Washington, USA, 2011, pp. 91-98

[MKM2010] F. Machida, M. Kawato, and Y. Maeno, “Redundant Virtual Machine Placement for Fault Tolerant Consolidated Server Clusters”, in Proc. of IEEE/IFIP NOMS’10, Osaka, Japan, 2010, pp. 32-39

[SBB2013] L. Shi, B. Butler, D. Botvich, and B. Jennings, “Provisioning of requests for virtual machine sets with placement constraints in IaaS clouds,” in Proc. of IFIP/IEEE International Symposium on Integrated Network Management, Ghent, Belgium, May 2013, pp. 499-505

[SD2010] P. Samarati and S. De Capitani di Vimercati, “Data Protection in Outsourcing Scenarios: Issues and Directions”, in Proc. of the 5th ACM Symposium on Information, Computer and Communications Security, Beijing, China, 2010, pp. 1-14

[SKL1989] K. Shin, C. M. Krishna, and Y. Hang Lee, “Optimal dynamic control of resources in a distributed system,” IEEE Transactions on Software Engineering, vol. 15, no. 10, 1989, pp. 1188-1198

[SKM2008] A. Singh, M. Korupolu, and D. Mohapatra, “Server-storage virtualization: Integration and load balancing in data centers,” in Proc. of SC'08, Austin, TX, USA, Nov 2008, pp. 53:1-53:12

[STT2008] W.E. Smith, K.S. Trivedi, L.A. Tomek, and J. Ackaret, “Availability Analysis of Blade Server Systems”, IBM Systems Journal, vol. 47, no. 4, 2008, pp. 621-640

[VN2010] K. Vishwanath and N. Nagappan, “Characterizing Cloud Computing Hardware Reliability”, in Proc. of SoCC’10, Indianapolis, IN, USA, 2010, pp. 193-204

[YXT2011] B. Yang, X. Xu, F. Tan, and D.-H. Park, “An utility-based job scheduling algorithm for cloud computing considering reliability factor,” in Proc. of International Conference on Cloud and Service Computing, Hong Kong, Dec 2011, pp. 95-102