Tech Report
Datacenter-wide Portfolio Scheduling for Managing Operational and Disaster-Recovery Risks in Virtualized Datacenters Hosting Business-Critical Workloads

Vincent van Beek Giorgos Oikonomou Alexandru Iosup

December 13, 2016

Contents

1 Introduction

Attracted by the dual promise of infrastructure efficiency [43] and widespread uptake [30], large enterprises and governmental organizations are increasingly using public- and/or private-cloud resources to run their large-scale *business-critical workloads*. Although the promises are enticing, hosting business-critical workloads is relatively new, and raises many resource management and scheduling challenges. Particularly challenging is enforcing for such workloads *risk-aware* service-level agreements (SLAs) that express not only strict customer demands for high performance and reliability, but also the aversion of enterprises and governments to the risk of their performance and reliability demands not being met. Traditional datacenters focus on best-effort execution of workloads, which serves well customer demands in traditional datacenters, but may not be suitable for risk-aware SLAs and thus may lead to high financial penalties for cloud operators. In contrast to traditional work in resource management and scheduling in datacenters, in this work we address the research question of *How to manage the risk of not meeting SLAs for datacenters hosting business-critical workloads?*

Business-critical workloads are comprised of diverse applications and services needed in daily organizational life, from common email to custom simulation software [68]. There is strong financial incentive to host business-critical workloads in datacenters: the cloud-based application market, which includes business-critical workload hosting, is growing rapidly at over 30% Compound annual growth rate (CAGR) [30], already exceeds $100 billion/year world-wide [19], and will likely contribute as much to the GDP of the European Union, by 2020 [30].

Managing business-critical workloads means adapting to new workload characteristics. According to Shen et al. [68], business-critical workloads are significantly different from the workloads previously addressed by the datacenter research community, and in particular from parallel production workloads [33], grid workloads [44], desktop grid workloads [51], and the analytics workloads common at large web-scale companies (e.g., Facebook [17], Google [61], and Taobao [63]). One major difference is that business-critical workloads are typically expressed as *streams of requests for (non-transparent) VMs*, instead of explicit queries or requests to execute a specific application, mainly because for datacenter customers details about their software are too sensitive to reveal.

The operational model emerging in datacenters *hosting* business-critical workloads focuses on *long-running* virtual machines (VMs)—instead of submitting traditional jobs or user-requests to the cloud operator, organizations lease VMs from clouds offering Infrastructure-as-a-Service, such as Amazon AWS, and run their workloads on these VMs; in turn, VMs run on powerful physical machines residing in large-scale datacenters. This means a shift from the traditional focus on job runtime, to a renewed focus on reliability and availability of resources (a form of resource utilization). We discuss this operational model in Section 3.

There are many types of risks in hosting business-critical workloads in datacenters: *operational risks* of customer requests not being responded to in time, *reliability risks* of requests failing, etc. These risks affect many *datacenter stakeholders*: DevOps engineers, reliability engineers, infrastructure engineers, business and legal representatives, capacity planners and business managers, etc.

Datacenters already span a large number of powerful computers, high-performance networks, and large-capacity storage systems [9]. Meeting complex SLAs in such datacenters requires monitoring and reasoning about diverse operational metrics, to an extent that exceeds since at least the early 2010s the capabilities of unassisted engineers [8]. Although many automated scheduling approaches exist, over the past two decades numerous studies [32, 50, 72] have shown that datacenter schedulers are brittle—different schedulers will ex-

hibit different periods of poor performance that in turn lead to unexpected performance issues, unnecessary resource overload, and even to (cascading) failures. Instead of trying to develop scheduling policies capable to address all possible workloads, which is error-prone and ephemeral [74], it may be possible to dynamically select active scheduling policies from a pool of available schedulers.

Selecting the active scheduler, while meeting the SLAs of business-critical workloads, in particular addressing risk, is the focus of this work. Current approaches to select schedulers dynamically either combine the use of simple scheduling policies with human expertise [50], or rely on a dynamic meta-scheduler, such as a portfolio scheduler [23, 66, 74]. Addressing the research question, in this work we extend the state-of-the-art in portfolio scheduling for datacenters with explicit risk management. Our main contribution is five-fold:

1. Significantly extending the state-of-the-art in portfolio scheduling with two types of risk (Section 4).
2. Three risk-aware utility functions which are used by the portfolio scheduler to activate one scheduler over another (Section 4.3).
3. Two heuristics-based risk-aware scheduling policies to address the new risk-aware utility functions (Section 4.4).
4. A method for selecting schedulers based on a novel CPU-contention predictor, which we use to estimate the risk of SLA-violations (Section 5).
5. New insight into the operation of portfolio scheduling in datacenters, through trace-based simulation using traces collected from real datacenters (Sections 5.5 and 6).

2

Background

In this chapter we provide background information about the workload type we investigate on this work, the business-critical workloads. We refer to the state of the art in risk management in clouds and in the two main topics in this work, the quantification of resource contention and portfolio scheduling as a concept in computer science.

2.1. Business-Critical Workloads

Business-critical workloads [68] are comprised of applications that have to be available for the business to not suffer significant loss. The applications span a broad range of user-facing and back-end services, often including email, database, CRM and collaborative, and management services. When these services experience downtime or even just low performance, they often lead to loss of revenue, of productivity, etc., and may incur financial loss, legal action, and even customer departure.

Business-critical workloads are different conceptually from other workloads common in datacenters, for example, from mission-critical workloads, which are for instance software that operate production lines, and from incidental workloads, which represent jobs of relatively short duration that are sent to the datacenter infrequently.

In this work, we use business-critical workloads as our testing workloads because of their importance in enterprise reliability and growth. Another reason to investigate this type of workload is the diversity of the constituent application types providing a real challenge to schedule them under multi-objective optimization goals.

A representative trace of business-critical workloads can be in found in Grid Workloads Archive [44] (trace GW-T-12).

2.1.1. Characterization of Business-Critical workloads

Business-Critical workloads differ from other workload types not only on conceptual level as mentioned, but also on their characteristics on resource-utilization level. Technical report [67] presents a comprehensive statistical characterization of BCW trace found in Grid Workloads Archive [44].

The analysis presents results by investigating basic statistics, correlations between different resources and time-patterns of resource utilizations. The applications of BCW have strong correlations for the provisioned resources of CPU and memory but weak correlations for the used resources of the same resource types. In addition, provisioned and used resources are weakly correlated for all the investigated resource types. Finally, network and storage utilizations are regularly on low levels but show extreme bursts during time periods under daily patterns.

The main findings of the business-critical workload characterization and their differences with other workload types are:

- More than 60% of the VMs use less than 4 cores and 8 GB of memory which is a relative to parallel workload jobs.
- Resource usage is low, under 10% of the provisioned resources, compared to the (high) utilization of scientific workload jobs. This gives opportunities for consolidation, subject to SLAs and risk tolerance.

- Peak workloads can be 10–10,000 times higher than mean workloads, depending on resource type. This is a challenge for guaranteeing SLAs as scheduling should consider these peak times and the severity of risky time periods that peak times induce.

2.1.2. SLAs

The main reason of companies moving their infrastructures from private to public clouds is the maintenance costs of a private datacenter and the upgrade costs of the continuously changing hardware. Using public clouds though, may have some implications for the job performances. The same resources are used by several jobs and schedulers try to evenly share the resources in terms of time. In addition, the resource management of public clouds are not managed by the cloud customers, thus customers cannot control the infrastructure operations. For these reasons, cloud providers offer agreements to customers assuring job performances.

A Service-Level-Agreement (SLA) is a contract between cloud providers and customers describing the requirements of customer jobs. These requirements refer to performance levels that job execution should meet according to customers. The required performance levels mainly refer to the availability of the public resources for usage as well as job execution times. The customers periodically receive reports by the cloud operators presenting the performance of customer jobs and possible SLA-violations of them. In case providers do not meet the agreed SLAs, there are financial penalties paid to customers.

For Businness-Critical workloads, SLAs extend to consolidation limits of virtual machines on physical machines as runtimes of mission-critical jobs cannot afford high latencies. In addition, there are SLAs requiring full recoverability of customer workloads in case of datacenter failures (e.g., power outage).

2.2. Risk Management in Clouds

Risk management in clouds includes an increasing body of work on SLAs between cloud customers and operators. Following more than a decade of evolution in the context of grids [22, 79], the state-of-the-art focuses on defining various system properties and SLA types [78], on negotiating and brokering SLAs [10, 25, 49], on monitoring for [20] and assessing SLA-violations [24, 54], and on selecting clouds who minimize them [35] and other aspects of SLA-lifecycle management [55]. Our work extends risk definition and management with two new types of SLA elements—operational risk of performance degradation due to consolidation and the reliability risk of datacenter disaster.

2.3. State-of-the-Art in Quantifying Resource Contention due to Virtualization in Datacenters

Resource contention is a major issue in clouds as resource sharing is the common technique to co-host cloud services. Research on the filed of predicting resource contention focuses on estimating the performance degradation of cloud services by co-hosting particular types of workloads and on using this knowledge to scheduling workloads.

The current research approaches the resource contention problem by estimating the contention of a specific resource type at a time. There are many components in a machine that contribute to the resource utilization of every resource type, thus addressing contention even of one resource type is a challenge. In [57], [13], authors estimate the performance degradation of applications due to memory contention derived from the related components as memory controller, memory bus and cache. In [48], predictions of execution times of storage commands are used towards a fair scheduling in terms of storage contention and in [82], a host-level scheduler considering network contention achieves performance improvement with better VM placement. In [62], the CPU utilization is predicted resulting to scheduling the applications concerning CPU contention levels.

Another approach of quantifying resource contention is by measuring the response times of applications running in co-hosted environments. In [28], the authors predict the contention level that an application experiences by comparing the response times of the application when is co-hosted with other applications and when runs alone in the system. In [81], an automated method is used to identify the resource type of the application which suffers the most from contention and estimates how the contention affects the application response-times.

Our work on quantifying resource contention focuses on CPU contention as one of the major performance factors in business-critical workloads. Our work differs from the state-of-the-art in quantifying resource contention as it estimates the contention levels of VM by a straightforward prediction of a contention

metric and not by inferring it from deviations of performance levels. In addition, we do not investigate external application metrics such as the response time, but we focus on metrics of internal VM states such as resource-level metrics.

The aforementioned features of our predictor estimating resource-level metrics declare also the reason why we investigate a new CPU contention predictor. Business-critical workloads, as mentioned in Section 1, comprise opaque VMs and it is not possible to measure the performance levels of the applications running in the VMs. Furthermore, the levels of response times of applications in business-critical workloads are not relevant, therefore a predictor estimating such metric cannot be used.

Our method on selecting the best CPU-contention predictor for business-critical workloads is described in Chapter 5.

2.4. State-of-the-Art in Portfolio Scheduling

Portfolio scheduling originates from the field of finance and has been added several stages throughout the years towards its creation. The concept of *computational portfolio design* which adapts portfolio scheduling to work on hard computational problems is introduced in [42]. The application of the concept to scheduling in datacenters is presented in [23] and further studied in [73], [74] and [65].

In contrast to works in [23], [65] and [73], our work adds the focus on risk management which is a new problem greatly extending the scope of portfolio scheduling. This much larger focus leads to the design of new families of risk-related utility functions and of new scheduling policies (*risk-aware portfolio scheduling*). In contrast to [74], we provide much more comprehensive work in experimental analysis by investigating when transitions between selected scheduling policies occur and how the portfolio selections affect the utility functions over time.

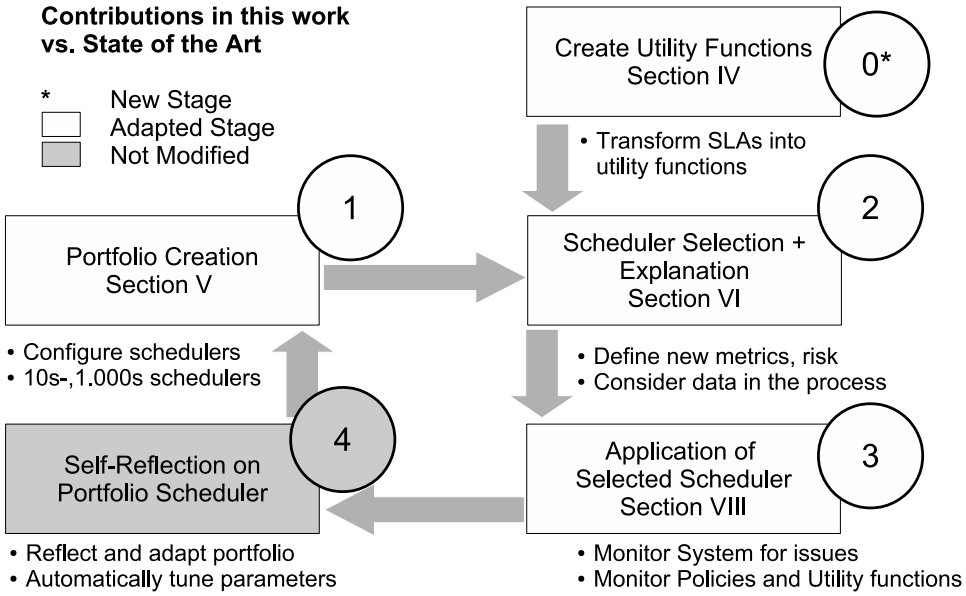


Figure 2.1: Portfolio Scheduling Stages.

Stages in portfolio scheduling Figure 2.1 presents the stages of portfolio scheduler creation. Stage 1 is designed to provide a set (portfolio) of meaningful scheduling policies that will be considered by portfolio scheduler and Stage 2 focuses on the process of scheduler selecting the policies. Stage 3 applies the policy selected in the previous stage and integrate the new system status to the current datacenter infrastructure. In addition, monitoring of portfolio scheduler's operations take place in Stage 3. The Stage 4, the Self-Reflection stage, addresses the concept of eliminating scheduling policies from the selection process to make faster decisions with the same quality. In this work we added stage 0, in which we perform the work of creating utility functions based on datacenter operator requirements.

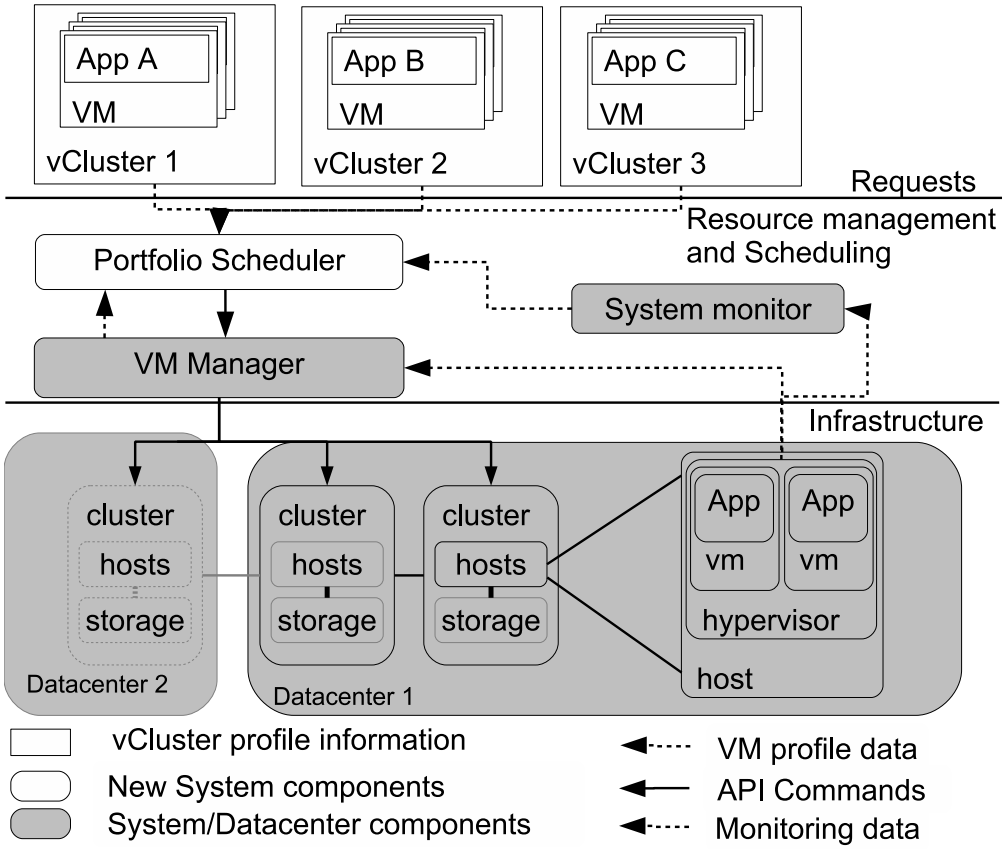


Figure 3.1: Topology and operation of virtualized datacenters hosting business-critical workloads.

3

System Model for Virtualized Datacenters Hosting Business-Critical Workloads

We use in this work the system model for virtualized datacenters hosting business-critical workloads introduced by van Beek et al. [74]. We describe the datacenter and the workload models, in turn.

3.1. Virtualized Datacenters Architecture

The architecture consists of a collection of inter-connected *datacenters* that use of the shelf virtualization technology, such as hypervisors on each node and the system-wide VM Manager, to host the VMs of business-critical workloads. Each datacenter consists of one or several *clusters* of *physical machines* that can host VMs, and a cluster-wide parallel storage system. A high-speed network operates inside each datacenter; we use in

this work an all-to-all Infiniband model, but a Clos topology [69] could also be used in our model without changes.

The typical operation of this architecture is depicted in Figure 3.1. In the top row (the section labeled "Requests" in Figure 3.1), workload arrives in the system as requests for hosting, one or several VMs, grouped as *vClusters*. Datacenter customers define their own vClusters, including detailed resource specifications (e.g., memory in GB, amount of CPU cores, storage space in TB), affinities between VMs (e.g., that one VM is not allowed to be in the same datacenter as another). In the middle row, core resource management and scheduling components respond to requests. The *VM manager* orchestrates the placement of VMs on the physical machines. The *System monitor* collects the status of each customer request and system component, at runtime. A non-traditional component is *the system meta-scheduler—the new Portfolio Scheduler component*. The main intuition for using a portfolio scheduler is that, instead of using the elusive single scheduler that can always work well in the changing datacenter, a portfolio scheduler [23, 74] uses a set of schedulers from which it dynamically selects the scheduler which promises to deliver the best performance until the next selection. Unlike previous dynamic schedulers [72], portfolio scheduling relies on a rigorous configuration-selection-application-reflection cycle. In the bottom row we show the physical infrastructure that hosts the vClusters (collections of VMs).

3.2. Business-Critical Workloads

We focus in this work on large-scale business-critical workloads, which have the following reported characteristics [68]:

1. Requests arrive in the system not as specifications of jobs, but as specifications of VMs that will run jobs (the vClusters introduced in Section 3.1).
2. Long run-times—VMs often run for months or years. In contrast, scientific workloads consist mainly out of shorter-lived jobs that run for only hours or days.
3. Consequence of point 2, reducing VM-runtime is not a performance goal here, unlike traditional workloads.
4. Most VMs are small relative to parallel workload jobs, that is, over 60% of the VMs use less than 4 cores and 8 GB of memory.
5. Most VMs have very low resource utilization compared to scientific workload jobs, that is, over 50% of the VMs have an average utilization of under 10%. This gives opportunities for consolidation, subject to SLAs and risk tolerance (see following).

Industry partners have indicated to us that business-critical workloads have additional important properties:

1. VMs of business-critical workloads can be *consolidated* on the same physical resource, which contrasts with many parallel and grid jobs. Datacenter operators have to limit the consolidation rate such that VMs can perform according to SLAs. For example, rarely more than 10 VMs can be consolidated on the same physical machine, and for reliability purposes the VMs of a vCluster may request to run in different datacenters.
2. The *operational risk tolerance* is low. A small fraction of under-performing VMs quickly leads to an escalated request for engineering time, until it is resolved, which is unscalable and costly.
3. The risk tolerance for VM failure is high for single-VM failures, but the *risk tolerance for disaster-recovery* for (near-)full-datacenter outages is very low. An hour of downtime cascading across many VMs will incur high financial penalties and damage datacenter reputation; instead, the cloud operator should recover from disasters quickly, by having other datacenters absorb the workload of the failed datacenter.

4

Portfolio Scheduling for Managing Operational and Disaster-Recovery Risks

To manage (reduce) operational and disaster-recover risks in datacenters, in this section we design a *risk-aware* portfolio scheduling to select dynamically the schedulers that guide the placement of VMs in a (multi-)datacenter environment.

4.1. Requirements When Hosting Business-Critical Workloads

(R1) Mitigate Operational Risk: The placement of VMs must take into account SLAs regarding service perfomance. The *Operational Risk* is the risk of not achieving service-perfomance SLAs. In this work, we use the VM CPU-performance as the service-performance indicator.

(R2) Guarantee Disaster Recovery: The placement of VMs must ensure that workload from a failed datacenter can be absorbed by remaining datacenters. The *Disaster-Recovery Risk* is the risk of not achieving this type of SLA.

(R3) Balance Multiple Optimization Objectives: the scheduler must be able to balance multiple SLAs with cloud operator's cost-related goals. In this work, we consider the cost-related goal of high utilization of resources through consolidation, balanced with the SLA objectives identified in R1 and R2.

(R4) Adapt Dynamically to New Workloads: The scheduler must adapt dynamically to new or changing workloads, which are common in datacenters that host business-critical workloads. The scheduler should additionally have the functionality to handle changing requirements of the datacenter operators (e.g., new optimization goals).

(R5) Explain decisions to human operators: The scheduler must *explain* scheduling decisions to datacenter operators, so that they can understand *why* specific policies were selected.

4.2. Overview of the Portfolio Scheduling Architecture

We propose in this section an architecture for risk-aware portfolio scheduling. Portfolio scheduling is the cyclic, multi-stage process of configuration, selection, application, and reflection that promises the dynamic use of the best scheduling policy for each decision. We extend the state-of-the-art in portfolio scheduling for datacenters [23, 74] to an architecture for *risk-aware* portfolio scheduling, where each stage of the cyclic process takes into account notions of risk. Specifically, our new architecture uses new risk-aware utility functions, new risk-aware scheduling policies, and a new method to simulate the datacenter that considers CPU-contention accurately.

Our architecture for risk-aware portfolio scheduling, depicted in Figure 4.1, receives as input requests for vClusters (see Section 3.1), which are queued (box labeled "Workload" at the top of the figure). We describe in turn the other components in the figure that implement the multi-stage cycle of the portfolio scheduler.

Utility Functions represent the optimization requirements given by datacenter stakeholders (see Section 1). New in this work, we allow for *risk-aware* utility functions, such as operational risk (R1) and disaster-recovery

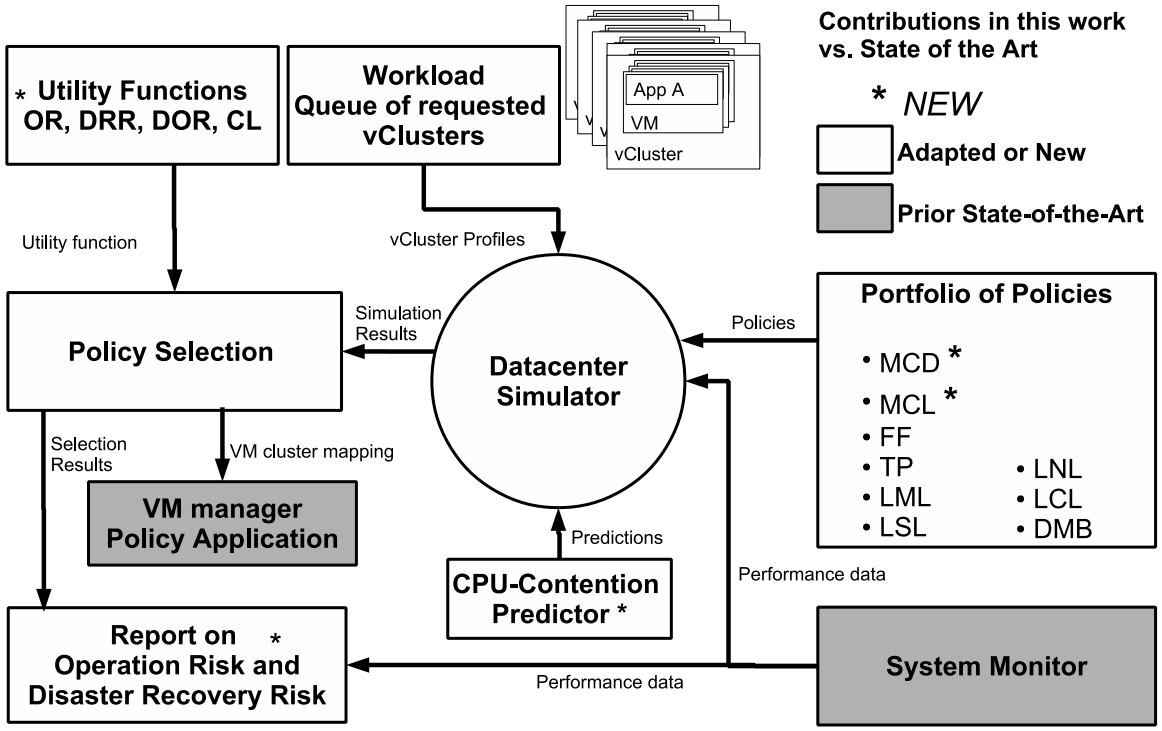


Figure 4.1: Architecture for *risk-aware* portfolio scheduling.

risk (R2). We introduce in Section 4.3 three risk-aware utility functions ("OR", "DRR", and "DOR" in Figure 4.1) and discuss how they capture requirements (R1) and (R2).

Similarly to the creation of utility functions, new in this work we allow the **Portfolio of Policies** to consider not only traditional but also *risk-aware* scheduling policies. We introduce in Section 4.4 two new scheduling policies that respond to requirement (R3).

Addressing (R4), the **Datacenter Simulator**, the **CPU-Contention Predictor**, and the **Policy Selection** components all focus on selecting policies from the portfolio. The Datacenter Simulator uses simulation to estimate, independently for each policy in the portfolio, the performance and risk score of each policy according to the utility function in use. Each simulation receives as input live performance data from the datacenter **Performance Monitor** component. A key issue is the ability of the simulator to cope with virtualized environments, in particular to estimate performance degradation due to consolidation; we address this issue with a CPU-Contention predictor, in Section 5. (We have investigated methods that successfully limit the time given to the simulator without significant performance degradation for the portfolio scheduler in our previous work [23].) Finally, Policy Selection includes the creation of metrics and mechanisms to select a policy, based on utility functions and on the output of the simulation process. Additionally, this component also provides data used later to explain the decisions (R5).

The **VM manager [for] Policy Application** applies the policy selected in the previous stage, and integrates our scheduler with existing datacenter infrastructure management tooling. The **Report (on Operational and Disaster Recovery Risk)** component is used by datacenter stakeholders, in particular engineers, for observing the operation of the portfolio scheduler, understand it (R5), and perhaps tune it. In Section 6, we use real-world workload traces to show the effects of using Reporting over time.

4.3. Risk-aware Utility Functions

To address requirements R1, R2, and R3 (Section 4.1) we design three new risk-aware utility functions. Our Operational Risk utility function addresses the risk of resource contention due to resource oversubscription. Our Disaster-Recovery Risk utility function expresses the risk of not being able to recover from a complete datacenter failure. The SLAs with regard to availability dictate that the non-failed datacenters should be able to absorbe the workloads from the failed datacenter. Finally we design DOR, a utility function that combines

the Operational Risk and Disaster Recovery Risk, responding to the requirement for managing multiple risks at the same time. We describe each utility function in detail in the following sub-sections.

4.3.1. Operational Risk (OR)

Responding to R1, we define OR as the risk that VMs will not receive the requested amount of resources. In this work, we consider explicitly *CPU* resources; more resources can also be considered [41]. In Equation 4.1, we define OR (r_o) as the proportion of resource demands (D_t) that can be met by the used resources (U_t), over a period time (T). The values give an intuition of the severity of performance degradation that VMs may experience, with lower values interpreted as better.

$$r_o = \frac{1}{T}\int_T \frac{D_t - U_t}{D_t}\,dt \in (0,1] \tag{4.1}$$

4.3.2. Disaster Recovery Risk (DRR)

Requirement R2 describes the need for guaranteeing the ability to absorb full-datacenter failures when multiple (n) datacenters are present. DRR presents the risk for not meeting this requirement, by expressing to what extend the remaining datacenters can absorb the workload of the failed datacenter. In Equation 4.2, we define DRR of a datacenter in a multi-datacenter infrastructure as r_{d_i}, where the workload in datacenter i is W_i, and $E_i^{\complement}$ is the complement empty-space (that is, the empty space in all datacenters excluding datacenter i). In Equation 4.3, we use the geometric mean to combine the *normalized* score of individual datacenters into a single value of *DRR for an entire multi-datacenter system* (r_d).

$$r_{d_i} = \begin{cases} \frac{W_i - E_i^{\complement}}{E_i^{\complement}} & , \quad W_i \le E_i^{\complement} \\ \frac{W_i - E_i^{\complement}}{W_i} & , \quad W_i > E_i^{\complement} \end{cases} \in [-1,1] \qquad i \in [1,\ldots,n] \tag{4.2}$$

$$r_d = \sqrt[n]{\prod_i \frac{r_{d_i}+1}{2}} \quad \in [0,1] \qquad i \in [1,\ldots,n] \tag{4.3}$$

In this work, we consider *memory* as the most prominent resource when computing W_i and E_i. When hosting business-critical workloads, it is common to guarantee the reservation of memory resources, which means that this resource is not over-provisioned but it needs to be guaranteed on a per VM basis. Thus, memory demands cannot be absorbed, if the empty space is insufficient.

4.3.3. Disaster–Operational Risk (DOR)

Requirement R3 emphasizes the need for complex utility functions, that can be used to construct practical SLAs. Meaning that datacenter operators need to reduce multiple risks and other datacenter optimization goals such as operational cost. In addition, when discussing system status and performance, datacenter stakeholders often need a few or even a single (utility) value that represents the current risk level of the datacenters. For DevOps engineers and site reliability engineers, a single value simplifies an already complex monitoring process, especially useful when immediate action is required to improve risk levels.

Addressing R3, in this work we design the DOR risk, which combines the utility functions for both OR and DRR. In Equation 4.4, we define the combined utility function, r_{od}, as a weighted average of the two risk metrics, where w_o and w_d are the weights, respectively. We analyze the impact of these weights on the risk exhibited by the system, in Section 6.2.

$$r_{od} = \frac{w_o \cdot r_o + w_d \cdot r_d}{w_o + w_d} \in [0,1] \tag{4.4}$$

4.4. Portfolio Creation

In this section, we design two new scheduling policies for placing the VMs of vClusters to physical datacenter clusters. As customary for this level of scheduling, we assume that a cluster-level scheduler (e.g., VMware's DRS, Condor, and Mesos) will take the placement request and actually run the VM on the physical machines of the cluster. Besides these policies, the portfolio scheduler can be equipped with many other policies; as described in Section 5.5.2, we include seven more policies in our experiments.

The two policies we design aim to balance the contention on CPU resources, and to maximize the consolidation of VMs on physical hosts, respectively. Reflecting R3, these policies align with the SLAs (e.g., service

Algorithm 1 Mean Contention Duration (MCD)

Input: V a new Virtual Machine, C all the Clusters;
1: $P = []$ (contention per cluster)
2: **for** $c \in C$ (all clusters) **do**
3: $\epsilon = 0$ (the contention)
4: **for** $v \in c$ (all VMs in cluster c) **do**
5: $p = (\Sigma D_v)/|D_v|$ (mean contention duration for v)
6: $\epsilon = \epsilon + p$
7: **end for**
8: $P[c] = \epsilon/|c|$ (normalize for the number of VMs)
9: **end for**
10: sort P (in increasing order of mean contention)
11: **for** $c \in P$ **do**
12: Break **if** V fits
13: **end for**
14: if the V fits in no cluster escalate to Engineer
15: **return** Cluster mapping for VM placement

performance) and business goals (e.g., minimizing the physical footprint and cost) that datacenter operators face when hosting business-critical workloads.

For simplicity, we present in this section single-VM versions of our policies. We have implemented and used in Section 6 vCluster (multi-VM) versions, which derive trivially from the single-VM versions presented here.

4.4.1. Mean Contention Duration (MCD)

Intuitively, VMs should suffer equally from performance degradation due to co-hosting. This contrasts to the possibility of having most VMs not suffer at all, while a few VMs suffer greatly. VMs that suffer greatly lead to SLA violations, e.g., performance-related, and to financial and other penalties.

Addressing the intuition, we design the MCD heuristic policy listed in Algorithm 1. MCD aims to balance long-term contention, and thus VM suffering, across all clusters. First, MCD computes the mean duration of the period for which VMs suffer of contention, per cluster (lines 2–9). In the simulator, instead of a real-world measurement, contention (line 5) is estimated using the predictor from Section 5. Second, MCD sorts the clusters by mean contention, lowest first (line 10). Finally, MCD places each VM in the cluster with the lowest contention that can fit the VM (line 11).

4.4.2. Maximum Consolidation Load (MCL)

Intuitively, increasing consolidation, that is, the number of VMs sharing the resources of a set of physical machines, leads to higher profit for the datacenter operator, because physical machines that are not occupied by any VM can be selectively powered off [34]. For operators, this is one of the important ways of reducing datacenter cost.

We design the MCL heuristic policy listed in Algorithm 2 to maximize the consolidation of VMs on physical clusters. the sum of resources used by all the running VMs in the cluster. Per cluster, MCL calculates (lines 2–5) the gap over time between requested and actually used resources. Because VMs for business-critical workloads are often over-provisioned, as indicated in Section 3.2, the cumulative gap is often large. Then, MCL sorts the clusters by gap, largest-gap first (line 6). Finally, MCL places a new VM in the cluster with the biggest gap where it fits (lines 7–9).

Algorithm 2 Maximum Consolidation Load (MCL)

Input: V a new Virtual Machine, C all the Clusters;
1: U = [] (unused resources per cluster)
2: **for** $c \in C$ **do**
3: u_v (unused resources for VM v)
4: $U[c] = \Sigma u_v$, $v \in c$ (unused resources in cluster c)
5: **end for**
6: sort U (in decreasing order of unused space)
7: **for** $c \in U$ **do**
8: Break **if** V fits
9: **end for**
10: if the V fits in no cluster escalate to Engineer
11: **return** Cluster mapping for VM placement

5

Scheduler Selection With a novel CPU-Contention Predictor

To address requirement R1, any portfolio scheduler must address a special problem of service performance in virtualized datacenters: when multiple VMs are co-hosted on the same physical machine, any of the VMs may experience transient performance degradation, caused by the contention for resources between multiple VMs.

To address this problem during scheduling, the portfolio scheduler must be able to anticipate contention in the simulations conducted during the "Scheduler Selection" stage. In this section, we design a new component of the simulator that is used to estimate contention. Additionally we propose a method to select policies from the portfolio based on the level of predicted contention.

We focus in this work on CPU contention, which is likely to occur because SLAs business-critical workloads allow for CPU resources to be over-committed (unlike, for example, memory resources, for which SLAs require it to be guaranteed as non-shared). Because a fraction of business-critical workloads do require full access to the CPU (e.g., HPC-like applications) predicting contention is vital to making good scheduling decisions.

In this work we design a CPU-contention predictor that can use data available in simulation. Ours is the first predictor able to address CPU-contention in VMs hosted in clouds.

5.1. Overview

Following the conceptual map of portfolio-scheduler description in Figure 2.1, we set the requirements to address issues of stage 2 on the portfolio scheduler to cope with virtualized environments, and in particular to estimate performance degradation in the system in order to quantify the corresponding risks (see Section 4.3). The requirements we identify are the following:

(R1) Measuring Performance Degradation in Simulation Process: Cloud applications may suffer from performance degradation when the consolidated VMs contend constantly for the same resources. To investigate the risks of SLA violations due to the decrease of VM performance, we need to measure the performance levels. Because of the different VM placements portfolio scheduling investigates, monitored contention data cannot be used since it reflects the contention in a specific setup. Therefore, measuring the potential performance degradations of VMs in different system setups needs a predictor.

(R2) Representative Predictor depicting contention in real environments running business-critical workloads: After investigation, there might be many good predictors estimating CPU contention in clouds. We have to define a screening process to identify the best among them in terms of accuracy and speed, two key elements for the quality of simulator.

5.2. CPU-Contention

There are many types of resouce contention that occur in cloud systems. Cloud applications run together with different workloads resulting in non-synchronized accesses to the shared resources. This leads to cases

where co-hosted applications contending for the common resources and not receiving the demanded resource amounts.

Research in the field of identifying the reasons of resource contention [62],[13] focus on CPU and memory unavailability when the applications demand such resource types. In this work, we investigate the contention in CPU resources as CPU is allowed to be over-committed by typical SLAs unlike, for example, memory resource which is guaranteed to be non-shared. Furthermore, application types of business-critical workloads such as HPC-applications require full access to the CPU to run efficiently.

CPU-Contention Metrics There are a couple of contention metrics engineers use to monitor contention in systems [75]. All of these can be directly retrieved by the Performance Monitor even for short time durations. Such metrics are the *CPU-Ready* (ms and %), the *CPU Contention* (%) and the *CPU co-stop* (ms). There are also other metrics which can be used indirectly for measuring contention in the system by exploiting the relations between the metrics. One example of this is the relation of *CPU-run* and *CPU I/O wait* indicating the time durations a CPU core spends on running and on waiting for I/O calls respectively.

CPU-Ready Metric In this work, we use the CPU-Ready as our contention indicator which measures the time a VM is ready to run but waits to be scheduled to run on the physical CPU [75]. The ready time increases when the scheduler is busy handling many waiting VMs which in turn have different provisions of cpu-cores and diverse resouce demands. The CPU-Ready metric has become the *de facto standard* for engineers to measure contention problems, thus we consider it as good indicator to the problem.

In Figure 5.1 we depict the contention durations of VMs retrieved from a subset of an operational trace of business-critical workloads running in three datacenters. To the best of our knowledge, it is the first work presenting contention periods of a production workload. The contention durations are calculated by measuring the consecutive timestamps of VMs that *CPU Ready* metric is above 10%. We observe a heavy-tailed distribution with few VMs having max contention durations of at least 1 hour (13%) while the mean duration of at least 1 hour is attributed to even fewer VMs (4.2%). The outliers of the figure depict a small number of VMs whose contention durations reach up to more than 2 weeks.

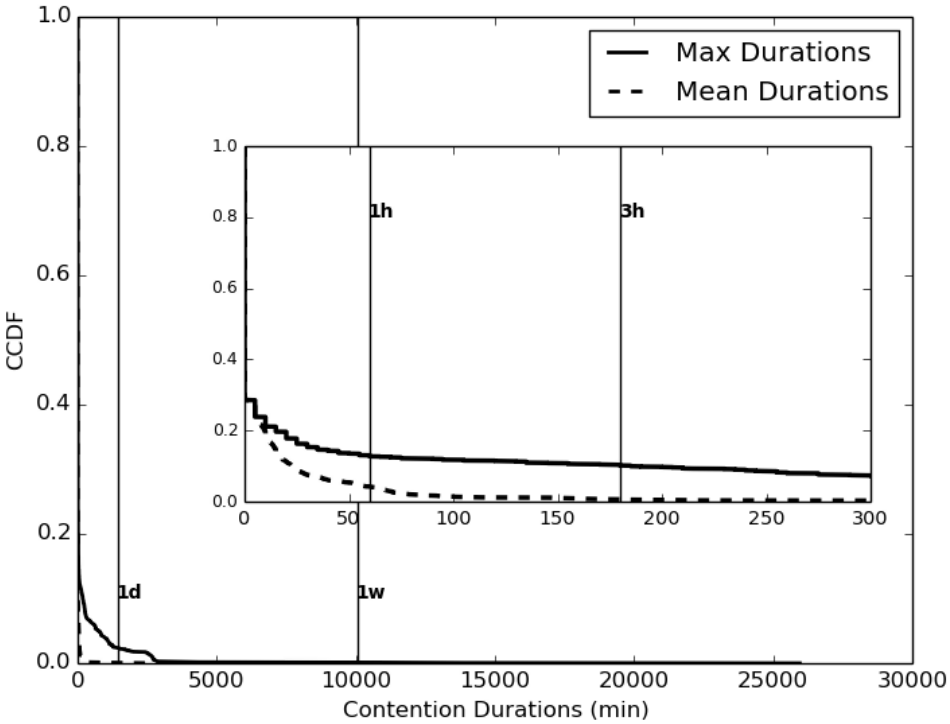


Figure 5.1: CCDF of contention Durations (min) in business-critical workloads.

5.3. Selection Method

We follow the modeling process described in [81] which can select a suitable model among various regressions models. The process is split into three steps as shown in Figure 5.2. More about the *Regression Analysis* process and relative terms can be found in Appendix A.2.

Firstly, we find correlated metrics to CPU-Ready metric from a set of monitored metrics (raw metrics - Step 1). The correlated metrics are candidates for independent variables of the predictor (regressors-predictor input) and the CPU Ready is the dependent variable of the predictor (predictor's output). Secondly, we select the correlated metrics from the first step and try out combinations of the selected metrics with different regression models to evaluate their accuracy (Step 2). In the end, we choose the regression model and the combination of regressors that together give the best accuracy of predicted values considering also the predictor runtime (Step 3). The values which the model predicts are the CPU-Ready values of VM v at time t since every VM experiences the contention according to individual characteristics.

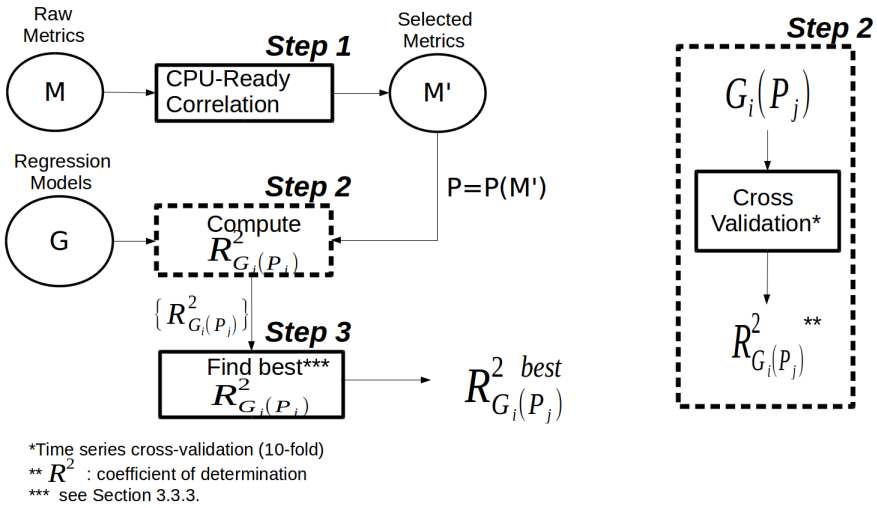


Figure 5.2: Selection Process. The Ellipses indicate sets of elements while the rectangles indicate a process.

5.3.1. Correlation-based Selection (Step 1 in Selection Method)

Addressing the first research sub-question, we collect multiple operational metrics and find their correlation with the CPU-Ready metric (Step 1 in in Figure 5.2). The metrics that have the highest correlations according to some boundaries are selected for the second step of the process.

There are hundreds of operational metrics that can be monitored and be candidates of our selection. The monitored metrics represent different levels of operations in VMs and in clusters starting from hardware performance counters of cpu cores up to cluster resource utilizations. The research in [13] and in [81] investigates many metrics for their purposes but we focus only on metrics showing the resource utilizations of VMs and of clusters preserving the concept of *opaque* VMs. As a result (see Results for Predictor Selection - Section 6), we end up with a high accurate predictor even if we start with only a few raw metrics (25 metrics).

We monitor the set M (see Figure 5.2) of metrics which consists of 25 performance metrics of VMs and of clusters. The performance metrics monitor all the four resource types we investigate in this work (CPU, memory, storage,network) and include metrics showing the resource utilizations (e.g. read/write KBs in network, used MBs in memory) and latencies of VMs (e.g.,read/write latency in storage, cpuWait in CPU).

Correlation in time-series We collect VM traces in Solvinity for almost three months (see Section 5.5 - Experimental Setup). The VM traces comprise of all the traces of the M metrics including the time-series of CPU-Ready. For each VM, we evaluate the correlations between the VM's CPU-Ready and the VM's metrics in the M set. The VM correlations regarding metric M_j and CPU-Ready are aggregated first by cluster (a VM is hosted by one cluster) and then by all the clusters reported in traces. In the end, for each metric $M_j \in M$, we have an aggregated correlation value depicting the correlation of the metric with CPU-Ready as well as the

p-value of the correlation. The p-value presents the validity of the correlation whether the correlation value is wrongly as high as it is calculated. The mean correlation coefficient of the correlation values of VMs and clusters is calculated using z-transformation [21] since the correlation coefficients are not additive quantities (ordinal scale type) to compute their average directly. To aggregating the p-values, we double the mean of the individual p-values [76].

Different correlation coefficients To calculate the correlations, we use two correlation coefficients and we present the result of both correlations in Section 6. The two coefficients, *Pearson* and *Spearman*, reflect different behaviors of data. Pearson's correlation (r) measures the linear relationship of data but it is susceptible to outliers. On the other hand, Spearman's correlation (ρ) measures a monotonic relationship of the variables because it applies ranks to data values and is more robust to outliers.

We use two different correlation types because, in Step 2 in the Selection Method, we investigate multiple regression models some of whom work better with linear data and other regress data accurately even with non-linear relationships among regressors.

In the end of the correlation step, we choose set $M' \subseteq M$ which includes the metrics with the highest correlation(r / ρ) with CPU-Ready and the lowest p-values ($p < 0.05$).

5.3.2. Model Accuracy (Step 2 in Selection Method)

Addressing the second research sub-question, we investigate multiple regression models to end up with the best model in predicting the CPU-Contention in the system. Step 2 in Figure 5.2 shows the process we follow to measure the accuracy of regression models given a set of input metrics (regressors).

As set of possible regressors for a regression model, we have the output of step 1 in Selection Method (set M') and it contains the most highly-correlated metrics with CPU-Ready among the raw metrics of set M. For each regression model in set G of regression models, we try out every combination of metrics in M'. The combination of metrics may comprise of one up to the total number of available metrics $|M'|$, therefore we define the power set $P = P(M')$ containing all the possible subsets of set M' as the input in Step 2.

Every regression model $G_i(P_j)$, where G_i is a regression model in G using as regressors the subset $P_j \in P$, is tested with cross-validation to measure the accuracy of the regression model. After calculating the accuracies of every possible $G_i(P_j)$, we select the best model (Step 3) under conditions explained in Section 5.3.3.

Cross-validation Cross-validation is a technique for model validation and is used to estimate how accurately a predictive model will perform in practice [60]. A predictor may work well with specific input datasets but may not generalize the same to an independent dataset. Therefore, by changing the input multiple times cross-validation achieves to reduce this uncertainty of a falsely good predictor. The technique splits the input dataset of the predictor into k folds of which some are used as the training dataset while the rest folds are used as the testing dataset. Due to the fact that we work with time-series predictions, we use a special version of cross-validation (*time-series cross-validation*) in which the folds are ordered by time, thus a fold used in training dataset must not include future data compared to data in the testing dataset. The accuracy of the predictor after cross-validating the model is the mean of k accuracies calculated by the k rounds of the technique.

Regression Models We investigate multiple regression models coming from both machine learning techniques, the Supervised and the Unsupervised learning methods. In supervised learning, we feed the predictor with desired output values to adjust the model's internal states while in the unsupervised method, the model tries to identify the desired output without any exterior information. Additionally, we try to investigate regression models working well either with linearly or with non-linearly correlated data as mentioned in Section 5.3.1. In the end, we conclude in different types of *Linear Regression* (**LR**), *Curvlinear Regression*, *k-Nearest Neighbors regression* (**k-NN**) and *Gradient Boosting regression* (**GB**).

The regression models along with some descriptions about the regression approach of the models are presented in Table 5.1. *Linear Regression* is the typical approach to model relationships on data by minimizing the error between the training data and the produced line. *Lasso* ,*Ridge* and *Elastic Net* work in a similar way with *Linear Regression* but they penaltize the errors with cost functions derived from the norms of $L1$,$L2$ and both norms respectively. To differentiate more the models, we use multiple solvers in *Ridge* to find the best fitting line in order to investigate the runtimes of finding a solution. *Lars* is an approach which adds available regressors to the model only if there is a significant improvement in the accuracy, thus resulting in

a model with a few meaningful regressors. *Polynomial Regression*, or curvlinear regression, fits the data using polynomial functions which can explain nonlinear relations of data while *k-NN* clusters the training data according to the regressors values and reveals trends of the values of the independent variable (CPU Ready in our case). Finally, *Gradient Boosting* regression uses cost functions l such as least-squares or quantiles (Q) and builds (iteratively) decision trees sized n that indicate which output values reduce the costs. This method is a multi-parametric method but we use some of the possible settings.

Regression Model	Description
Linear Regression (*LR*)	Fit data points to a line by minimizing least squares distance
Lasso	Least squares Model with regularization [1] using L1 norm
Ridge	Least squares Model with regularization using L2 norm
Elastic Net	Least squares Model with regularization using L1,L2 norm
Lars	Stepwise regression using forward selection
Polynomial Regression	n-th degree of polynomial regression
k-NN	Prediction according to the properties of k nearest neighbors
Gradient Boosting (*GB*)	Regression using decision trees with n estimators by applying loss function

Table 5.1: Description of regression models.

R^2 - Coefficient of determination The coefficient of determination (R^2) [26] is a metric used to assess the accuracy of a predictor model. It evaluates the proportion of the variance in the predictor output (independent variable) that is predictable from the input (regressors). The range of the metric is $[0,1]$, the higher the better, although it can have negative values if the selected regressors are meaningless (our interest is in the non-negative range). We can use this metric to measure each predictor's accuracy (see $G_i(P_j)$ before) in Step 2. The metric limitation is that R^2 is monotone increasing, meaning that always increases when we add extra regressors to the model. We address this issue in Section 5.3.3.

In the end of Step 2, we have a list of all models $G_i(P_j)$ (see Figure 5.2) with their respective R^2 scores. This list is the input of the final step of our selection process.

5.3.3. Predictor Selection (Step 3 in Selection Method)

In the final step, Step 3 in Figure 5.2, we select the most promising predictor for estimating CPU contention. We make our selection among the list of models $G_i(P_j)$ considering three factors: a) The accuracy of the model depicted by the R^2 score, b) the runtime of the regression model and c) the number of regressors in the model. The first factor indicates better performance of the regression model in terms of the proportion of the variance of the real values being estimated by the predictor while the second factor is needed to limit the computation runtime of the prediction process since some regression models run for long time. The third factor applies for having as few regressors as possible given the first two factors. The many regressors in the model do not add significant accuracy to the model (inflation of R^2) as well as they impose extra overhead to the computations.

The first two factors (a) and (b) are easily to be addressed by sorting the results in an increasing order for the R^2 scores of the models and in a decreasing order for the runtime of the models. As for the third factor, we leverage the method in [81] where we add a new regressor in the model only if it improves the R^2 score more than a given threshold (R^2_{impr}). A metric contributing to the predictor's accuracy less than R^2_{impr} will not be considered as an additional regressor.

The results of our selection method are presented in Section 6 where we show the best accuracies for every type of regression model we use. As a result of the selection process we present in this chapter, we choose as our predictor for CPU-contention in business-critical workloads the model in Equation 5.1. Table 5.2 summarizes the terminology used in the predictor model. The selected regression model is *Linear Regression* .

$$G_i(P_j)^t_v = \mathbf{LR}(d^t_v, d^t_c, n^t_v, r^{t-1}_v, \overline{r^{t-1}_c}) \tag{5.1}$$

[1] Penaltizing bad fitting of data, mitigating data overfitting

Term	Description
d_v^t	Demanded CPU of VM v at time t
d_c^t	Demanded CPU of cluster c hosting VM v at time t
n_v^t	Number of virtual CPU cores of VM v at time t
r_v^t	CPU Ready of VM v at time t
r_v^{t-1}	CPU Ready of VM v at time $t-1$
r_v^{t-2}	CPU Ready of VM v at time $t-2$
$\overline{r_c^{t-1}}$	Mean CPU Ready of VMs in cluster c hosting VM v at time $t-1$

Table 5.2: Terminology in Selection Method for CPU-contention predictor.

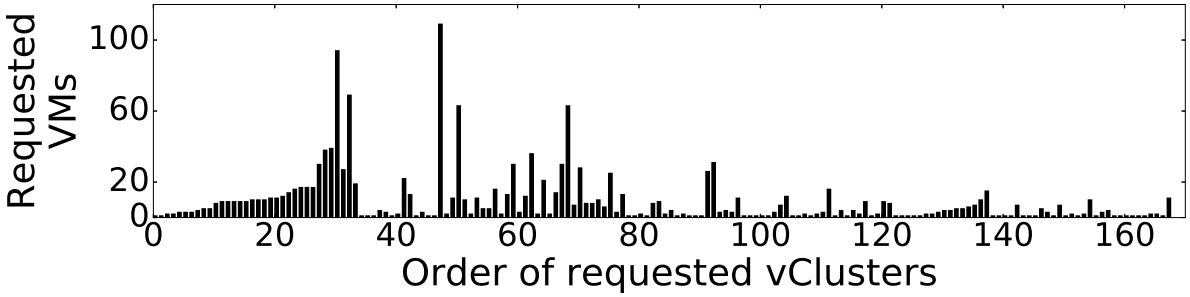


Figure 5.3: Sequence of vCluster arrivals in the real multi-datacenter workload, over a period of 3 months.

5.4. Summary

In this chapter, we address our research question on selecting an efficient predictor for CPU contention among VMs running business-critical workloads. We present a method through which we pick metrics correlated to CPU contention and we use combinations of those metrics to investigate the accuracy of multiple regression models. The method we propose may be used for selecting a predictor of a different contention metric than CPU Ready or even of a performance metric with different purpose.

Selecting an efficient predictor does not only require accuracy measurements since the prediction process might be time-inefficient. Thus, we also consider the computation overhead of the predictions in our selection process. The CPU Contention predictor will be incorporated to portfolio scheduler's architecture presented in the following chapter.

5.5. Experimental Setup

A major contribution of this work is the experimental analysis of the portfolio scheduler and the impact of the presented utility functions on the scheduler's decisions. In this section, we present the setup of our experiments.

5.5.1. Representative Environment and Workload

We use one representative trace of real-world commercial workloads hosted in a multi-datacenter multi-cluster infrastructure. The workload trace follows for about 3 months the real operation of the cloud operator, recording business-critical workload and the datacenter performance metrics. We use the former as **input workload** for our simulations, and the latter as **baseline for comparison** (*REPLAY* in Figures 6.2b and 6.3b).

Typically for business-critical workloads, the recorded requests combine HPC, web, and many other types of applications; this leads to various anti-/affinities expressed in the request, e.g., HPC VMs have a strict requirement to run on HPC compute nodes in the multi-datacenter environment, a vCluster running a replicated database requesting to run its VMs on separate clusters or datacenters, etc. Due to NDAs, we cannot reveal more details about the features of the current datacenter infrastructure.

The trace includes a diverse stream of over 160 $vClusters$, depicted in Figure 7.1, to be placed in 12 clusters having about 200 physical hosts in total.

5.5.2. Portfolio Setup: 2 New and 7 Common Scheduling Policies

For our experiments, we equip the portfolio scheduler with the 2 new policies we design in this work (Section 4.4), plus 7 common policies. This allows us to see how both our new policies and the complete portfolio scheduler perform; if our policies are not useful, they will simply not be selected by the (automated) portfolio scheduler. The 7 common policies are:

1. First-Fit (FF) is a commonly-used policy, simple but applicable in many domains of workload scheduling and VM placement. To place a request, FF chooses the first available cluster in which the request fits.

2. Type Priority (TP) extends FF to also consider request types (e.g., HPC), which are then matched against cluster capabilities. This enables specialized clusters to execute efficiently demanding workloads, such as HPC. TP may be a competitive policy for matching complex SLAs. In contrast to FF, TP will try to not place non HPC workloads on HPC clusters.

3. Memory Datacenter Balance (MDB) is a policy that uses memory utilization to evenly distribute the VMs over datacenters. MDB is a possible competitor to our new policies in lowering the DRR in the system.

4-7. Lowest Resource Load (L*) is a family of policies that attempt to balance over all clusters the loads of a specific resource, e.g., memory. We consider in this work all four common resource types, with the specific L* policies: Lowest CPU Load (**LCL**), Lowest Memory Load (**LML**), Lowest Storage Load (**LSL**), and Lowest Network Load (**LNL**).

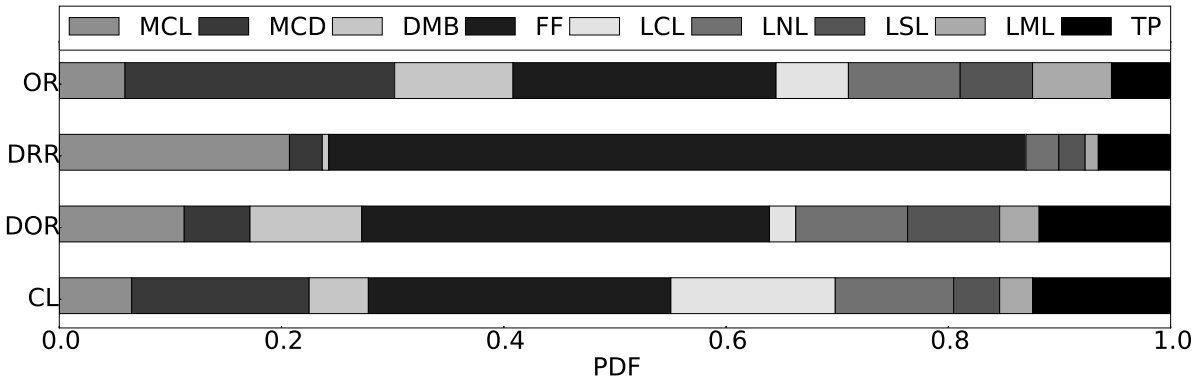


Figure 6.1: How many times is each scheduling policy selected by the portfolio scheduler, per utility function (e.g., OR).

6

Experimental Results

In this section, we analyze our portfolio scheduling architecture for managing Operational and Disaster Recovery Risks in virtualized datacenters hosting business-critical workloads. Through trace-based simulations based on real-world multi-datacenter workloads, we evaluate our portfolio scheduler when equipped with 9 scheduling policies (see Section 5.5.2), and compare it with the state-of-the-art and with a relevant real-world baseline (see Section 7.2.1). We follow two main aspects: whether selection of different policies actually occurs, and analyzing the evolution of risk over the studied period (lower is better!).

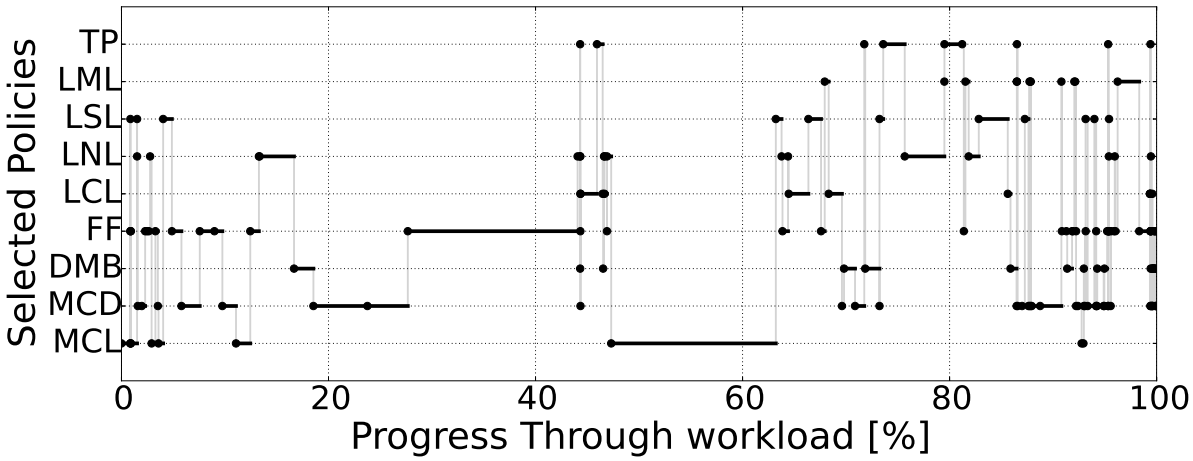
In section 6.0.1, we show how much different scheduling policies are selected. In Section 6.1, we investigate the performance of portfolio scheduler for the different utility functions, and compare the results with the real-world baseline. Finally, in Section 6.2, we show the performance of the portfolio scheduler when used to balance multiple objectives (the realistic situation in datacenters).

6.0.1. Is each Policy Useful?

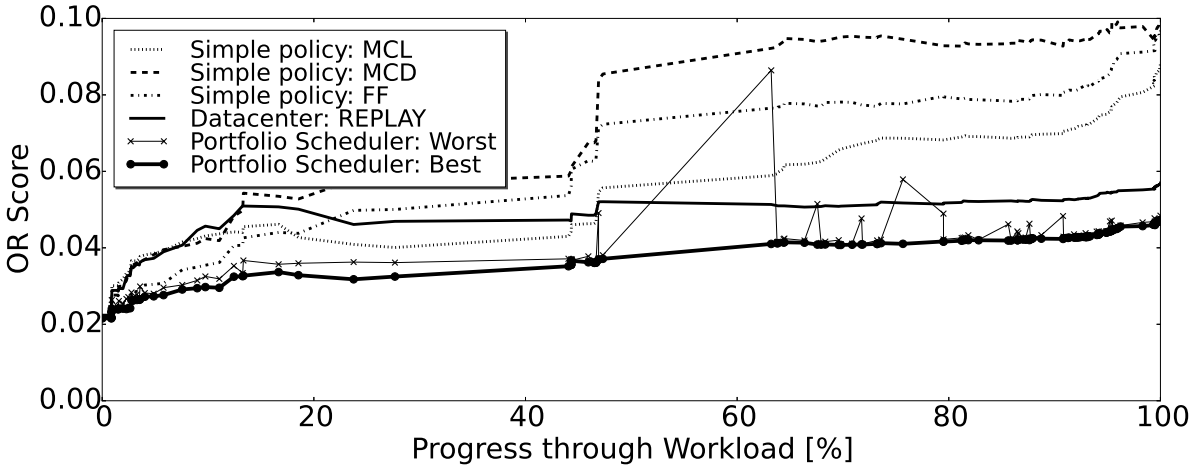
The intuition of portfolio scheduling is that under changing workloads or changing datacenter optimization goals (utility functions) different scheduling policies are selected. Figure 6.1 depicts the distribution of selected policies, for our new utility functions (OR, DRR, and DOR), and for the utility function Cluster Load (CL). CL is included to compare this work with our previous state-of-the-art work on portfolio scheduling [74]. The main findings are:

1. All policies are selected for all utility functions.
2. Using different utility functions results in widely different distributions of selected policies.
3. MCL and MCD are selected 16-30% of the time.

Figure 6.1 breaks down in each of its horizontal bars the distribution of selected scheduling policies for each utility function. Each horizontal bar includes boxes, each corresponding to one scheduling policy, whose *width* indicates the fraction of times the scheduling policy was selected, from the total set of decisions taken by the portfolio scheduler. In Figure 6.1, each complete horizontal bar is highly segmented, except for



(a) Selected scheduling policies for the OR utility function.



(b) Evolution of OR over the entire workload.

Figure 6.2: Results for Operational Risk.

DRR. When DRR is tha active utility function the FF policy is selected about 60% of the time. This explains why, despite not being designed to address complex performance and risk metrics, it is still much used by datacenter engineers. However in next Section we will show that solely selecting FF will result in much higher DRR risk than our portfolio scheduling approach can achieve.

Across all utility functions, we see a wide diversity of distributions. FF, LSL, LNL, TP, and our two policies MCL and MCD all account for over 10% of the selections for at least one utility metric.

6.1. Does Portfolio Scheduling Improve System Utility?

Is the portfolio scheduler improving the utility provided by the system? The main findings are that the portfolio scheduler:

1. Decreases OR by up to 2x, vs. individual policies.
2. Decreases OR by up to 1.5x, vs. the real-world baseline (REPLAY).
3. Improve DRR significantly compared to just using FF up to 35%.
4. Improve DRR significantly up to 40%, even for a well-managed commercial datacenter (REPLAY).

6.1.1. Operational Risk

Figure 6.2a depicts the selected scheduling policies when the OR utility function is active, meaning that the portfolio scheduler is set to optimize for OR. We create a new type of graph for this depiction, which the

portfolio scheduler can also use to *explain its decisions to datacenter stakeholders*. The vertical axis is divided into parallel rows, each representing the progress (that is, non-/selection) of a different scheduling policy. For example, the new policies MCL and MCD occupy the two bottom-most rows. The horizontal axis depicts the progress through the workload: large dots placed on the rows indicate that a policy has been selected, whereas the thick horizontal bars emerging from the dots indicate the portion of the workload that is to be scheduled by the selected policy (wider bars indicate a higher portion). The last element of this graph is the set of vertical gray lines, which indicate transitions between consecutively selected policies. A deeper analysis of this type of graph is outside the scope of this work.

Figure 6.2a includes many transitions, and in particular each scheduling policy is selected at least once (each row corresponding to each policy has at least one large dot). In some cases, policies are selected multiple times in succession.

Figure 6.2b depicts the evolution of the OR utility-function value (*score*), for different scenarios (lower score is better). The curves depicted for the portfolio scheduler are for the actually selected policy ("Best Policy"), and for a worst-case scenario of selecting the worst possible policy in that particular decision moment ("Worst Policy"). The figure also shows comparisons with the use of a single policy (FF, and our policies MCL and MCD) and with the decisions taken in the real-world by datacenter engineers in the baseline for comparison ("REPLAY"). The portfolio scheduler results in much lower Operational Risk compared to selecting individual policies (up to 2x better) and to the REPLAY baseline (up to 1.5x better). Thus, using a portfolio scheduler leads to much lower risk of violating OR SLAs.

To explain why this happens, observe in Figure 6.2b that, at around 50% through the workload, the worst scoring policy shows a big spike—one of the policies performs significantly worse than the best policy. This shows that portfolio scheduling can effectively alleviate the risk of scheduling policies performing badly sporadically.

6.1.2. Disaster Recovery Risk, Operational Risk, and Cluster Load

Similarly to OR, for the DRR utility function we depict the results in Figures 6.3a and 6.3b. Overall, we observe that the approach results in lower risks of violating SLAs with regards to Disaster Recovery guarantees. What is interesting to notice is that even for a multi-datacenter setup (operated by highly trained engineers, they hire only top-10% engineers in the market) evaluated in this study we see good opportunities for significant improvements. Figure 6.3b shows that using the portfolio scheduler can lead to a 40% improvement over the real-world baseline (REPLAY).

Similarly, for both DOR and CL we also investigate the policy selection and utility function scores. The portfolio scheduler results are positive, in-between the results for OR and DRR (for complete results, see the extended results section: Section 7).

6.2. Does Addressing One Risk Influence the Other Risks?

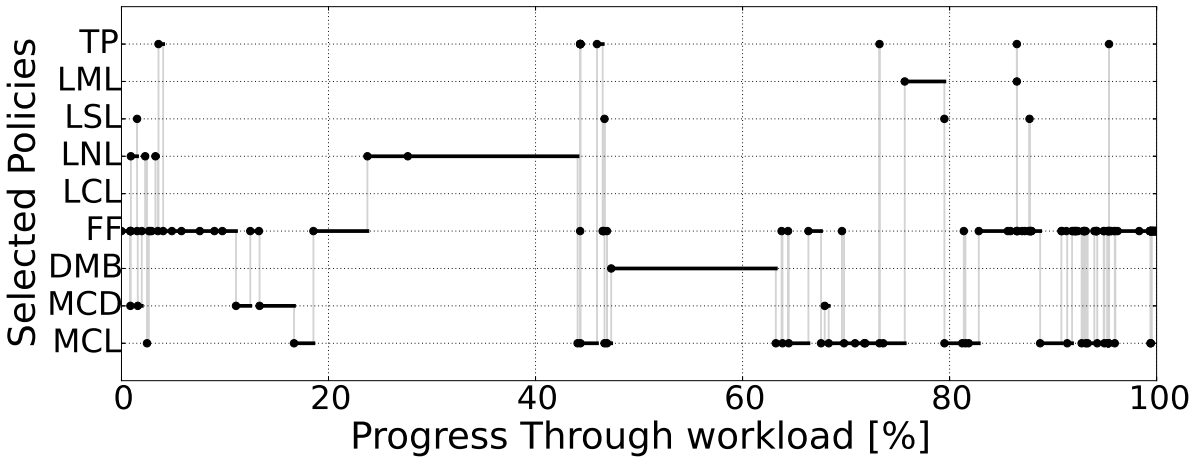
Datacenter operators often use multiple utility functions (UFs) at the same time to optimize for multiple objetives at the same time. We evaluate in turn each UF and analyze how this affects the maximum scores for all the other UFs; Table 7.2 summarizes the results and shows that:

1. DOR can be used to effectively balance DRR and OR.
2. If DOR 3-1 is used, both DOR and OR are very close to their (minimum) best.
3. Activating CL worsens all the other utility functions.

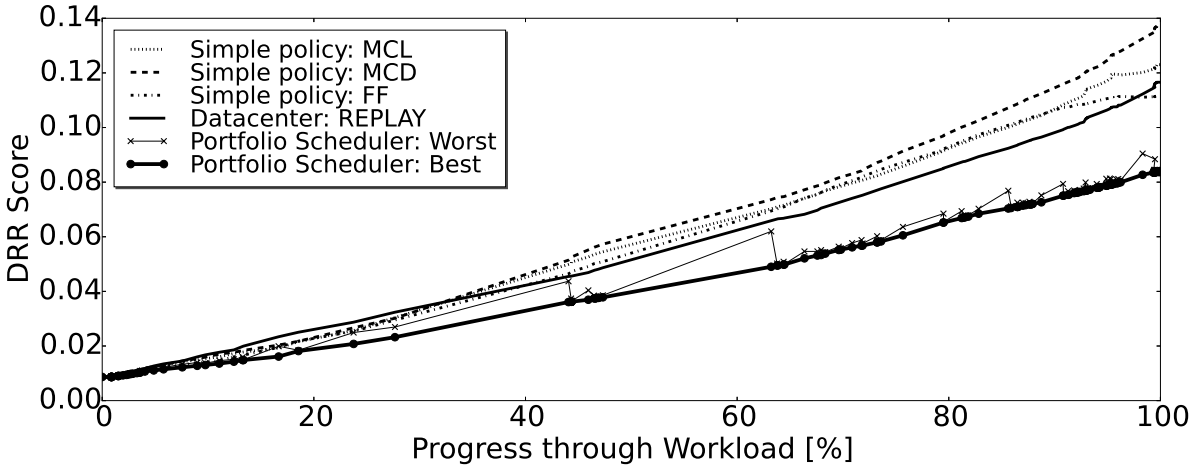
The results in Table 7.2, when considered column-wise, indicate the influence that selecting an UF (e.g., OR) has on the other UFs. When considered row-wise, the results allow comparing for an UF its scores when it is active and when other UFs are active. If we optimize for the Disaster-Recovery Risk (column "DRR" in Table 7.2), the OR score will increase up to 2x (risk nearly 2x worse), compared to the case when OR is active. Conversely, activating OR increases DRR. Thus, DOR explores an important risk-management trade-off.

The results for DOR $w_o - w_b$ for the 1-1, 3-1, and 1-3 settings, show that the OR and DRR utility functions can be combined and, depending on the datacenter SLAs and optimization target, can be balanced. When more emphasis is put on OR, by increasing w_o to 3, column "DOR 3-1" values show that the OR score is very close to its achievable minimum (the OR score when OR is activated).

We also compare with the state-of-the-art UF CL [74], which balances load distribution for one or more resource types (e.g., CPU, Memory, IO). Activating OR actually leads to the best outcome we have observed



(a) Selected scheduling policies for the DRR utility function.



(b) Evolution of DRR over the entire workload.

Figure 6.3: Results for Disaster Recovery Risk.

Table 6.1: Maximum utility function (UF) scores for all UFs, for each activated UF. For DOR, we report results for three different settings (columns DOR $w_o - w_d$). Column R presents the Replay scores.

Active UF − >	**OR**	**DRR**	**DOR 1-1**	**DOR 1-3**	**DOR 3-1**	**CL**	**R**
OR score $\times 10^{-2}$	5	10	5	7	5	7	6
DRR score $\times 10^{-2}$	12	8	11	9	11	12	12
DOR score $\times 10^{-3}$	8	9	8	9	6	9	9
CL score $\times 10^{-5}$	4	16	3	13	10	4	5

for CL. This means that our new OR utility function has a better long term effect then directly optimizing for CL. Conversely, activating CL worsens all the other utility function scores, so the state-of-the-art CL is a bad predictor for the optimization goals considered in this work.

7

Extended Evaluation of Portfolio Scheduler Managing Risks in Business-Critical Workloads

In this chapter, we evaluate our portfolio scheduler by conducting experiments to investigate the risks in business-critical workloads. Moreover, we present the results of our method described in chapter 5 on selecting a CPU-contention predictor for business-critical workloads.

7.1. Overview

For the evaluation of portfolio scheduler, we first describe the high-level architecture of the environment (Solvinity infrastructure) we record our traces. We then describe the workload traces and the characteristics of VMs comprising the traces. The workloads are: the real-world workload and the synthetic workload. Further, our experimental setup is presented for the two experimental groups, the experiments for the predictor selection and the experiments for the portfolio scheduler evaluation. We finish the chapter by presenting the results of our experiments.

The main goals of this chapter is to provide insights into risk-aware portfolio scheduling and into the potential of portfolio scheduling for business-critical workloads towards a production ready scheduling system.

7.2. Experimental setup

A major contribution of this work is the experimental analysis of the portfolio scheduler and the impact of the presented utility functions on the scheduler's decisions. In this section, we present the setup of our experiments.

7.2.1. Representative Environment

For the evaluation of the portfolio scheduler for business-critical workloads in a multi-cluster, multi-datacenter setting, we use the infrastructure of Solvinity. Solvinity has 3 datacenters , each one comprising a multi-cluster topology. The datacenters are connected through a fiberoptic ring network while compute servers in clusters are connected through a low latency InfiniBand network. Evach cluster houses 6 Storage Area Network (SAN) devices, 10TB each.

In Solvinity, there are two types of clusters, the standard cluster and the bigmem cluster. The servers in a bigmem cluster have a higher memory to CPU ratio than the servers of a standard cluster. This is the only difference between the two cluster types. The standard servers have two 8-core CPUs and 128GB of memory. In constrast, the bigmem servers have four 8-core CPUs and 768GB of memory.

The VM placement in clusters is handled by cloud engineers considering the SLAs agreed for every customer application. For the experiments, we follow the portfolio scheduler's decisions to place the VMs in clusters. The scheduling of VMs in the compute servers is handled by VMWare product, the DRS. The automigration of VMs for any scheduling reason among servers in the same cluster is enabled while the VM automigration across clusters is disabled.

7.2.2. Representative Workloads

We evaluate our concepts by using two workloads: a production workload recorded in Solvinity infrastructure (Real-world workload) and a synthetic workload which we generate comprising of VM traces of the real-world trace but with different characteristics. The two workload sets will provide different insights for our portfolio scheduler since we use the real-world workload to evaluate a real world case and the synthetic workload to stress the system to identify corner cases of portfolio scheduler.

Real-world workload

We use one representative subset of real-world workload trace recorded in Solvinity infrastructure and spanning an almost 3-month period from 1 August 2015 until 22 October 2015. The trace includes a diverse stream of over 160 $vClusters$ with 1800 VMs, depicted in Figure 7.1, to be placed in 12 clusters having about 200 physical hosts in total. The clusters reside on the subset of three datacenters of the total infrastructure. We present the system load in terms of memory because the workloads are in line with restrictions of non-overcommitted memory. Therefore, the provisioned memory by the workload represents the allocated capacity of the total system memory capacity. The system load by the real-world workoad reaches up to 32% of the total memory capacity in the system. This is because the system must be able to absorb the workload resulting from a full-datacenter crash into the other (two) datacenters.

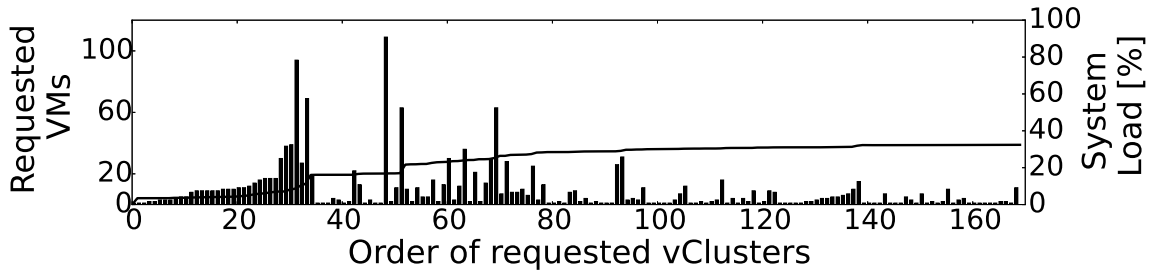


Figure 7.1: Workload Distribution - Real-workload trace (RT).

The VMs in the trace belong to different subtypes of business-critical applications such as multi-tier web applications, firewall applications and HPC workloads. For every subtype of business-critical workloads, there are guarantees about the consolidation levels inside the hosts. For HPC compute nodes, the consolidation should be low (around 10 VMs/host) as many contention problems occur due to CPU-intensive utilization while web applications can be packed more due to different utilization patterns (around 50 VMs/host).

In simulation, we follow for each $vCluster$ every anti-/affinity rule, as in the real-world situation. For every placed $vCluster$, the portfolio scheduler decides on the most promising policy for the overall system performance.

Synthetic workload

We generate a synthetic workload to evaluate our portfolio scheduler when the system is under stress. The total workload fully provisions the memory capacity in the system, thus the total load of the workload coincides with the total system load. We select vClusters from the real-world trace and populate each one of them with more identical VMs of the same vCluster. The result is representative business-critical vClusters produced from the production workload and provisioning more resources in line with the characteristics of business-critical workloads, e.g., CPU-memory ratio, resource utilizations. For the generation of the workload, we focus on vClusters comprising web-application and HPC workloads since these types cover most of the application spectrum of enterprise customers. We do not follow anti-/affinity rules of the VMs in the synthetic workload because we want to evaluate the general case on which every VM can be scheduled in any place.

In Figure 7.2 we present the profile of the synthetic workload comprising 57 vClusters with 3500 VMs in total. The vClusters include more VMs now ranging from 10 up to more than 300. We see that the system memory load of the synthetic workload reaches the full capacity when all the workload will be placed in the system. We expect that vCluster placements will stop before the full system load since loading fully the system is a hard optimization problem. In the remainder of the chapter, we refer to the workload point where the system cannot host more workload as *system breaking point*.

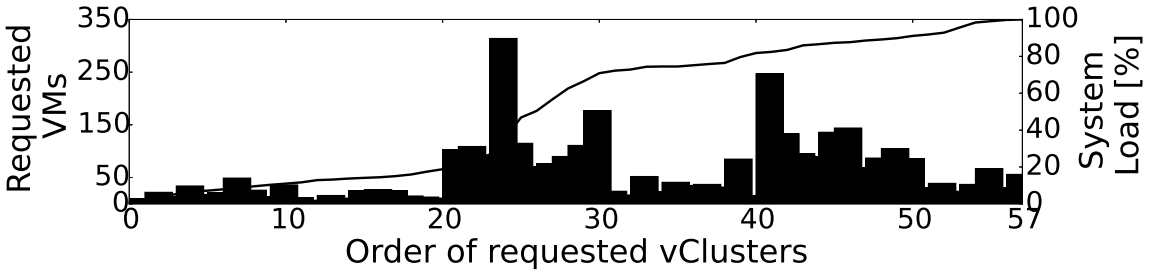


Figure 7.2: Workload Distribution - Synthetic trace (ST).

7.2.3. Configuration for Predictor Selection results

We present the experimental setup describing the decisions we make about conducting the experiments for our selection method. Due to the long-running process of the method presented in chapter 5, we choose a subset of the trace described in Section 7.2 to correlate the different metrics with the CPU Ready metric and also to train and to test the regression models. The subset of the real-world trace we experiment on Predictor Selection comprises traces of VMs running at most 1 month (the total trace has duration of 3 months).

We decide on having separate predictors for every cluster in the infrastructure and not one general predictor for the whole system. The reason behind this is that clusters may have different system architectures in a multi-cluster multi-datacenter system and VMs experience resource contention because of their "local" environment which includes the co-hosted VMs in the cluster. In addition, in the current trace we collect from Solvinity, the VMWare schedulers are allowed to schedule VMs only inside clusters (scheduling between servers in a cluster), thus the produced contention of VMs is more related to the cluster environment of a VM.

At **step 1 in Selection Method** (see Section 5.3.1), we choose the metrics having high correlation with the CPU Ready metric which is the contention indicator for our evaluation. To begin the investigation of the correlated metrics with CPU Ready, we select 25 metrics (set M in Figure 5.2) whose traces are included in the real-trace workload described in Section 7.2.2. This set of metrics comprise metrics from all available resource types (CPU,memory,storage I/0, network I/O). Most of these metrics refer to VM utilizations while the rest indicate resource utilization on cluster level. In addition, we investigate the auto-correlation of CPU-Ready in VMs to identify possible relations of current CPU-Ready levels with those in the past.

We set the correlation value between a raw metric and the CPU Ready metric to be higher than $r/\rho >= 0.4$ and the p-value less than $p < 0.05$ to select a raw metric as a candidate regressor of the predictor.

At **step 2 in Selection Method**, we form the set of regression models (set G in Figure 5.2) to combine them with every combination of the correlated metrics found at step 1. The regression models are tested with several parameters to identify the most suitable estimating CPU contention. The selected parameters in our experiments are presented in Table 7.1.

At **step 3 in Selection Method**, we use metric R^2 and the runtime of the regression models to select the predictor with high accuracy and low computation overhead as explained in Section 5.3.3. For the value R^2_{impr} limiting the selection of many regressors in the prediction model, we set it to $R^2_{impr} = 0.005$.

Regression Model	Parameters
Linear Regression (*LR*)	with/without intercept
Lasso	weight $l_1 \in [1,4]$
Ridge	weight $l_2 \in [1,4]$, multiple solvers [1]
Elastic Net	weights $l_1, l_2 \in [1,4]$
Lars	-
Polynomial Regression	$n \in \{2,5,8\}$
k-NN	$k \in [2,10], neighborchoice = euclidean$
Gradient Boosting (*GB*)	$l = LR, Q2 \quad n = 400,800 \quad learning_{ratio} = 0.01$

Table 7.1: Parameters in regression models in Step 2 in Selection Method.

[1] We use different techniques to test the speed of converging to a solution

7.2.4. Configuration for Portfolio Scheduling results

We present here the experimental setup describing the configurations we make about about the evaluation of portfolio scheduler.

As mentioned in Section 1 in the System model, it is essential to monitor system performance, thus load balance of the workloads, despite of our risk objectives. Therefore, in addition to the three utility functions OR, DRR, DOR presented in Section 4, we experiment with a fourth metric, the **Cluster Load score (CL)**, evaluating how balanced the workloads are across the available clusters [74]. The score calculates the system mean utilization (U_i) and compares it with the mean cluster utilization over time. The bigger the deviation of the mentioned utilizations for a cluster, the more imbalanced the load in that cluster is. This metric gives an insight on system-performance level since severe load imbalances affect the performance of applications and impose high risk on performance degradation of VMs.

The CL score evaluates the cluster load imbalances regarding the four resource types of CPU, memory, storage and network I/O. Firstly, we calculate the deviations of cluster loads for each resource type and then we aggregate the results having in the end one score value for the policy selection step of portfolio scheduling. Equation 7.1 presents CL score of a cluster in terms of resource type i and Equation 7.2 presents the aggregated score (system CL) used in our experiments. Lower CL values indicate low imbalance on the workload distribution across clusters.

$$r_{cl_i} = \frac{(\frac{W_{cl_i}}{C_{cl_i}} \cdot 100 - U_i)^2}{100^2} \in [0,1] \tag{7.1}$$

with i be the resource type.

$$r_{cl} = \frac{\sum_{j=1}^{m} \sum_{i=1}^{n} r_{cl_{ij}}}{n \cdot m} \in [0,1] \tag{7.2}$$

with n be the number of resource types and m be the number of clusters.

For the portfolio of scheduling policies, we use the 9 scheduling policies presented in Section 4. The policies consider different resource types to place VMs and 5 of them focus on either CPU or memory. We decide on having more policies about these two resource types as in business-critical workloads CPU and memory are the dominant resource types considered by engineers when placing VMs.

In addition to the scheduling policies, we also evaluate the real datacenter decisions made by the engineers in the same workload trace. We use the real placements of VMs in the datacenters, *REPLAY*, as **baseline for comparison** in the results of portfolio scheduler evaluating the performance of utility functions (see Section 7.4.2).

We activate one utility function per experiment to test both portfolio scheduler's behavior on the selected policies and the performance of the activated utility function. In every experiment, we use the 9 scheduling policies+*REPLAY* to fairly compare the performances in the end. For training the CPU-contention predictor, we collect data related to contention for a time window of 5h. This time window is also used for the experiments of predictor selection as it is explained in Section 7.3. For the utility function DOR, which is a combination of OR and DRR, we use different weight values to identify the most suitable combination of them when comparing the performance of different utility functions (see Section 7.4.3).

7.3. Results for Predictor Selection

In this section, we present the results of the selection method for a CPU-contention predictor described in chapter 5. The selected predictor is incorporated into our portfolio scheduler to estimate the performance degradation in the system in terms of the delivered CPU resources to the hosted VMs. The predictor estimates the CPU Ready levels of every running VM in the system at time t (r_v^t). This section addresses the first research sub-question of this chapter.

7.3.1. Results of Step 1 in Selection Method

As a result of this step, we have a set of 6 selected metrics (set M' in Figure 5.2) having correlation with the CPU Ready metric. The correlation coefficients of these selected metrics are higher than our limits mentioned in

the experimental setup for either the Pearson's (r) or the Spearman's (ρ) correlation. Table 5.2 summarizes the terminology used for the metrics of set M'.

The six selected metrics are: a) the demanded CPU of a VM at time t (d_v^t), b) the demanded CPU of the cluster hosting a VM at time t (d_c^t), c) the number of provisioned virtual cores of a VM at time t (n_v^t), d) the CPU Ready of a VM at time $t-1$ (r_v^{t-1}), e) the CPU Ready of a VM at time $t-2$ (r_v^{t-2}) and f) the mean CPU Ready of VMs in the cluster hosting a VM at time $t-1$ ($\overline{r_c^{t-1}}$). .

In Figure 7.3, we present the results for the selected metrics. We show the correlation coefficients between each selected metric and the CPU Ready r_v^t. Besides from the correlation values for every cluster, we show the mean correlation coefficient which is used to select a metric for the step 2 in selection method. The other raw metrics that are not selected through step's 1 process have either lower correlations than our boundaries or higher p-values than the respective p-boundary. In Figure 7.3a we present the auto-correlation of CPU Ready having been calculated by aggregating the autocorrelations of all the VMs in the trace subset. The trace has a monitoring period of 5 minutes and we present the lags in hour units. We see that for a very short time period in the recent past, there is a significant correlation between the current CPU Ready r_v^t and its value at a previous timestamp. For this reason, we select the two last entries, r_v^{t-1} ($corr = 0.85$) and r_v^{t-2} ($corr = 0.79$), as our candidate regressors in step 2.

In Figures 7.3b-7.3e we show the correlations of the other selected metrics which will be the input of the process in step 2. We present the Pearson's and the Spearman's correlation coefficients for every cluster as well as the mean correlation values. The differences of the correlation values among the values of the same correlation type are small implying stable correlation levels between the metrics. The only exception to the similarity of the correlations of the clusters is the correlations of the provisioned virtual cores of a VM and its CPU Ready values (Figure 7.3d). The CPU Ready metric counts the time until the first vCPU of a VM is scheduled on a physical core, thus the relation between this metric and the total number of vCPUs of a VM is not concrete. However, we observe that even though the correlations of clusters differentiate a lot, the mean correlation is high enough to select the number of vCPUs n_v^t as a possible regressor.

7.3.2. Results of Steps 2 and 3 in Selection Method

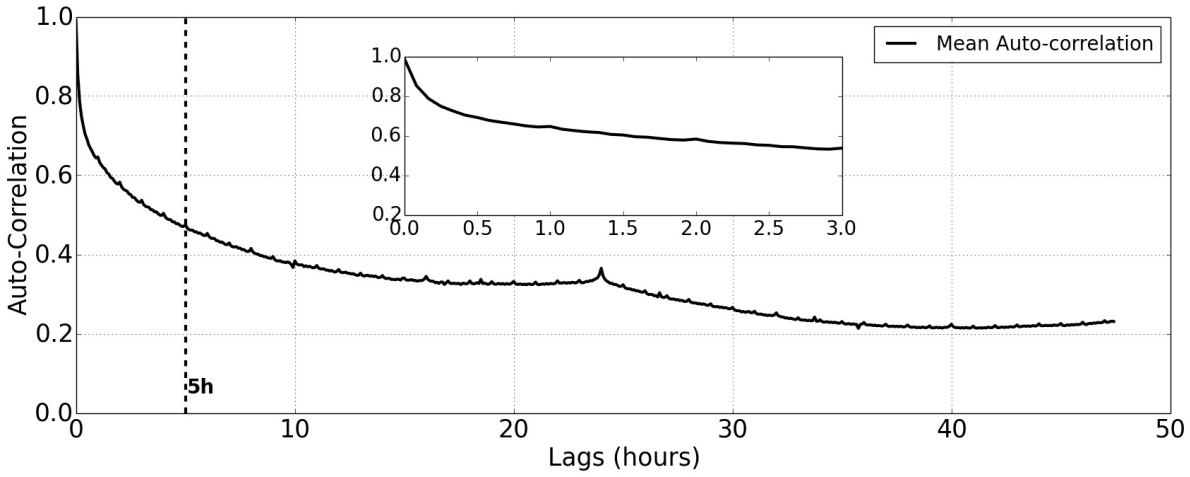
At step 2, we try every combination of the six selected metrics described in Section 7.3.1 with each regression model in Table 5.1. We test every pair of regression model and set of metrics with every (sliding) time window of 5 hours in the trace and we calculate the accuracy of the pair by aggregating the accuracies of the time windows. The reason we split the trace into time windows is that the CPU contention levels for VMs is a temporary effect (we also see it in Figure 7.3a where the values are related to values of the recent past-5h data) and a predictor should be trained only by recent contention levels.

Step 2 In Figure 7.4 we present the best (highest) accuracies that every regression model achieved in step 2. Every regression model ends up in these accuracies with the same or different set of metrics (the sets always comprise more than 3 metrics).

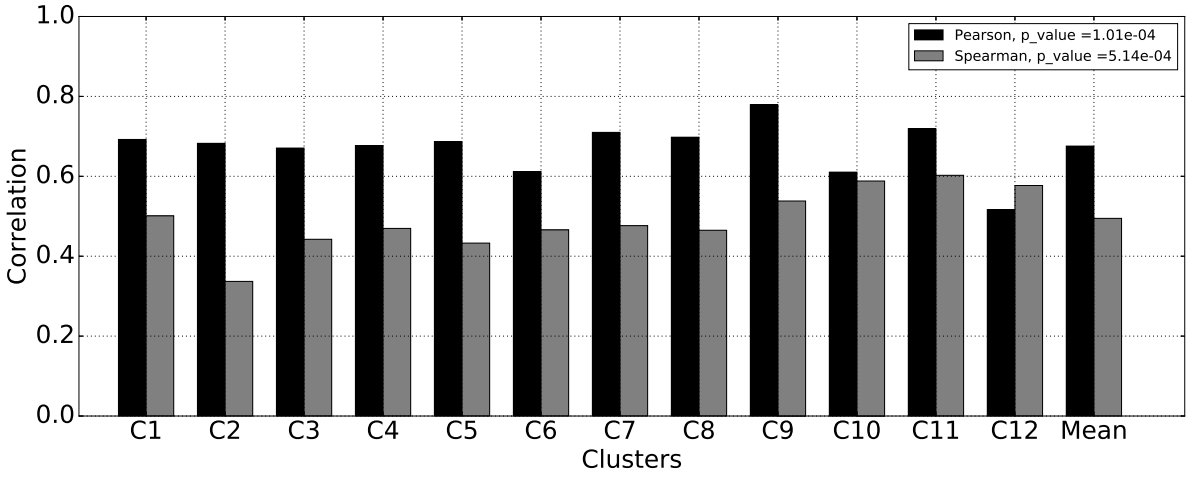
Figure 7.4 depicts for each regression model the average of R^2 accuracies of the time windows along with the standard deviation of those accuracies. We see that *Gradient Boosting* (GB) achieves the best accuracy (0.89) from all the other regression models while we have three models in the second place (*LR* =*Lars* =*Ridge* =0.86). The *Polynomial* model achieves impractical accuracy (negative R^2) with high standard deviation thus we exclude it from the selection process of Step 2. We also observe that all the presented accuracies are fairly good for our purposes but this mainly happens because we begin our search with a small number of metrics ($|M| = 25$) and of models ($|G| = 8$).

Step 3 In step 3, we take the results shown in Figure 7.4 and we select the most suitable pair of model and set of metrics for our case as described in Section 5.3.3. In Figure 7.5, we present the runtimes of the models-set of metrics that are depicted in Figure 7.4.

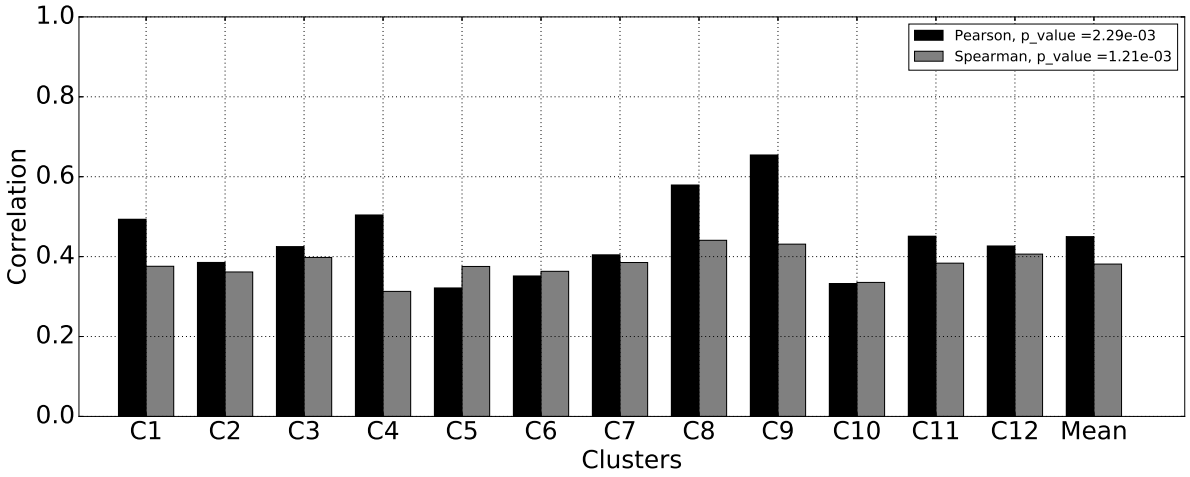
The runtime values are the averages of the runtimes of all time windows explained in the previous section. The runtime of *Gradient Boosting* is significantly higher than of the second-best models with difference larger than 800%. This increase of runtime is crucial for our simulations as the predictor is used at every monitoring event to estimate the current contention for each running VM. Therefore, we choose **Linear Regression** as our regression model which is the simplest among the other second-best models and introduces a lower computation overhead for the predictor runtime than that of *Gradient Boosting* . As for the number of metrics/regressors of the linear regression model, we end up with the 5 out of 6 selected metrics because



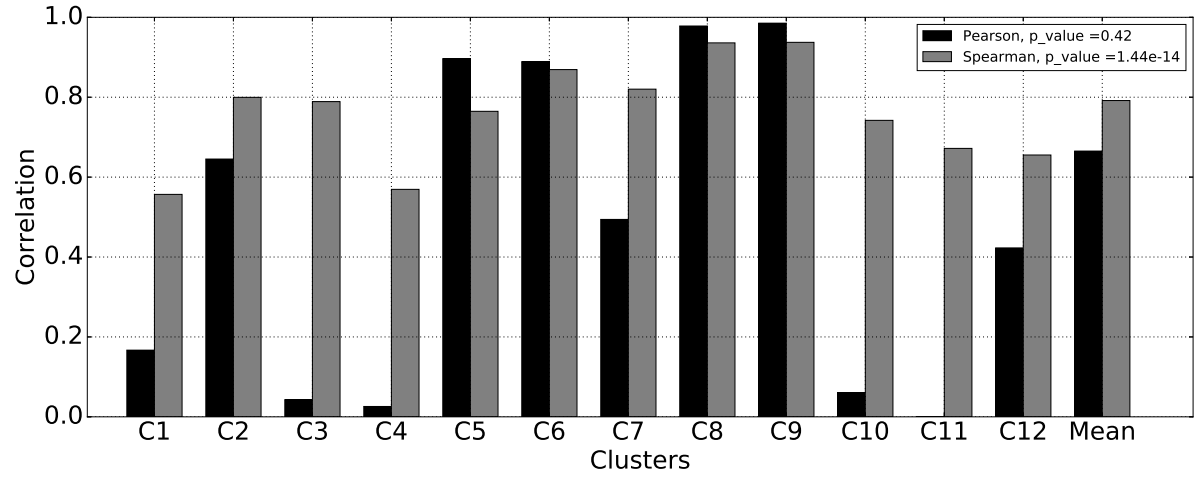
(a) CPU Ready auto-correlation.



(b) d_v^t - r_v^t correlation.



(c) d_c^t - r_v^t correlation.



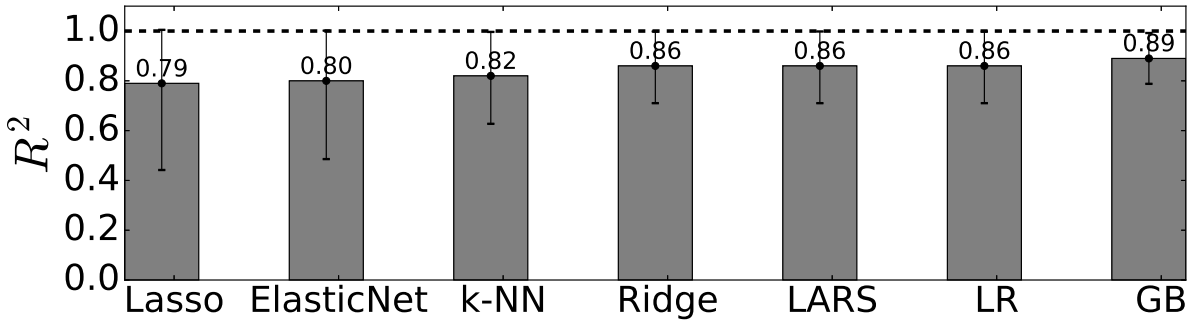


Figure 7.4: Accuracy of seven regression models.

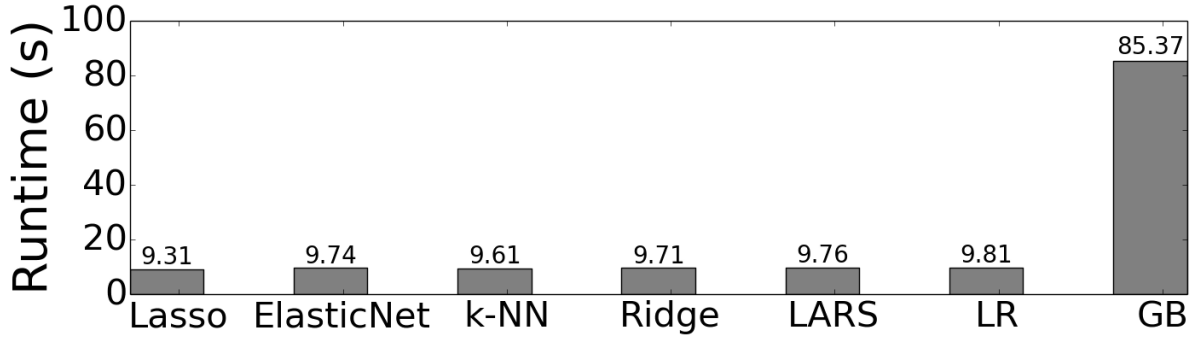


Figure 7.5: Runtimes of the seven regression models.

the remaining metric increases the model accuracy less than R^2_{impr} ($LR^5 = 0.862, LR^6 = 0.865$). In the end we choose as CPU contention predictor the model presented in Equation 5.1 in Section 5.

Verification of the model accuracy

To verify our chosen regression model, we run the simulator using the CPU-Contention predictor and compare the results with the real-world values collected by the system monitor. Figure 7.6 depicts this comparison by presenting the R^2 score for the percentage of data being tested ($\#data > 19$ million values). The results indicate a high accurate predictor of CPU-Contention for 99% of the values (0.93), albeit a lower accuracy for the total number of values (0.61).

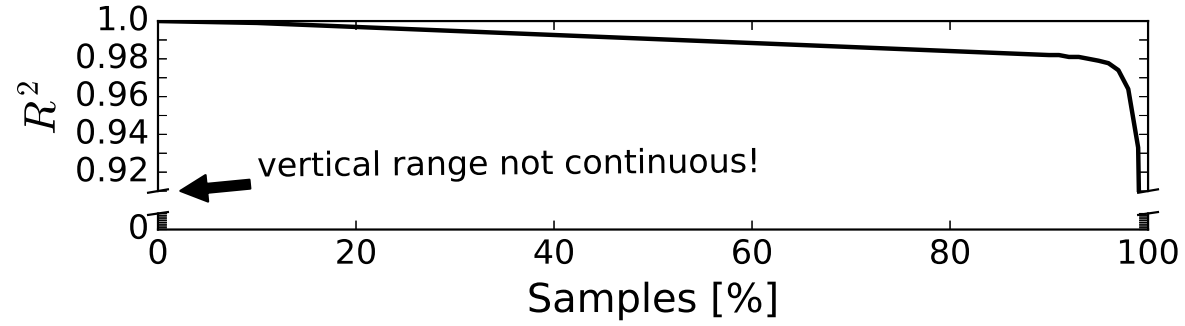


Figure 7.6: Accuracy of the tuned predictor.

7.4. Results for Portfolio Scheduling

In this section we present the results for portfolio scheduling. We present firstly the results using the real-world workload followed by the results using the synthetic workload. We conduct three sets of experiments for each workload. This section addresses the remaining research sub-questions presented in the beginning

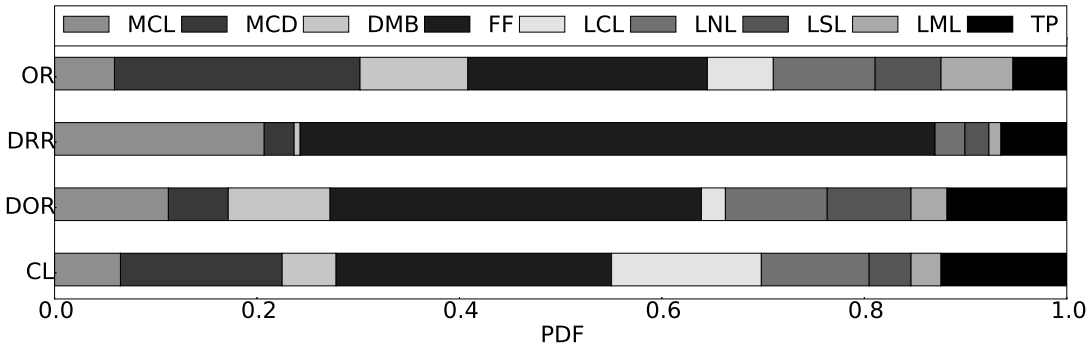


Figure 7.7: Policy Distribution for all utility functions - Read-world workload.

of this chapter.

Real-world workload

7.4.1. Distribution of Scheduling Policies for Real-world Workload

This subsection addresses the second research sub-question of this chapter about the scheduling policies selected by the portfolio scheduler. The scheduler decides on a new policy for every new vCluster need to be placed in the system. The scheduler keeps track of the decision taken for every vCluster and in the end evaluates these decisions. The intuition is that the portfolio scheduler under changing workloads and different performance goals (utility functions) will select different scheduling policies to place the input workload. We present the results of the different decisions taken by portfolio scheduler in Figure 7.7.

The vertical axis in Figure 7.7 indicates the utility function that is activated and the widths of boxes in a horizontal bar depict the fraction of times each scheduling policy was selected from the total set of decisions taken by the portfolio scheduler. The main findings are:

1. **All policies are selected for all utility functions we investigate.**
2. **Using different utility functions results in widely different distributions of selected policies.**
3. **Our new policies, MCL and MCD, are selected often for all the tested utility functions.**
4. **Simple policies, such as FF and TP, are surprisingly selected many times for all utility functions.**

The variability of the selected policies for every utility function not only validates our intuition of a scheduler that has to change decision plannings to different workload patterns but strongly points out the necessity of an adaptive and reactive resource management in cloud systems. For each optimization goal, we observe the different distribution of the selected policies implying that, given a specific vCluster, portfolio scheduler decides on placing the constituent VMs of the vCluster with different policy.

For the scheduling policies used the most, we observe that MCL and MCD belong to the top 3 of the selected policies occupying 18 – 30% of the selections for all utility functions. FF is surprisingly good and it is the most-selected policy for the DRR, DOR and CL utility functions, ranging from 25 – 60% of the selections. This explains why, despite not being designed to address complex performance and risk metrics, it is still much used by datacenter engineers.

To investigate more the distribution of the policy selections we measure the number of occurrences where the best-two policies of a selection have equal risk scores. At these cases, engineers can use either of the two policies for the current placement. We find that occurrences of the best-two policies of selections having even risk severity in the system range from 20-40% for the utility functions in Figure 7.7. Cloud engineers can leverage this finding in order to schedule by optimizing many goals at the same time. If portfolio scheduler is used as an advisor for VM-placement by cloud engineers, it will provide different suggestions on placing VMs subject to the defined goals; thus engineers, by combining and matching the scheduling suggestions of the best-two policies of every utility function, might select a placement which satisfies more than one optimization goal in the system. The satisfiability of multiple optimization goals by portfolio scheduler is left for future work.

For all utility functions though, we conclude that no scheduling policy is good enough to dominate the policy selection in portfolio scheduling due to the multi-dimensional scheduling problem.

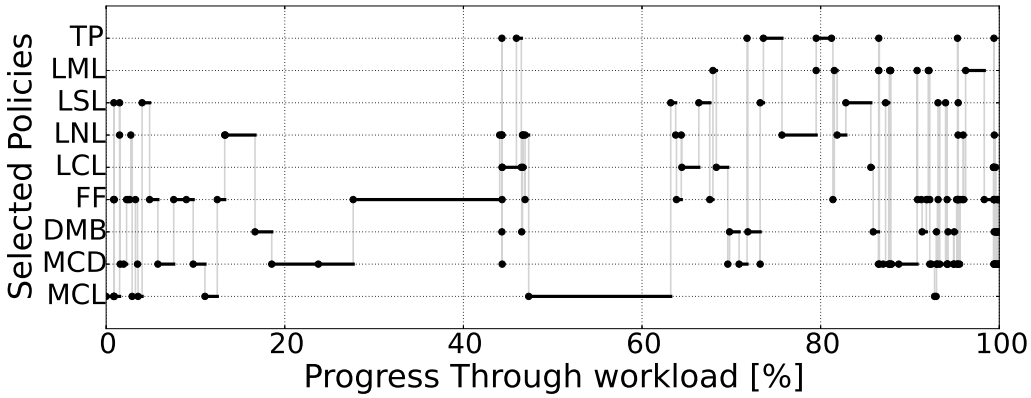


Figure 7.8: Selected Policies for Operational Risk - Real-world workload.

Appendix B presents the policy distribution of all utility functions we conduct experiments on portfolio scheduling using the real-world workload.

7.4.2. Utility functions for Real-world Workload

In this section, we investigate the performance of utility functions and we give some insights on the performance levels of the system while placing VMs with and without portfolio scheduling. This section addresses the third research subquestion of this chapter. We show and explain representative figures in this section while the rest of the results can be found in Appendices C-F.

Operational Risk

In Figure 7.8 we create a new type of figure for depicting the selected scheduling policies when a utility function is activated. This figure type visualizes the portfolio-scheduler decisions throughout the workload progress and can also be used to *explain portfolio scheduler's decisions to datacenter stakeholders.* The vertical axis is divided into parallel rows, each representing the progress (that is, non-/selection) of a different scheduling policy. For example, the new policies MCL and MCD occupy the two bottom-most rows. The horizontal axis depicts the progress through the workload: large dots placed on the rows indicate that a policy has been selected, whereas the thick horizontal bars emerging from the dots indicate the portion of the workload that is to be scheduled by the selected policy (wider bars indicate a higher portion). The last element of this graph is the set of vertical grey lines, which indicate transitions between consecutively selected policies. The main finding is:

Portfolio scheduler adapts to different workloads by switching scheduling policies.

Same policies are rarely selected consecutively pointing out the different workloads in the system and the scheduler's adaptability to work with them. In addition, this figure can provide an overview not only of portfolio scheduler but also workload-related features such as the preferred scheduling policies for a specific workload type.

Figure 7.9 depicts the evolution of the OR-utility-function value (score) for different scenarios (lower score is better). The curves indicating the portfolio-scheduler's OR scores represent the scheduler performance of the best selected policy (Best policy) and of the worst-performance policy (Worst policy). The remaining curves indicate the performance of individual policies when running solo and the system OR performance in read world (REPLAY). We observe that:

1. **Portfolio scheduler achieves the lowest risk of all scenarios.**
2. **Portfolio scheduling results in more than two times (2x) lower risk levels than individual policies.**
3. **Individual policies do not perform well in risk-aware scheduling.**
4. **Portfolio scheduler achieves 1.5x lower risk than the system risk in real-world.**

We observe that the performance of the portfolio when selecting the best policy achieves the lowest risk of all the other presented scenarios. We also show the risk performance of the worst-scenarios of portfolio to depict the fact that portfolio succeeds in avoiding bad performance and effectively alleviating the risk of

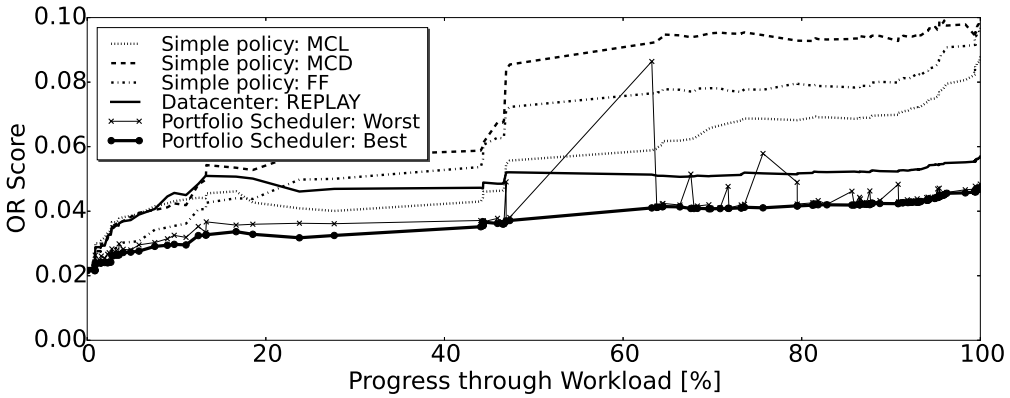


Figure 7.9: Evolution of OR over the entire workload - Real-world workload.

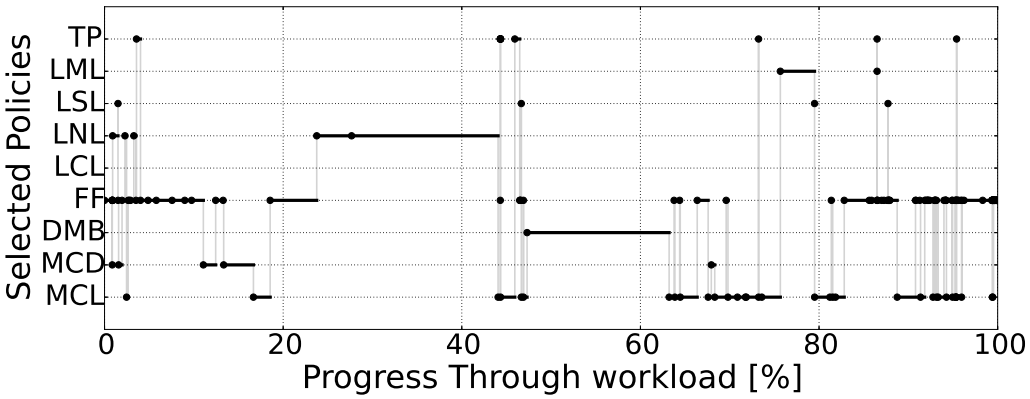


Figure 7.10: Selected Policies for Disaster Recovery Risk - Real-world workload.

scheduling policies performing badly. Although the differences in the performances between best and worst scenarios are not much, one should consider the following: Every new entry of vCluster in the system uses the previous VM placements of the best scenario as we try to optimize system performance, thus worst scores are evaluated given the previously optimized setting of the best scenario. In addition, the performance of utility functions is subject to the performance of the available policies in the portfolio. Therefore, by adding more policies will definitely show different performance levels. With this set of experiments, we verify that achieving lower risk bounds by using portfolio scheduling is possible.

To answer the third subquestion on the performance of constituent policies of portfolio when acting individually, we present OR scores of few policies and by comparison we result in more than two times (2x) risk levels when policies running alone. Therefore, even if the presented constituent policies in Figure 7.9 are selected often by the portfolio scheduler (see Figure 7.7), these individual policies do not perform well when running alone unless portfolio scheduler incorporates them to the portfolio set.

The curve of REPLAY in Figure 7.9 depicts the OR performance of the decisions taken in the real-world by datacenter engineers. This curve may be used by engineers to understand the risk improvement of the system when OR is the active utility function. We see that portfolio scheduler achieves 1.5x lower risk than the system risk in real-world.

Disaster Recoverability Risk

In Figures 7.10, 7.11 we present the performance of DRR metric using the same type of figures for OR performance. Figure 7.10 depicts the transitions of scheduling decisions by portfolio scheduler trying to optimize system performance considering the risk of workloads remaining unrecoverable after datacenter failure. We see that even though FF is selected many times (also mentioned in Section 7.4.1), portfolio scheduler must switch to other policies throughout workload to achieve low risk levels. In comparison to Figure 7.8, it is obvious that portfolio scheduler results in different selection patterns according to the activated utility function.

Figure 7.11 shows the evolution of DRR-utility function value for different scenarios. As a reminder, DRR

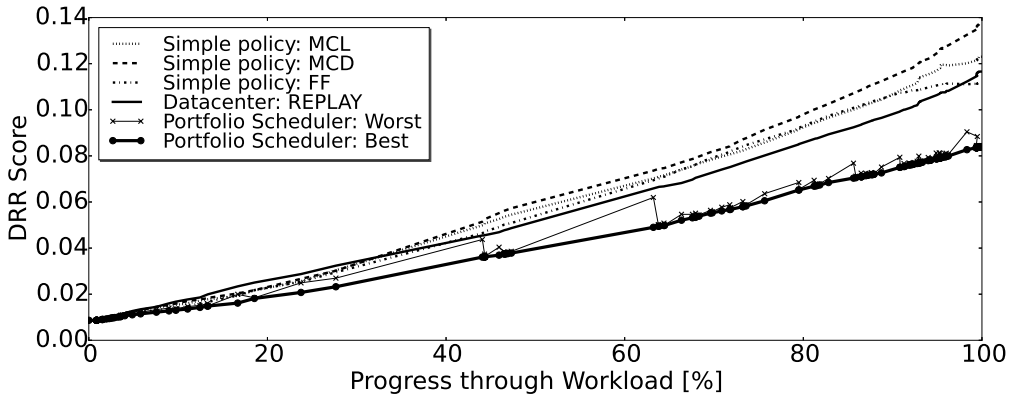


Figure 7.11: Evolution of DRR over the entire workload - Real-world workload.

values below 0.5 show that any workload is at risk of not be recovered while values above 0.5 indicate that part of workload cannot be absorbed when the host datacenter fails. The higher the DRR value above 0.5 is, the more workload is in jeopardy. Naturally, due to the fact that the real-world workload is recorded in a commercial datacenter system in which cloud operators focus on this particular risk, the DRR levels of the workload should be below the critical 0.5. We verify this in Figure 7.11 where DRR is much lower than 0.5. The main findings for DRR-utility function are:

1. **Portfolio scheduler achieves the lowest risk of all scenarios.**
2. **Portfolio scheduler can improve DRR levels up to 35% compared to risk levels of individual policies.**
3. **Portfolio scheduler achieves 40% lower risk levels than a well-managed commercial datacenter.**

We see that portfolio scheduler (Best policy) achieves the lowest risk scores than all the other depicted scenarios. Similarly to OR, the approach results in lower risks of violating SLAs with regards to Disaster Recovery guarantees. Furthermore, we see that individual policies achieve at least 35% worse risk levels than portfolio scheduler. This verifies the suitability of portfolio scheduler to place diverse workloads in the system by switching scheduling strategies.

Comparing DRR levels between portfolio scheduler's and real-world scenario's, we find that DRR can improve 40%. Thus, there is much room for improvement of real-world placements even the already existing focus of the operator on this risk.

Similarly, for both DOR and CL we also investigate the policy selection and utility function scores. The portfolio scheduler results are positive. Appendices C-F present the respective figures of these utility functions.

7.4.3. Performance of Optimization Policies for Real-world Workload

In this subsection, we present the performance of the selected policies for multiple utility functions (UFs). Table 7.2 summarizes the impact of activating one utility function on the scores of other useful utility functions. The results, when considered column-wise, indicate the influence that selecting a utility function (e.g., OR) has on the other utility functions. When considered row-wise, the results allow comparing for a utility function its scores when it is active and when other utility functions are active. If we optimize for the Disaster-Recovery-Risk (column "DRR" in Table 7.2), the OR score will increase up to 2x (risk 2x worse), compared to the case when OR is active. Conversely, activating OR increases DRR 1.5x. The main findings from this experiment are:

1. **DOR can combine successfully OR and DRR and achieves performance with tradeoffs.**
2. **Activating CL worsens all risk utility functions and DOR achieves lower CL levels than when CL is activated.**
3. **All utility functions have improved the risk levels in the real system (REPLAY).**

Active UF	OR	DRR	DOR 1-1	DOR 1-3	DOR 3-1	CL	REPLAY
OR score $\times 10^{-2}$	5	10	5	7	5	7	6
DRR score $\times 10^{-2}$	12	8	11	9	11	12	12
DOR score $\times 10^{-3}$	8	9	8	9	6	9	9
CL score $\times 10^{-5}$	4	16	3	13	10	4	5

Table 7.2: Maximum utility function (UF) scores for all UFs, for each activated UF. For DOR, we report results for three different settings (columns DOR $w_o - w_d$).

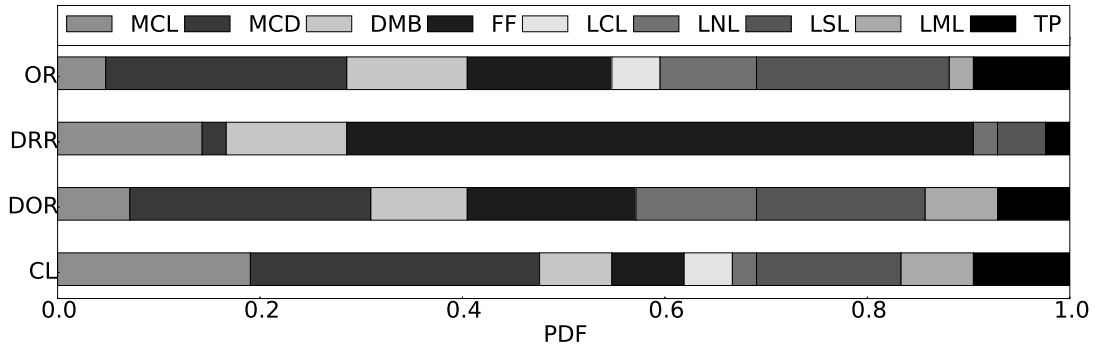


Figure 7.12: Policy Distribution for all utility functions - Synthetic workload.

DOR explores an important risk-management trade-off by combining our proposed utility functions OR and DRR. Thus, this metric can be used successfully to evaluate the system performance risk-wise. The results for DOR $w_o - w_d$ for the 1-1, 1-3, 3-1 settings, show cases in which different weights to risk scores can be set depending on the optimization targets and on the datacenter SLAs of a cloud provider. The risk levels are tuned according to the weights of the constituent risk types.

For CL which optimizes the system performance in terms of load balancing, we see that worsens all the risk scores. Therefore, portfolio scheduler should utilize the risk utility functions such as DOR to mitigate risks in the system and not try to reduce risks implicitly by activating CL. The load balancing of the real-world workload can be effectively achieved also by DOR which achieves CL performance even better than by activating CL. We denote that this finding applies to the real-world workload as we observe different behavior for the synthetic workload (see Section 7.4.6).

In Table 7.2 we also present the scores of the real placements in the system by the operator (REPLAY). As for the risk scores, we see that by using portfolio scheduling and optimizing the proposed risk utility functions, the risk levels have improved from those of the real placement (1.2x for OR and 1.3x for DRR). Similarly the load balancing in the system, CL score, is also improved when activating CL. These results point out that there is still much room for improvement on VM placement in a real production system.

Synthetic workload

In the remaining subsections, we present the results of the experiments using the synthetic workload described in Section 7.2.2. We conduct the same sets of experiments as with the real-world workload to have insights on how portfolio scheduling performs when system is under stress.

In all the experiments using the synthetic workload portfolio scheduler stops placing vClusters when the workload utilizes the 86% of the total memory capacity in the system. We call this point *system breaking point*. The system breaking point indicates the proportion of the workload deployed in the system as well as the proportion of the total system capacity that is provisioned to the workload. We consider that the system breaking point is the load at which the current virtualized infrastructure cannot host additional workload given our portfolio set of policies. In case new policies are introduced to portfolio scheduler, there might be different system breaking point.

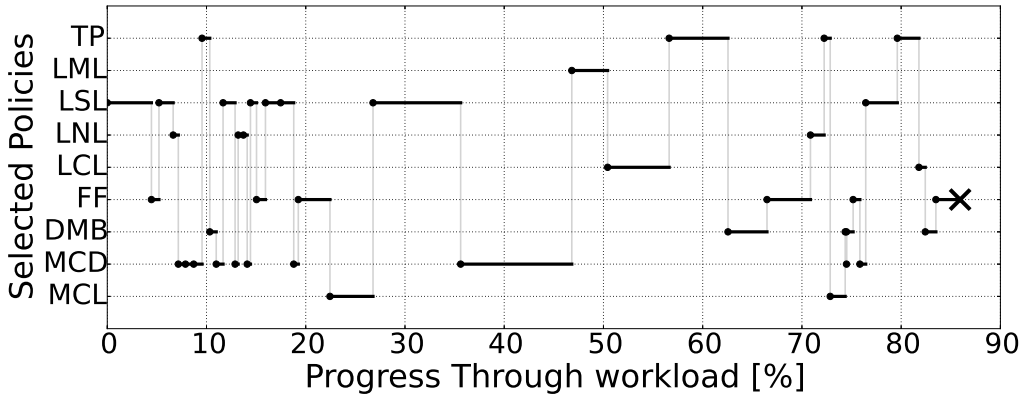


Figure 7.13: Selected Policies for Operational Risk - Synthetic workload. The symbol X shows the system breaking point.

7.4.4. Distribution of Scheduling Policies for Synthetic Workload

In Figure 7.12 we present the policy distribution of all the investigated utility functions when portfolio scheduling optimizing the placement of the synthetic workload. This figure address the second research subquestion. The main findings are:

1. **FF is widely used from 10% up to 60% of the total selections.**
2. **All utility functions except DRR need the diversity of the policies to achieve their optimizations.**

We see that the simple policy FF is surprisingly good and is selected mainly by DRR (60%). We point out again, as in Section 7.4.1, that even if this policy has no optimization goals yet achieves good system performance and that is why datacenter engineers still use it. For OR and CL, we see more dispersion on the policy selections. This implies that the corresponding risks of those utility functions need more the diversity of policies to achieve their goals. The number of occurrences where the best-two policies of a selection have equal risk scores is less than that in real-world experiments and ranges from 7%-14% of the total selections. Thus, the performance of different scheduling policies deviates when stressful workload arrives at the system.

Finally, all utility functions use all the available scheduling policies to reduce the most their respective risks, a fact that justifies the necessity of portfolio scheduling in business-critical workloads.

Appendix G presents the policy distribution of all utility functions we conduct experiments on portfolio scheduling using the synthetic workload.

7.4.5. Utility functions for Synthetic Workload

In this subsection we present the performance of the utility functions using portfolio scheduling while optimizing their goals. In this way we address the third research subquestion of this chapter. We present some figures in this subsection, the rest of the results can be found in Appendices H-K.

Operational Risk

In Figures 7.13 and 7.14 we depict the history of the policy selection for OR and its evolution respectively over the synthetic workload. In these figures we denote with an *X* the system breaking point at which the system stops placing vClusters in the infrastructure. The findings for the Operational Risk utility function are:

1. **Portfolio scheduler needs to incorporate policies related to contention and load balancing to minimize OR.**
2. **Portfolio scheduler achieves 10% lower risk levels than individual policies.**

In Figure 7.13 we see that the policies MCD and L*L are selected very often indicating that Operational Risk is strongly related to contention experienced by VMs and to load balancing in the system. We recommend then, that many scheduling policies considering contention and load balancing in the system should be included in the portfolio set when optimizing OR.

In Figure 7.14 we show that risk performance of individual policies might be 2x worse than portfolio scheduler, thus they are not recommended for running alone and scheduling heavy workloads. In contrary,

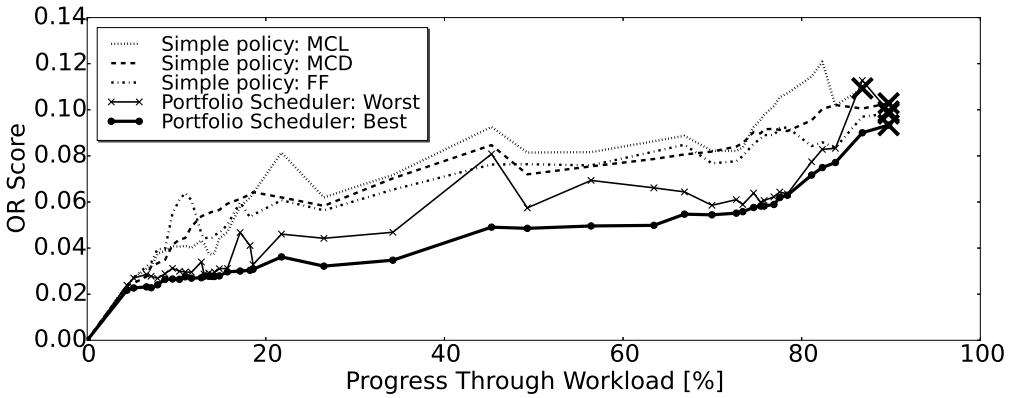


Figure 7.14: Evolution of OR over the entire workload - Synthetic workload. The symbol X shows the system breaking point.

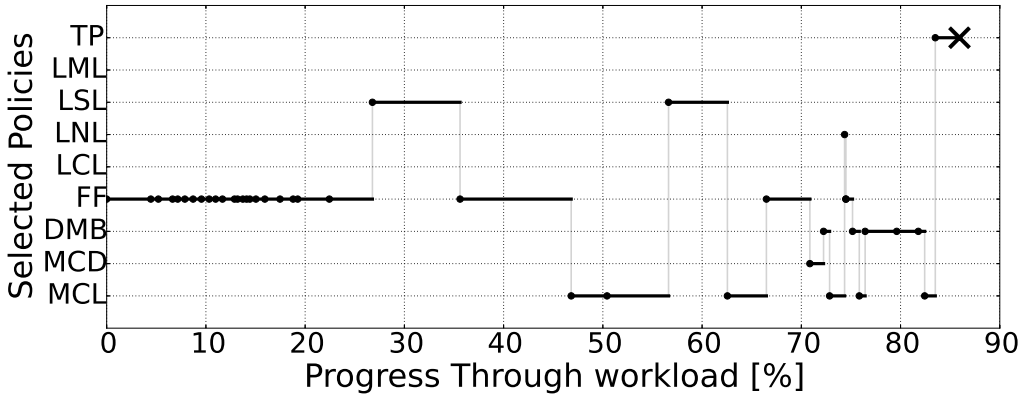


Figure 7.15: Selected Policies for Disaster Recovery Risk - Synthetic workload. The symbol X shows the system breaking point.

portfolio scheduler is always the best (lowest) and achieves more than 10% lower OR risks than individual policies at the system breaking point. We also compare the workload distribution presented in Figure 7.2 with the scores of the worst-scenarios of portfolio scheduler. We observe that when beefy vClusters including hundreds of VMs arrive at the system, there are big deviations between scores of worst and best policies. This relation points out that massive arrivals can harm (increase) more the OR risk levels in the system if the right policy is not used. Thus, by examining many policies to scheduling, a.k.a., portfolio scheduling, can provide engineers with valid suggestions on VM placement.

Disaster Recoverability Risk

In Figures 7.15 and 7.16 we present the results with respect to DRR utility function when scheduling the synthetic workload. The main findings are:

1. **FF is widely used by portfolio scheduler but cannot achieve the same performance alone.**
2. **When system load is high, individual policies achieve risk levels closer to portfolio scheduler.**

In Figure 7.15 we see that the selection of FF as the most promising policy by portfolio scheduler in terms of DRR is not only frequent but also consecutive. This result indicates the importance of incorporating simple policies to portfolio of policies since the contribution of such policies to risk-aware scheduling is confirmed to be significant. However, in the same figure, we see that portfolio scheduler needs also other policies to interrupt the consecutive scheduling of FF and refine bad decisions.

In Figure 7.16 we observe that the risk levels between the individual policies and portfolio scheduler are closer together when the system is lightly (<20%) and heavily loaded (>60%). In the former case, there is much free capacity to absorb the light workload regardless of the different placements by the policies while in the latter case, the scheduling policies have limited options to place VMs in the loaded system and thus, the risk levels are relatively close despite of the underlying scheduling strategy. Nonetheless, portfolio scheduler

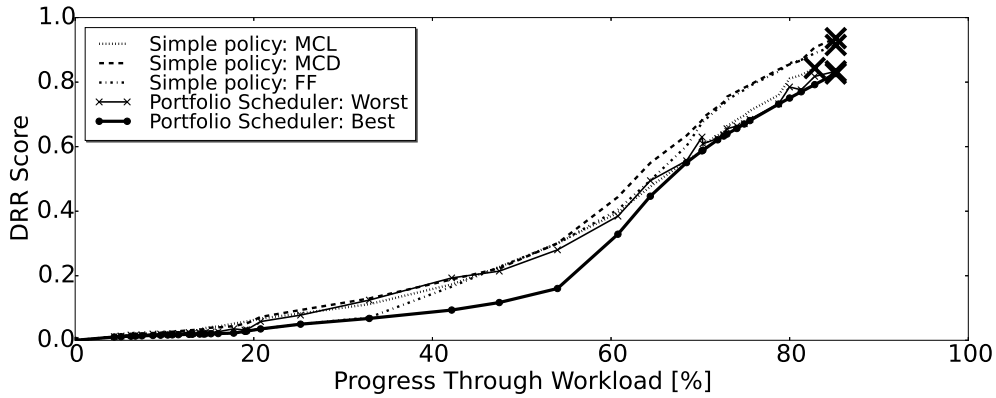


Figure 7.16: Evolution of DRR over the entire workload - Synthetic workload. The symbol X shows the system breaking point.

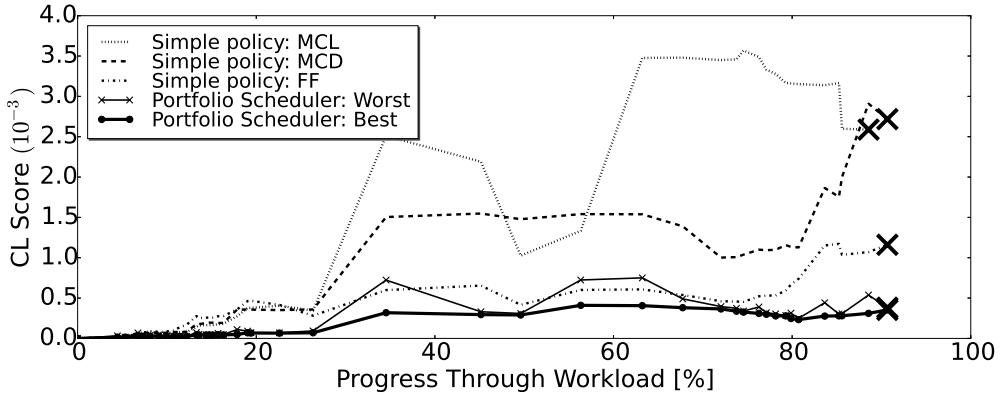


Figure 7.17: Evolution of CL over the entire workload - Synthetic workload. The symbol X shows the system breaking point.

achieves 10% lower DRR score than individual policies at the system breaking point. In the remaining load range (20%-60%), we see portfolio scheduler fully exploiting the advantage of policy selection and keeping risk levels much lower than individual policies.

Cluster Load score

In Figure 7.17 we present different behavior of portfolio scheduler compared to individual policies when optimizing CL for the synthetic workload.

1. **Portfolio scheduler achives 3.5x lower CL levels than individual policies.**
2. **Individual policies show good CL performance only when the system is not loaded enough.**

In contrast to the results about OR and DRR, portfolio scheduler achieves significantly better CL performance (3.5x) than the performance of individual policies when the system is highly loaded. The load balancing of clusters is a different optimization goal with respect to the other utility functions focusing on SLA risks. We see that portfolio scheduling results in better load balancing even if the policies considering the load balancing (L*L policies) are not mainly selected (<30% of total selections).

Furthermore, individual policies are good enough in terms of CL levels (yet worse than portfolio scheduler) only when the system load is low. When the workload occupies more than 30% of the system capacity, only portfolio scheduler maintains low CL levels.

In conclusion, portfolio scheduler achieves better performance than individual policies despite the optimization goal and can provide insights on system potentials when benchmarking the system risk levels. However, **the performance of portfolio scheduler is always subject to the performance of the policies incorporated in the portfolio set.** Therefore, if more policies are included and evaluated by the portfolio sched-

uler, it will consequently result in performance improvement since portfolio scheduler always favors the best out of the available policies. In the following section, we present our conclusions about the examined utility functions when used by portfolio scheduler to place the synthetic workload.

7.4.6. Performance of Optimization Policies for Synthetic Workload

Similarly to Table 7.2 for the real-world workload, we present the results for the synthetic workload in Table 7.3. The columns indicate the influence of selecting a utility function on the other utility functions. If we activate for the Disaster-Recovery-Risk (column 'DRR' in Table), then OR score will increase 1.3x, compared to the case when OR is active. Likewise, activating the combined score DOR with weights 3-1 of OR and DRR respectively decreases OR score 10% and keeps stable DRR.

Active UF	**OR**	**DRR**	**DOR 1-1**	**DOR 1-3**	**DOR 3-1**	**CL**
OR score $\times 10^{-2}$	9	12	11	12	8	12
DRR score $\times 10^{-2}$	89	83	82	81	83	90
DOR score $\times 10^{-3}$	49	47	46	64	26	51
CL score $\times 10^{-4}$	37	44	30	39	32	4

Table 7.3: Maximum utility function (UF) scores for all UFs, for each activated UF. For DOR, we report results for three different settings (columns DOR $w_o - w_d$).

An important result from this Table is the performance of CL score when CL is activated in portfolio scheduler. When CL is activated, the CL score is reduced at least 7.5x than the CL scores of all the other utility functions when activated. However, the corresponding OR and DRR scores when CL is active are increased.

The insights we have about the utility functions and their importance on datacenter scheduling and operation are:

1. **Combined utility functions can achieve lower risk levels than utility functions focusing on a specific risk type.**
2. **Portfolio scheduler should select placements by activating UFs on SLA violation risks regarding the system performance levels (load balancing).**
3. **DOR and CL scores can be used to benchmark system performance and reveal performance tradeoffs.**

A surprising finding of this work is that utility functions optimizing more than one goal might achieve better performance levels than utility functions specialized for one goal. In Table 7.3, we see that utility function DOR achieves lower risk levels of OR and of DRR when activating DOR 3-1 and DOR 1-3 respectively. The same weighted DOR functions also result in better CL levels than the levels of OR and DRR. Therefore, single optimization goals for portfolio scheduling might not achieve the best possible scores given a portfolio set. Due to the fact that performance in datacenter scheduling depends on multiple factors, if portfolio scheduler considers the optimization of many risk types at the same time, it might improve the overall system performance. The possible improvement is affected by the importance we give to risk types assigning them suitable weights.

Load balancing is a primary goal for datacenter engineers in order the system to be operational. For this reason, load balancing is a baseline standard to achieve further goals such as our focus on mitigating SLA violations. From the experimental results , we see that the two objectives , load balancing levels and system risk levels, can be achieved separately but they can be met together with some tradeoffs. Therefore, we can use portfolio scheduling to select policies with respect to risk performance levels in the system and then sorting the selections with similar risk performance according to the load balancing levels they achieve. As we see from the results, there are often a few policies with similar performance for every utility function (best and worst scores are close), thus we could use those policies to implement our proposal.

Utility functions can also be used separately from portfolio scheduler as performance indicators in system performance. CL and DOR scores can be utilized to benchmarking the systems with different workloads and have insights which workload types impose more risk on SLA violations. As a result, datacenter engineers can plan scheduling strategies which in turn will lead to the design of new scheduling policies addressing risks. Furthermore, datacenter stakeholders can agree on capacity plannings if future risk levels in the system are

inappropriate. Finally, the metrics (e.g., DOR 3-1, CL) can be used to daily monitoring the system and alarm engineers for irregular system behavior which may lead to SLA violations.

7.5. Summary

In this chapter, we present the results of the predictor selection and of the evaluation of portfolio scheduler managing risks in a multi-cluster multi-datacenter setting running business-critical workloads.

Through the prediction selection, we choose a linear regression model with metrics having been correlated with the CPU contention in business-critical workloads. The selected predictor is then used to estimate the contention of the workloads in the experimental evaluation of portfolio scheduler. The predictor has high accuracy for the business-critical workloads. The accuracy results of some of the regression models show no big differences. We explain this by the fact that we include many models related to linear regression (but with different constraints). We could have enriched the regression model set with more models but we already have a satisfying result for our problem.

We also present the results of the evaluation of portfolio scheduling. We use a real-world and a synthetic workload for the experiments. The evaluation validates our intuition that portfolio scheduler will deliver good (low) risk results since it evaluates the risk results of many scheduling policies. In addition, we show that the performance of portfolio scheduler is always better than the performance of individual policies running solo. It is also interesting to observe that the risk performance of placements by the engineers decisions taken in the real system is worse than the performance of portfolio scheduler. This points out the importance of the concept being used to large scale complex systems.

The implemented simulator shows that it is possible to implement a scalable model. This means that for business-critical workloads similar to the tested workloads of Solvinity, a.k.a. low arrival rate of customer applications, we can implement an online version of portfolio scheduler. For higher ingest rates and larger infrastructures, the implementation of the scheduler should be redesigned to be time-efficient. The implementation leaves room for code optimization that can reduce the runtime of the scheduler to be an effective online version.

8

Related Work

In this section, we survey related research from two main areas: portfolio scheduling and risk management in clouds. Overall, our study extends this entire body of work conceptually and technically, by proposing conceptual advances in the definition of risk and of policies to manage *risk*, by addressing the new problem of managing the *combined* operational and disaster-recovery risks, and by conducting experiments that are more comprehensive and reveal significantly more about portfolio scheduling operation than previous studies.

Portfolio scheduling originates from the field of finance [56]. In computational portfolio design [42], extensive work in the field of artificial intelligence has focused on the selection and application stages of the portfolio [39, 71].

Closest to our work, portfolio scheduling has been applied to academic [23] and commercial [65, 74] datacenters. In contrast to earlier work [23, 65], our work adds the focus on risk management, which as a new problem greatly extends the scope of portfolio scheduling in datacenters, and on long-running VMs, vs. the much shorter jobs considered by the others. In contrast to van Beek et al. [74], our work greatly extends the problem scope to *combined* risks, leads to many new advancements in scheduling, and is much more comprehensive in experimental analysis. Specifically, our work focuses not only on the resource overcommitment risk, but on a much larger family of risk problems (resource contention in general for operational risks, and *also* disaster-recovery and *combined* operational-disaster-recovery risks). This leads us to design new families of risk-related utility functions, new scheduling policies, and a new resource-contention predictor. Finally, our experimental work greatly extends the insights into the operation of portfolio scheduling offered by related work, in particular through the first analysis of when transitions between selected scheduling policies occur and of how the portfolio selections affect the utility functions over time.

Risk management in clouds includes an increasing body of work on SLAs between cloud customers and operators. Following more than a decade of evolution in the context of grids [22, 79], the state-of-the-art focuses on defining various system properties and SLA types [78], on negotiating and brokering SLAs [10, 25, 49], on monitoring for [20] and on assessing [24, 54] SLA-violations, and on selecting clouds to minimize them [35] and other aspects of SLA-lifecycle management [55]. Eyraud-Dubois et al. [31] present the notion of SLAs and consolidation and show that with dynamic bin-packing a theoretical global CPU utilization of 66% can be achieved. Our work also considers resource consolidation and the associated operational risk, but we additionally investigate the reliability risk of datacenter disaster and also combined risks. Moreover, we evaluate our schedulers with real-world datacenter workloads.

9

Conclusion and Future Work

For large enterprises and governments to host their business-critical workloads in virtualized datacenters, risk management needs to become a first-class citizen of datacenter-level resource management and scheduling. Toward this end, in this work we propose a new *risk-aware* portfolio-based scheduling architecture, which repeatedly and dynamically selects the scheduler that is estimated to minimize risk, among a set of schedulers considered in the portfolio.

Our is the first work to address scheduling that is aware of the following two types of risks: Operational Risk, which is the risk of not meeting the SLA performance requirements, and on Disaster Recovery Risk, which is the risk of not being able to absorbe the failure of an entire datacenter in a multi-datacenter operation. We further extend the state-of-the-art in portfolio scheduling through the following contributions: three new utility functions that respond to various concepts of risk, two new risk-aware scheduling policies, and a new method for selecting at runtime the risk-minimizing scheduling policy while taking into account the important but difficult to predict CPU-contention arising from multiple VMs sharing the same physical machine.

We show through trace-based simulations that use traces collected from a commercial multi-datacenter cloud operator and synthetic traces that our risk-aware portfolio scheduling compares favourably with all its constituent policies, and also with the real baseline of manually optimized operations as recorded by the cloud operator. We also show *why* each decision is taken.

In future work, we will extend our contention predictor and DRR to more resource types. We will also work on adoption, about which we know from previous studies [50], but also from our industry partners, that is a difficult process. We have created for this work graphs that explain scheduling decisions in detail, over time. For the future, we plan to use these and similar results as explanations given to datacenter engineers–understanding is believing.

A Literature Survey

Cloud refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. M.Armbrust et al. [7] mention the new aspects that Cloud Computing offers to the public: the illusion of infinite resources available on demand, the elimination of an up-front commitment by Cloud users and the ability to pay for use of computing resources on a short-term basis as needed.

B.P.Rimal et al. [64] present a taxonomy of cloud computing systems and survey existing cloud services developed by various projects world-wide (Google, Amazon AWS). *Cloud architecture* is the first categorization of cloud systems classifying the infrastructures on their ownership and in turn the accessibility of the resources (Public Cloud, Private Cloud, Hybrid Cloud). The *Virtualization Management* abstracts the coupling between the hardware and operating system , thus providing important advantages in sharing, manageability and isolation of resources. *Cloud services* is another categorization in the proposed taxonomy further split the services according to the level of service to cloud users. The services are classified to Infrastructure-as-a-Service (*IaaS*),Platform-as-a-Service (*PaaS*), Service-as-a-Service (*SaaS*) and Hardware-as-a-Service (*HaaS*). Cloud systems can also be categorized according to the *Fault Tolerance* and *Security* features they provide to cloud users. Further classes of identification depict the provided *Load balancing* techniques and Interoperability issues to allow applications to be ported between clouds.

The resource management in clouds refers to the management of the available resources. B.Jennings et al. [47] mention the constituent fields of resource management such as the *resource utilization(demand) estimation(profiling)* and *resource pricing and profit maximization*. An essential part of resource management, however, is the *workload management* and the ***scheduling*** of the resources.

In the remainder of this survey, we provide latest developments on different topics related to scheduling in clouds. *Metrics* are widely used for evaluating and monitoring cloud systems, *predictor models* are used for profiling and estimating various cloud issues, *schedulers* are used for the main part of scheduling and *simulation tools* explore different cloud techniques to be applied to real systems.

A.1. Metrics

System behavior must be monitored in order engineers to supervise the system and take actions when needed. Metrics measure the non-functional system requirements which depict the way system operates. Examples of such non-functional requirements in cloud is the performance of cloud services and the elasticity of cloud system.

Most metrics can be categorized according to the resource type they measure. Thus, there are metrics measuring utilization of CPU, memory, network and storage. Those metrics can further be classified by the entity whose the resource type is measured. For example, different CPU metrics may depict the CPU utilization of server and of cluster. There are metrics measuring the performance on host-level, on cluster-level and on system-level. In total, hundreds of metrics can be used to monitor cloud systems according to needs. Cloud providers should selectively choose the monitored metrics for the system since the retrieval of abundant metrics in short time periods causes overhead and possible performance bottleneck in the system.

Contemporary commercial monitoring tools support the collection of metrics and are able to produce further metric values with statistical techniques, i.e., CPU workload in near future. VMWare's [4] , Citrix's [2], Microsoft's [3] and Amazon-AWS's [1] cloud monitors are widely used currently in the market. There are also enterprise-class monitoring tools which are open-source such as Zabbix [5].

In the remainder of this section, we present some references of the state-of-the-art metrics being used by monitoring tools to supervising systems.

Z.Li et.al [53] present a survey on cloud metrics after Systematic Literature Review. They collect the de-facto metrics adopted in the existing cloud services evaluation works and categorize them. They identify three categories according to cloud service features and these categories in turn are divided into more. The first category is "Performance metrics" encompassing metrics referring to performance evaluation of the system. Each performance metric is divided into parts: the physical property part and the capacity part, for example Communication Latency or Storage reliability. As a consequence of this partition, we may have numerous metrics if we match a physical property and a capacity part. The authors present metrics for every common resource type in clouds and for every capacity part they identify in their survey.

The second category of metrics is "Economics". This part comprise cost-related metrics used in clouds measuring cost effectiveness and cost efficiency.

The last category encompasses "Security" metrics. According to the authors, this category is relatively

new and there is a research gap on quantifying security issues in cloud.

Ang Li et. al [52] introduce CloudCmp, a systematic comparator of the performance and cost of cloud providers. The framework utilizes cloud metrics which are claimed to be suitable for using them in cloud benchmarks. The authors provide results of benchmarks when running in systems of public cloud providers.

The metrics are classified according to features common in cloud services. Elastic computing set encompasses metrics about the scaling latency and the costs of scaling up or down applications. This set also includes the benchmark runtime as a metric since the completion time of a benchmark is related to the available resources being scaled through workload.

The second set of metrics, the set for persistent storage, measures the ability of application instances to access and share application data. Metrics such as the consistency time to synchronize data and response times of storage operations are included in the mentioned set.

Finally, the metrics set about network is further divided into metrics measuring the communication among instances and the communication of instances with end-users.

The authors conclude that there are performance and cost variations across providers, thus customers are advised to outsource their application instances according to their needs.

V.C. Emeakaroha et al. [29] present the LoM2HiS framework which bridges the gap between the monitored metrics and SLA parameters. Since SLA metrics depict high-level performance of applications, the authors attempt an automatic approach to translate those metrics to low-level resource metrics which are directly monitored. The mapping between the two domains is sometimes straightforward, e.g., disk space metric represents SLA metrics of storage, or more complex such as in case of response time (SLA) and the related resource metrics of bandwidth and data transfers. The proposed framework using the automatic mapping monitors high-level info, i.e., SLA objectives and violations, by collecting low-level monitored values.

In the work of N.Herbst et.al [41], the authors propose new metrics together with measurement approaches about non-functional properties of cloud systems. They argue that traditional performance metrics, such as throughput and response time, are not sufficient alone to benchmark current systems and inform cloud users about complex cloud properties affecting QoS.

Elasticity is a cloud system property about accommodation of large variations in the amounts of services requested in a system. In this work, many elasticity-related metrics are proposed to test systems under widely varying workloads.

Performance Isolation is a system property about the interference between tenants of shared cloud systems. This work introduces related metrics focusing on the level of interference that tenants might have in a system leading to performance issues.

Availability is also addressed in this work which proposes metrics measuring the un-/availability of the resources, an issue having high impact on th reputation of cloud providers.

Finally, the authors propose *Operational Risk*-related metrics measuring the compliance of the performance of running applications against their agreed performance described in SLAs.

A.2. Predictor Models

In statistical modeling, *Regression Analysis* is a statistical process for estimating the relationships among variables. More specifically, to investigate the relationship between a dependent variable and one or more independent variables. Regression analysis is widely used for prediction and forecasting. In this case, the independent variables are also called *regressors* and the dependent variable *response*. In Figure A.1 we see the aforementioned parts of the process.

The analysis is used firstly to understand which among independent variables are related to the *response* and secondly to explore the forms of these relationships. The former part of the process identifies the *regressors* and the second part selects the regression model as depicted in Figure A.1.

The *regressors* are independent variables which are related to the *response*. The relationship is commonly evaluated by correlation metrics, each one of them presents different type of relationship. After identifying the *regressors* set, a regression model should be selected which will "predict better" the response according to regressors values. The regression model fits the regressors data into functions whose output is the *response*. The goodness of fitness can be measured by metrics such as the coefficient of determination (R^2). The coef-

ficient of determination indicates the proportion of the variance of the *response* that is predictable from the *regressors*.

Figure A.1: Regression Analysis elements.

S.Islam et al. [46] present empirical prediction models to estimate the resource utilizations of applications. They focus on estimating the needs of CPU resources for an application. The input as well as the output of the predictor is the CPU utilization for an application. The former derives from historical data of CPU utilization while the latter is the estimated value from the prediction model. They investigate two different learning methods for predictors, the neural networks and the linear regression approach.

By evaluating the accuracy of the predictors with many metrics (R^2, RMSE, PRED), they conclude that the neural network model using sliding window for the input of the predictor can be really effective on forecasting the resource utilization in the cloud.

T.Vinh at al. [27] propose a predictor estimating power consumption of cloud applications in order to be used in energy-aware scheduling in clouds. Green scheduling is relatively new in cloud research and addresses the energy problem of cloud systems since power consumption of servers is significant even when running idle. The proposed energy predictor is feasible to run in dynamic real-time settings.

The predictor uses neural network techniques to forecast the CPU workload demand of applications and along with their power consumption profiles, it estimates how much energy an application will consume. Using this knowledge, the power consumption estimations can further be used at hosting applications together and saving up energy by turning on and off servers.

J.Mars et al. [57] introduce a method named *Bubble-up* which uses predictions of performance degradations of applications. They focus on the degradations resulting from contention for shared resources in the memory subsystem.

The predictor process is built on two steps. Firstly, they profile an application in terms of memory load and by using a secondary co-hosted application, the *bubble*, they identify the impact on the performance of the former application when sharing resources with the latter. Thus, using the bubble memory loads as standard for the profiling, they record the loss of QoS of the applications when sharing memory. They further quantify the recorded degradations from the contention for memory resources and match applications to be co-hosted that have the least impact on memory contention.

T.Dwyer et al. [28] present a method of creating predictors of performance degradations of applications. They focus on the degradation resulting from contention for sharing resources in multicore chips. The input of the predictor is a set of hardware counters whose values are recorded throughout the application runtime. The input values are obtained by the applications co-hosted in the same multicore chip. The predicted value - the output of the predictor - is the estimated runtime of the applications.

The authors investigate many regression models to identify the most accurate model for the problem. In

the end, they select the model with the least average error.

P.Xiong et al. [81] introduce an automated model-driven framework which identifies the system metrics that are most predictive of application performance. The predictive set of metrics might change during runtime, thus framework is capable of potential shifts.

The framework monitors hundreds of metrics on host and application level and selects those which are highly correlated to the response time of applications for a short time period. Then, the regression model is selected which has the highest accuracy of predictions for a specified time window. In the end, the set of correlated metrics and the regression model with the highest accuracy and low computation overhead is chosen as the predictor.

The prediction model might change through runtime since the relationship between performance and system resources is not always the same. The changing point is defined when the prediction errors between two sliding windows are significantly different and do not come from the same distribution (result of null hypothesis).

A.3. Schedulers

Scheduling is the process of provisioning resources and assigning jobs to these resources [73]. The resources that are available to be provisioned for customer services in the cloud are commonly the CPU, the memory, network and storage I/O. The jobs arriving in the cloud and have to be assigned to the physical resources in order to be executed vary according to workload types. The workloads may comprise sequential, parallel jobs or even mixtures. Every workload type has different characteristics on the requested resources need to be provisioned. Scheduling considers these characteristics aiming at co-scheduling workload types efficiently.

Scheduling workloads can be mathematically formulated by the Vector Bin Packing [16] problem where we consider multi-dimensional bins sized according to the resource amounts of CPU,memory, network and storage. A solution to that problem depicts an assignment of jobs with specific requests to the bins they fit. An optimal solution fits the jobs to bins, i.e., resources, in such way that is optimal under one or more utility functions.

Scheduling workloads in clouds has also to consider the *Quality of Service* (QoS). QoS indicate some constraints of workloads that are related to the performance of the workload execution. Such constraints can be deadline constraints of the execution times, availability of resources, latency of components of the physical machines. The most required constraints of for the workloads to run in clouds are guaranteed through agreements between cloud service providers and cloud customers. These agreements are defined as *Service Level Agreements* (SLAs).

Solving the Vector Bin Packing problem proves to be very complex. Woeginger [80] shows that it is APX-hard meaning that there is no polynomial-time approximation scheme (PTAS) for this problem. Therefore different techniques are used to address the scheduling problem [70], [77], [59], [38]. *Scheduling policies*, or scheduling algorithms, describe a process in which assignments of jobs take place under specific heuristic. Later in this section, we present related work about scheduling policies. A *meta-scheduling* approach to address the optimization of multi-objective scheduling is the portfolio scheduling.

Portfolio scheduling originates from the field of finance, started by the seminal work of H.Markowitz [56] and later refined by R.Merton [58]. F.Black et al. [12] later added the reflection step in the process of portfolio scheduler. The concept of *computational portfolio design* is introduced by B.A.Huberman [42] who adapts portfolio scheduling as an economics approach to computation problems that involve variability in performance. He claims that by constructing portfolios of heuristic algorithms, good solutions can be found in reasonable time. C.Gomes et al. [40] present later that by creating portfolios with different instances of stochastic algorithms, there is a significant improvement on hard satisfiability problems. In computational portfolio design, extensive work in the field of artificial intelligence has focused on the selection and application stages of the portfolio [15, 39, 71]. Extensive work has also been conducted on speeding up the selection step of portfolio scheduling [23, 36, 37]. Portfolio scheduling applied to datacenters is relatively new approach at the multi-objective scheduling problem in datacenters.

K.Deng et al. [23] present a model for the exploration of portfolio scheduling. Portfolio set of policies in the scheduler comprises tens of scheduling policies being evaluated to selecting the most promising policy for a specific workload.

The authors introduce an online algorithm for time- constrained policy selection, which aims at increasing the change of selecting the best policy under time limits. This method addresses the reflection step in portfolio scheduling. During the portfolio scheduler's runtime, the policies are divided into best-performing policies (Smart policies) and poor-performing policies (poor policies). The former include the policies that are more possible to lead to good solutions on time and the latter the opposite. However, as the authors argue, the poor policies are also needed to optimizing the utility functions.

O.Shai et al. [65] propose a portfolio scheduler approach for scheduling batch jobs in systems. The optimization goal is the maximization of the total number of running jobs. The authors, after verifying that single policies cannot improve performance by running alone, introduce a meta-heuristic approach in which many policies are used for scheduling and the best policy in terms of balanced resource usage is selected. The result is the improvement of resource-utilization levels in the servers by a significant factor.

V.van Beek et al. [74] present a self-expressive framework in resource manaagement of datacenters. The framework incorporates portfolio scheduling approach in the placement decisions. The components of the framework proactively gather system information that will increase their self-aware and self-expressive capabilities. The information is then used by portfolio scheduler to efficiently select the policy that best fits the current workload. In this work, portfolio scheduler is also informed about required placements (anti-/affinity rules) since some application types require their instances to be hosted or not together. The experimental results show different distributions of selected policies according to the activated utility function.

Scheduling policies

A.Beloglazov et al. [11] present resource allocation heuristics considering the energy efficiency in the system. The proposed policies attempt to alleviate high CPU utilization of servers which in turn lead to increasing power consumption. The policies estimate the power consumption of the servers and decide on migrating VMs according to their utilization patterns. The result is less energy consumption which reduces the SLA violations in terms of CPU utilization levels of the workload.

A.J.Ferrer et al. [35] present an approach to cloud service provisioning considering service as scheduling unit described with lists of requested resources and agreed SLAs on service performance. They introduce elasticity policies considering performance risks such as service costs, reliability of service and energy efficiency. The policies decide on scaling up or down (reactively or proactively) the service resources according to gain parameters. The parameters evaluate the periodic changes of system load, the future service load and the system service rate over time.

Y.Chen et al. [18] propose scheduling strategies based on predictions of future resource demands and feedback control of system execution. The predictions of requested resources are used to a proactive algorithm using the estimations to solve a given optimization problem on placement. The second proposed algorithm is a reactive solution, which uses periodic feedback to achieve energy efficiency on servers. The authors argue that a hybrid approach combining the mentioned strategies results in efficient server provisioning considering server configurations and workload behavior.

S.Blagodurov et al. [14] present a scheduling algorithm for multi-objective job placement. After categorizing the application jobs according to the contention level, communication loads and their power consumption, the algorithm finds the best placement to balance the degrees of collocation with other performance-related factors. The algorithm works online, thus they propose strategies to limit the runtime of the algorithm by reducing the search space of the solutions.

A.4. Simulation Tools

Cloud simulator helps to model various kinds of cloud application by creating *Datacentre*, *Virtual Machine* and many *Utilities* which can be added to configure it, thus making it very easy to analyze.

The system model of a simulator should be detailed depending on the goal of the simulator. A simulator specific for energy savings should have a detailed energy model but might not have fully implemented all the different state-of-the-art memory architectures. On the other hand, simulator specialized in resource contention should have options for memory architectures in order to investigate concepts about contention on different simulated systems. In the following paragraphs, we mention a survey on cloud simulators and describe the initial simulator (DGSim) we refine later for our work.

A.Ahmed et al. [6] present a survey on cloud simulators and compare their features. The survey covers most of the popular cloud simulators such as CloudSim and GreenCloud. The analysis discusses the underlying platform the simulators operate which is important for the performance and the usability of the simulators. The authors also describe for each simulator the available communication models and the energy models if supported. Finally, they refer to the federation policy which considers the inter-networking of applications in different geographical locations and is supported by only but a few simulators.

A.Iosup et al. [45] introduce the grid simulator *DGSim* which is used later by K.Deng et al. [23]. *DGSim* is a simulation framework evaluating resource management solutions. It supports many workload models such as individual jobs and workflows , and iter-/intra-operation models to depict possible architectural alternatives. The framework also supports workload generator to create synthetic workloads according to given characteristics from the user or from a real workload.

The *DGSim* simulator is refined lated by V.van Beek [73] which in turn we use in our work and incorporate the CPU contention predictor.

B Policy Distribution - RT

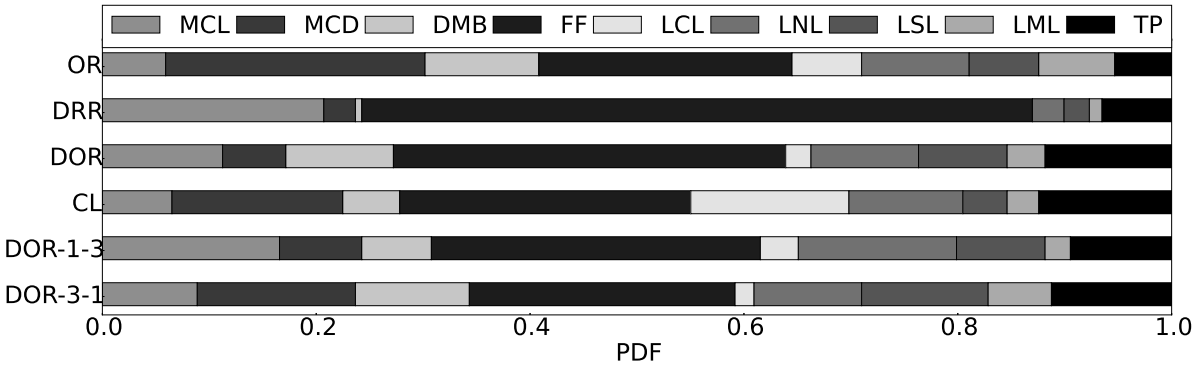


Figure B.1: Policy Distribution for all utility functions including the weighted DORs - Real-world workload.

C Utility Function: DOR-1-1 - RT

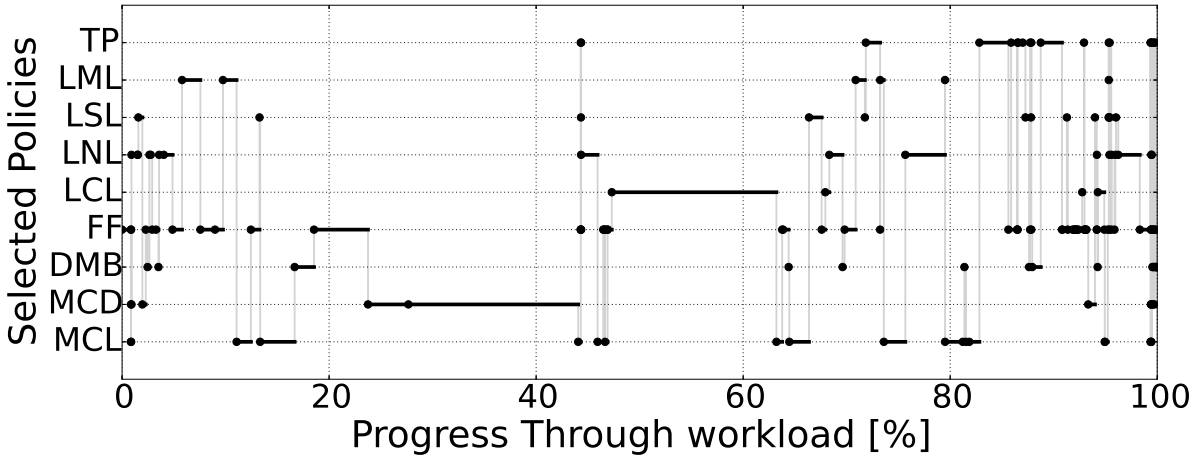


Figure C.1: Selected Policies for DOR with weights $(w_o, w_d) = (1,1)$- Real-world workload.

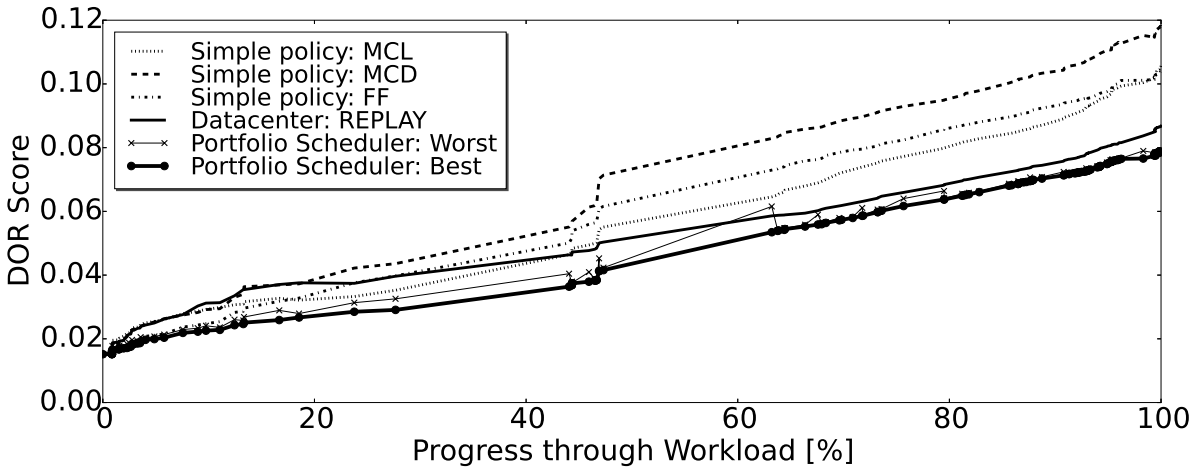


Figure C.2: Evolution of DOR with weights $(w_o, w_d) = (1,1)$ over the entire workload - Real-world workload.

D Utility Function: DOR-1-3 - RT

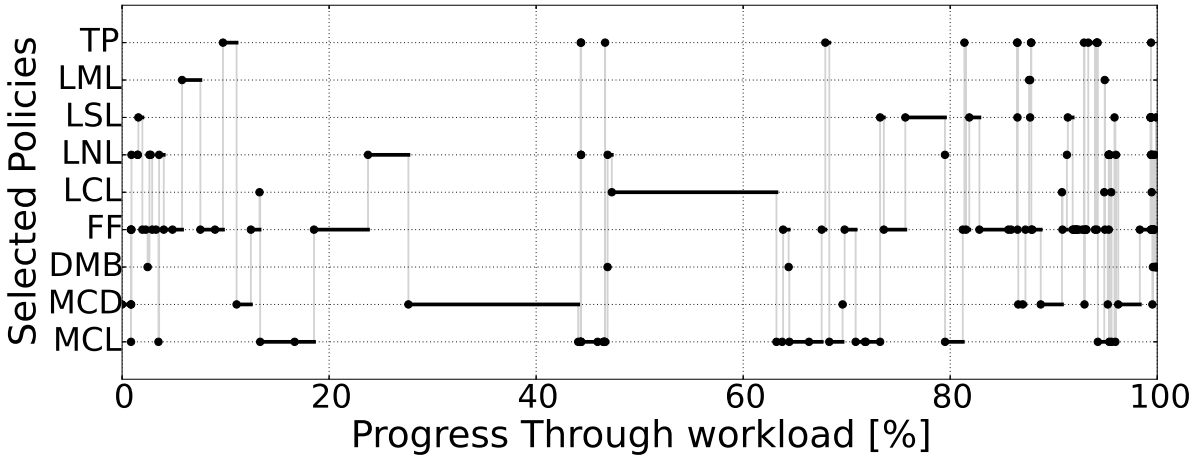


Figure D.1: Selected Policies for DOR with weights $(w_o, w_d) = (1,3)$ - Real-world workload.

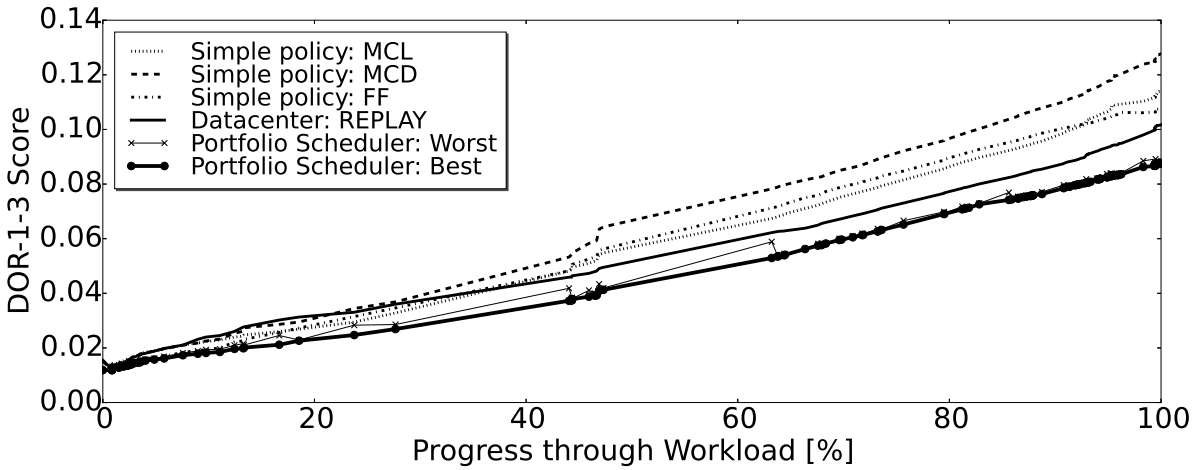


Figure D.2: Evolution of DOR with weights $(w_o, w_d) = (1,3)$ over the entire workload - Real-world workload.

E Utility Function: DOR-3-1 - RT

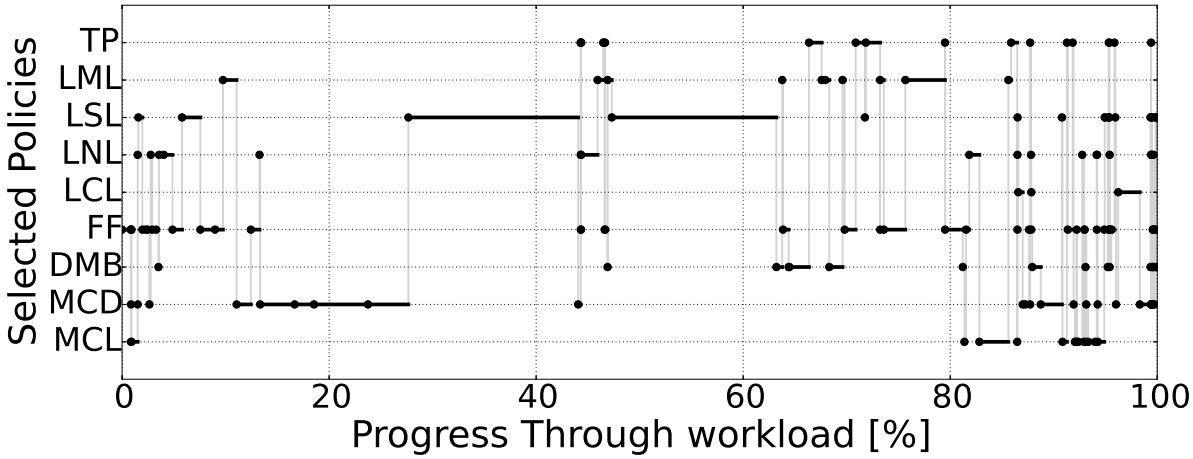


Figure E.1: Selected Policies for DOR with weights $(w_o, w_d) = (3,1)$ - Real-world workload.

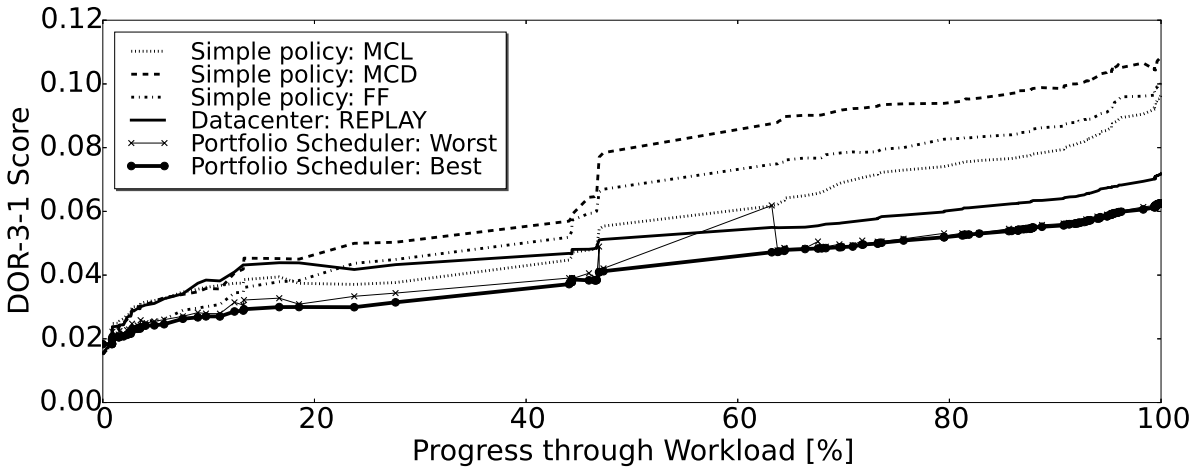


Figure E.2: Evolution of DOR with weights $(w_o, w_d) = (3,1)$ over the entire workload - Real-world workload.

F Utility Function: CL - RT

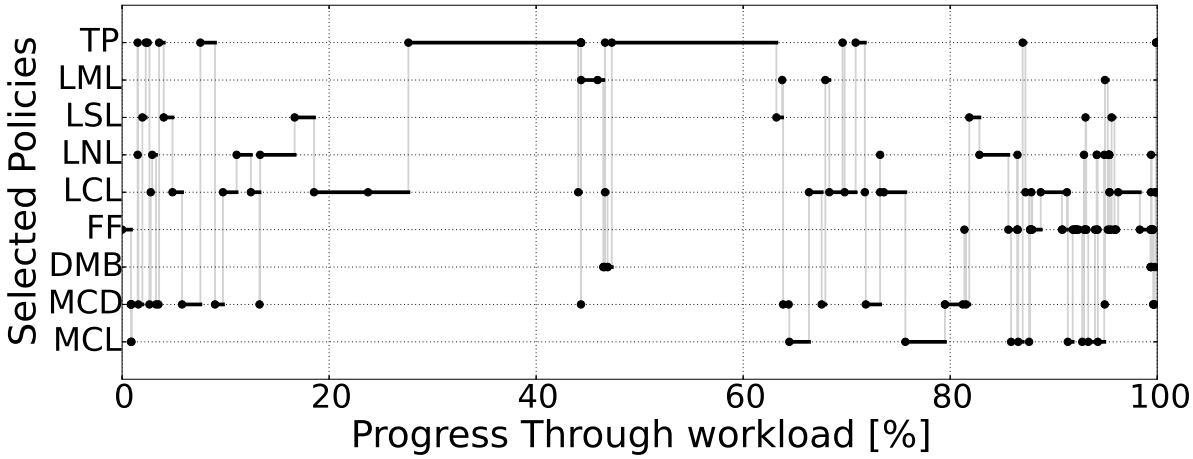


Figure F.1: Selected Policies for CL - Real-world workload.

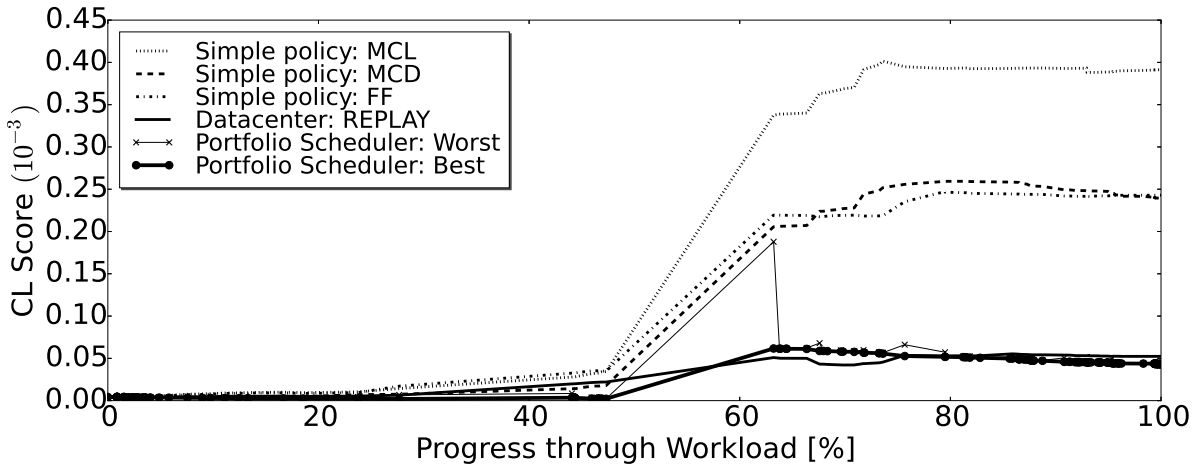


Figure F.2: Evolution of CL over the entire workload - Real-world workload.

G Policy Distribution - ST

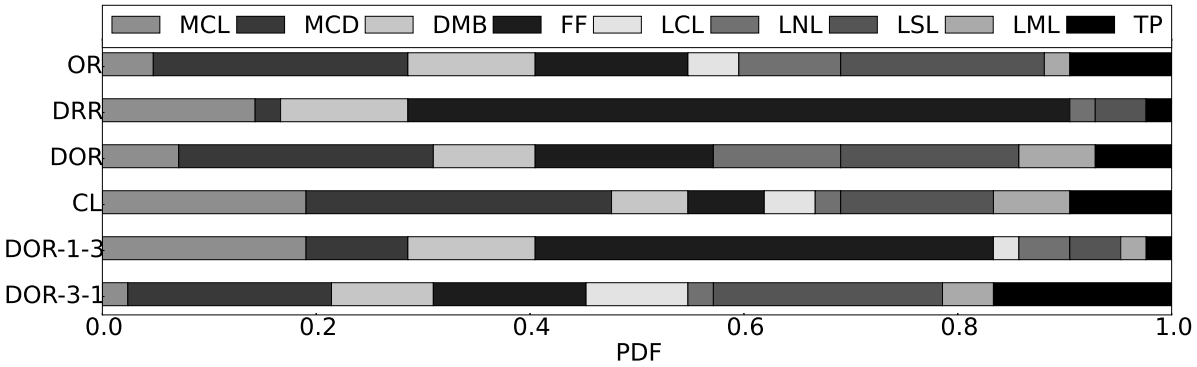


Figure G.1: Policy Distribution for all utility functions including the weighted DORs - Synthetic workload.

H Utility Function: DOR-1-1 - ST

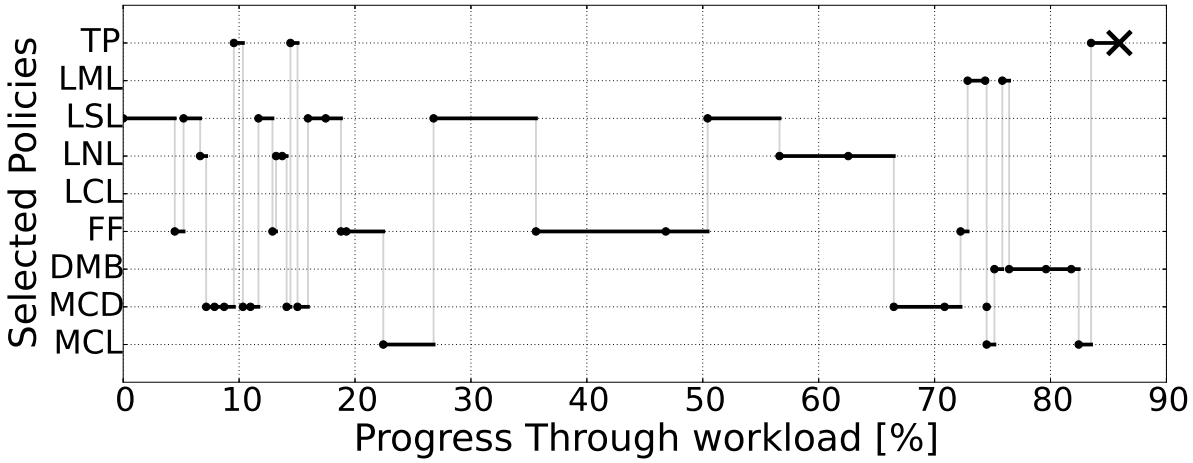


Figure H.1: Selected Policies for DOR with weights $(w_o, w_d) = (1,1)$- Synthetic workload. The symbol X shows the system breaking point.

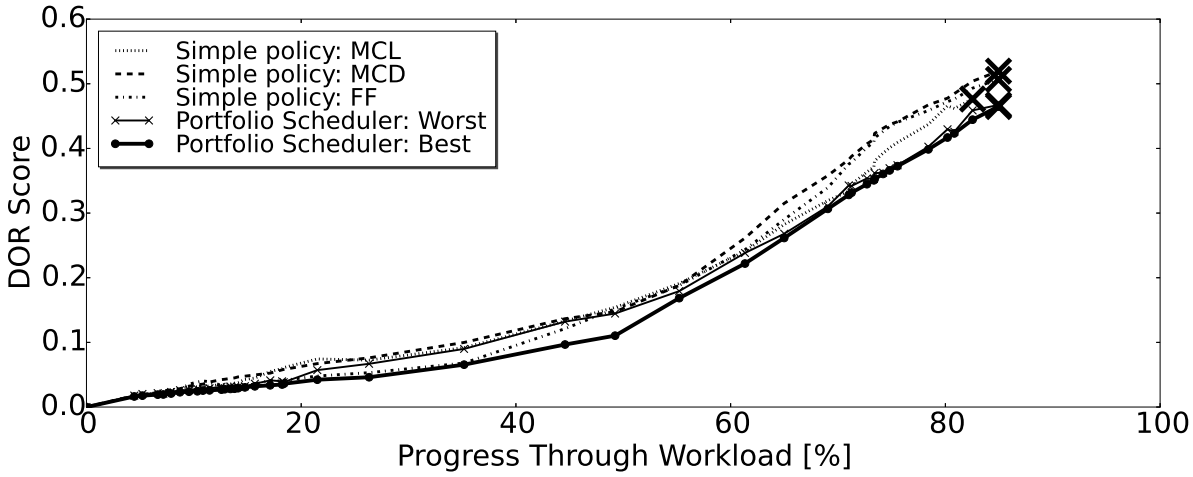


Figure H.2: Evolution of DOR with weights $(w_o, w_d) = (1,1)$ over the entire workload - Synthetic workload. The symbol X shows the system breaking point.

I Utility Function: DOR-1-3 - ST

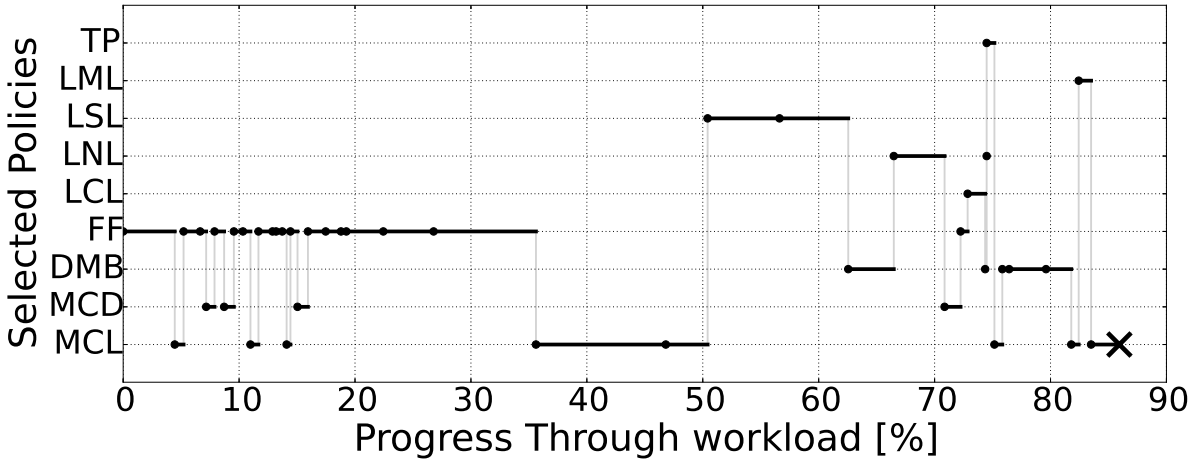


Figure I.1: Selected Policies for DOR with weights $(w_o, w_d) = (1,3)$ - Synthetic workload. The symbol X shows the system breaking point.

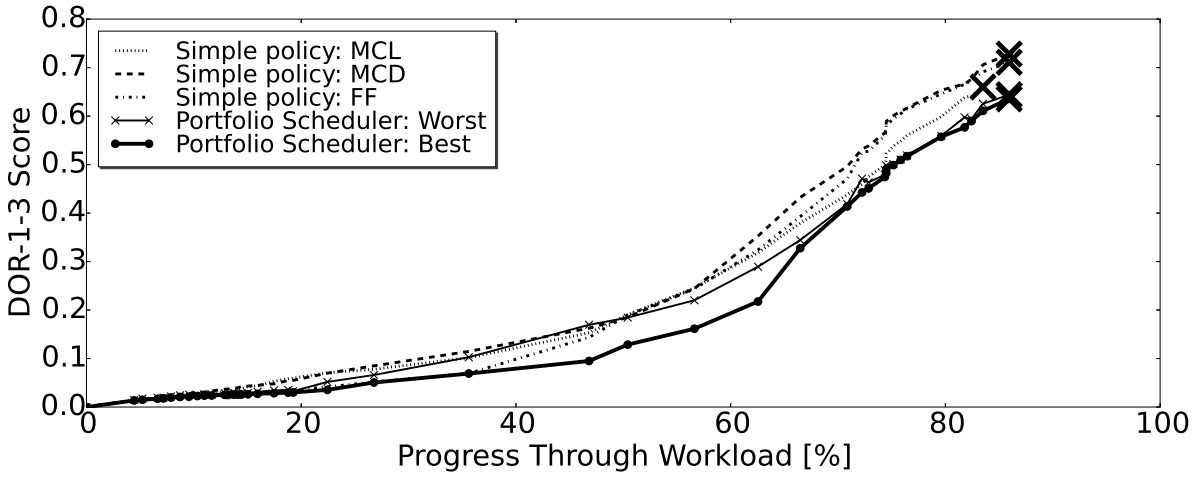


Figure I.2: Evolution of DOR with weights $(w_o, w_d) = (1,3)$ over the entire workload - Synthetic workload. The symbol X shows the system breaking point.

J Utility Function: DOR-3-1 - ST

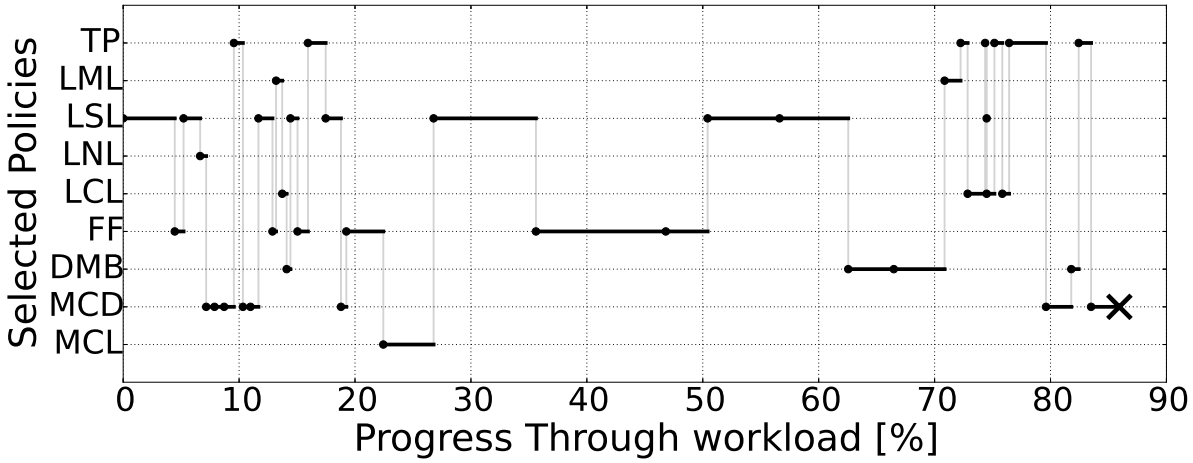


Figure J.1: Selected Policies for DOR with weights $(w_o, w_d) = (3, 1)$ - Synthetic workload. The symbol X shows the system breaking point.

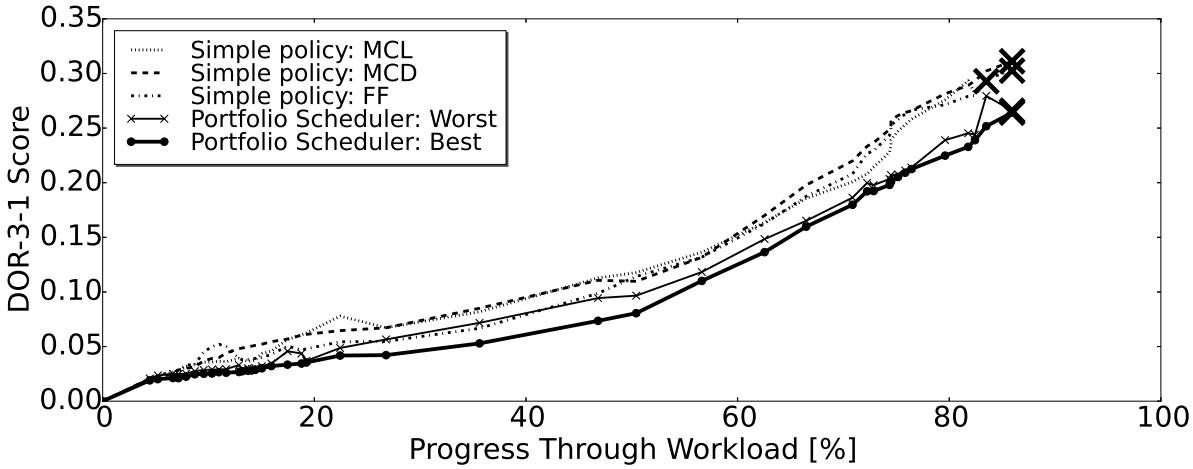


Figure J.2: Evolution of DOR with weights $(w_o, w_d) = (3, 1)$ over the entire workload - Synthetic workload. The symbol X shows the system breaking point.

K Utility Function: CL - ST

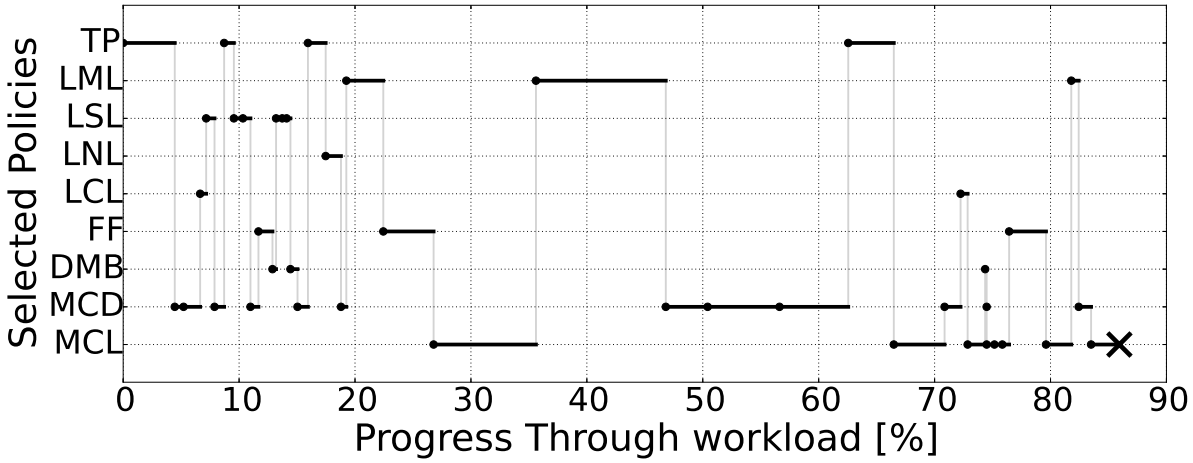


Figure K.1: Selected Policies for CL - Synthetic workload. The symbol X shows the system breaking point.

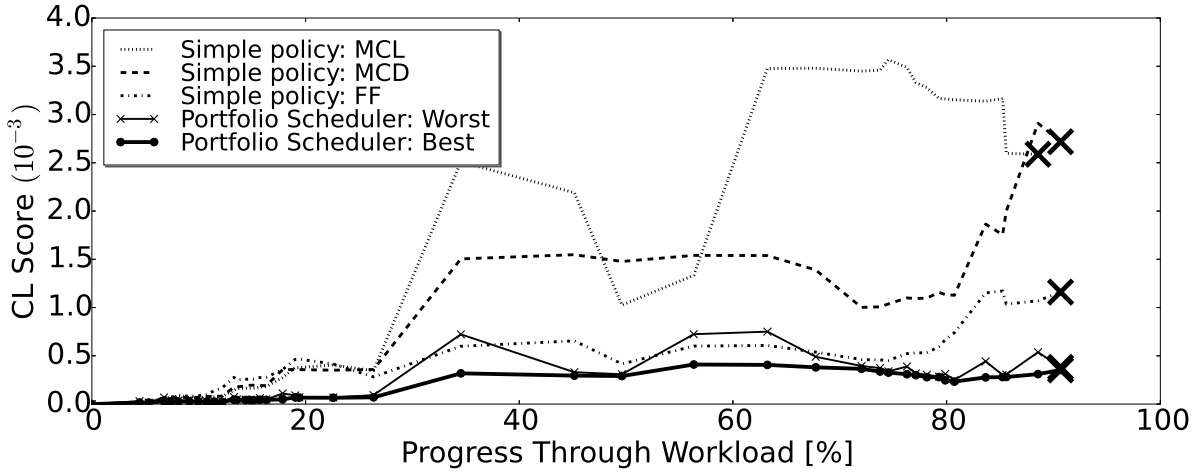


Figure K.2: Evolution of CL over the entire workload - Synthetic workload. The symbol X shows the system breaking point.

Bibliography

[1] Amazon AWS homepage. URL `https://aws.amazon.com/`.

[2] Citrix homepage. URL `https://www.citrix.com/`.

[3] Microsoft homepage. URL `https://www.microsoft.com/`.

[4] VMWare homepage. URL `http://www.vmware.com/`.

[5] Zabbix homepage. URL `http://www.zabbix.com/`.

[6] Arif Ahmed and Abadhan Saumya Sabyasachi. Cloud computing simulators: A detailed survey and future direction. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 866–872. IEEE, 2014.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. 2009.

[8] Luiz Andre Barroso. Warehouse-scale computing: Entering the teenage decade. *SIGARCH Comput. Archit. News*, 39(3), June 2011. ISSN 0163-5964.

[9] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines, second edition. *Synthesis Lectures on Computer Architecture*, 8(3):1–154, 2013.

[10] Dominic Battré, Frances M. T. Brazier, Kassidy P. Clark, Michel A. Oey, Alexander Papaspyrou, Oliver Wäldrich, Philipp Wieder, and Wolfgang Ziegler. A proposal for ws-agreement negotiation. In *Proceedings of the 2010 11th IEEE/ACM International Conference on Grid Computing, Brussels, Belgium, October 25-29, 2010*, pages 233–241, 2010.

[11] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5): 755–768, 2012.

[12] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *The journal of political economy*, pages 637–654, 1973.

[13] Sergey Blagodurov, Sergey Zhuravlev, and Alexandra Fedorova. Contention-aware scheduling on multicore systems. *ACM Transactions on Computer Systems (TOCS)*, 28(4):8, 2010.

[14] Sergey Blagodurov, Alexandra Fedorova, Evgeny Vinnik, Tyler Dwyer, and Fabien Hermenier. Multi-objective job placement in clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 66:1–66:12, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3723-6.

[15] Marin Bougeret, Pierre-François Dutot, Alfredo Goldman, Yanik Ngoko, and Denis Trystram. Combining multiple heuristics on discrete resources. In *IPDPS*, pages 1–8, 2009.

[16] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004.

[17] Yanpei Chen, Sara Alspaugh, and Randy H. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *PVLDB*, 5(12):1802–1813, 2012.

[18] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing server energy and operational costs in hosting centers. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 303–314. ACM, 2005.

[19] Louis Columbus. Roundup of cloud computing forecasts and market estimates, 2015. Forbes Tech Report, Jan 2015.

[20] Marco Comuzzi, Constantinos Kotsokalis, George Spanoudakis, and Ramin Yahyapour. Establishing and monitoring slas in complex service based systems. In *IEEE International Conference on Web Services, ICWS 2009, Los Angeles, CA, USA, 6-10 July 2009*, pages 783–790, 2009.

[21] David M. Corey, William P. Dunlap, and Michael J. Burke. Averaging correlations: Expected values and bias in combined pearson rs and fisher's z transformations. *The Journal of General Psychology*, 125(3):245–261, 1998. doi: 10.1080/00221309809595548. URL `http://dx.doi.org/10.1080/00221309809595548`.

[22] Karl Czajkowski, Ian T. Foster, Carl Kesselman, Volker Sander, and Steven Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Job Scheduling Strategies for Parallel Processing, 8th International Workshop, JSSPP 2002, Edinburgh, Scotland, UK, July 24, 2002, Revised Papers*, pages 153–183, 2002.

[23] Kefeng Deng, Junqiang Song, Kaijun Ren, and Alexandru Iosup. Exploring portfolio scheduling for long-term execution of scientific workloads in iaas clouds. In *SC*, 2013.

[24] Karim Djemame, Iain Gourlay, James Padgett, Georg Birkenheuer, Matthias Hovestadt, Odej Kao, and Kerstin Voss. Introducing risk management into the grid. In *e-Science*, pages 28–28. IEEE, 2006.

[25] Karim Djemame, James Padgett, Iain Gourlay, and Django Armstrong. Brokering of risk-aware service level agreements in grids. *Concurrency and Computation: Practice and Experience*, 23(13):1558–1582, 2011.

[26] Norman Richard Draper, Harry Smith, and Elizabeth Pownell. *Applied regression analysis*, volume 3. Wiley New York, 1966.

[27] Truong Vinh Truong Duy, Yukinori Sato, and Yasushi Inoguchi. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[28] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, and Jian Pei. A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads. In *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 -15, 2012*, page 83, 2012. doi: 10.1109/SC.2012.11. URL `http://dx.doi.org/10.1109/SC.2012.11`.

[29] Vincent C Emeakaroha, Ivona Brandic, Michael Maurer, and Schahram Dustdar. Low level metrics to high level slas-lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 48–54. IEEE, 2010.

[30] European Commission. Uptake of cloud in europe. Final Report. Digital Agenda for Europe report. Publications Office of the European Union, Luxembourg, 2014.

[31] Lionel Eyraud-Dubois and Hubert Larchevêque. Optimizing resource allocation while handling SLA violations in cloud computing platforms. In *27th IEEE IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, pages 79–87, 2013. doi: 10.1109/IPDPS.2013.67.

[32] Dror Feitelson. Packing schemes for gang scheduling. In *JSSPP*, 1996.

[33] Dror G. Feitelson, Dan Tsafrir, and David Krakov. Experience with using the parallel workloads archive. *J. Parallel Distrib. Comput.*, 74(10):2967–2982, 2014. doi: 10.1016/j.jpdc.2014.06.013.

[34] Eugen Feller, Louis Rilling, and Christine Morin. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *CCGRID*, pages 482–489, 2012.

[35] Ana Juan Ferrer, Francisco Hernández, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, RaüL Sirvent, Jordi Guitart, Rosa M Badia, Karim Djemame, et al. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.

[36] Matteo Gagliolo and Jürgen Schmidhuber. Learning dynamic algorithm portfolios. *Ann. Math. Artif. Intell.*, 47(3-4):295–328, 2006.

[37] Matteo Gagliolo and Jürgen Schmidhuber. Algorithm portfolio selection as a bandit problem with unbounded losses. *Ann. Math. Artif. Intell.*, 61(2), 2011.

[38] Javad Ghaderi, Yuan Zhong, and R Srikant. Asymptotic optimality of bestfit for stochastic bin packing. *ACM SIGMETRICS Performance Evaluation Review*, 42(2):64–66, 2014.

[39] Alfredo Goldman, Yanik Ngoko, and Denis Trystram. Malleable resource sharing algorithms for cooperative resolution of problems. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[40] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.

[41] Nikolas Herbst, Rouven Krebs, Giorgos Oikonomou, George Kousiouris, Athanasia Evangelinou, Alexandru Iosup, and Samuel Kounev. Ready for rain? a view from spec research on the future of cloud metrics. Technical Report SPEC-RG-2016-01, SPEC Research Group — Cloud Working Group, Standard Performance Evaluation Corporation (SPEC), 2016.

[42] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 27(5296), 1997.

[43] IDC. Worldwide and regional public it cloud services: 2013-2017 forecast. IDC Tech Report. [Online] Available: `www.idc.com/getdoc.jsp?containerId=251730`, 2013.

[44] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H. J. Epema. The grid workloads archive. *FGCS*, 24(7):672–686, 2008.

[45] Alexandru Iosup, Omer Ozan Sonmez, and Dick H. J. Epema. Dgsim: Comparing grid resource management architectures through trace-based simulation. In *Euro-Par 2008 - Parallel Processing, 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008, Proceedings*, pages 13–25, 2008. doi: 10.1007/978-3-540-85451-7_3. URL `http://dx.doi.org/10.1007/978-3-540-85451-7_3`.

[46] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155 – 162, 2012.

[47] Brendan Jennings and Rolf Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

[48] Myung Hyun Jo and Won Woo Ro. Contention-free fair queuing for high-speed storage with raid-0 architecture. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, pages 174–183. IEEE, 2015.

[49] Foued Jrad, Jie Tao, and Achim Streit. SLA based service brokering in intercloud environments. In *CLOSER*, pages 76–81, 2012.

[50] Dalibor Klusácek and Simon Tóth. On interactions among scheduling policies: Finding efficient queue setup using high-resolution simulations. In *Euro-Par*, 2014.

[51] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Comp. Syst.*, 23(7):888–903, 2007.

[52] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[53] Zheng Li, Liam O'Brien, He Zhang, and Rainbow Cai. On a catalogue of metrics for evaluating commercial cloud services. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 164–173. IEEE Computer Society, 2012.

[54] Kuan Lu, Thomas Röblitz, Ramin Yahyapour, Edwin Yaqub, and Constantinos Kotsokalis. Qos-aware sla-based advanced reservation of infrastructure as a service. In *CloudCom*, pages 288–295, 2011.

[55] Kuan Lu, Ramin Yahyapour, Philipp Wieder, Edwin Yaqub, Monir Abdullah, Bernd Schloer, and Constantinos Kotsokalis. Fault-tolerant service level agreement lifecycle management in clouds using actor system. *Future Generation Comp. Syst.*, 54:247–259, 2016.

[56] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[57] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259. ACM, 2011.

[58] Richard C Merton. *Optimum consumption and portfolio rules in a continuous-time model*. MIT, 1970.

[59] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *research. microsoft. com*, 2011.

[60] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.

[61] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *SoCC*, 2012.

[62] Xiaojuan Ren, Seyong Lee, Rudolf Eigenmann, and Saurabh Bagchi. Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation. *Journal of Grid Computing*, 5(2):173–195, 2007.

[63] Zujie Ren, Xianghua Xu, Jian Wan, Weisong Shi, and Min Zhou. Workload characterization on a production hadoop cluster: A case study on taobao. In *IISWC*, pages 3–13, 2012.

[64] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. *INC, IMS and IDC*, pages 44–51, 2009.

[65] Ohad Shai, Edi Shmueli, and Dror G. Feitelson. Heuristics for resource matching in intel's compute farm. In *JSSPP*, 2013.

[66] Siqi Shen, Kefeng Deng, Alexandru Iosup, and Dick H. J. Epema. Scheduling jobs in the cloud using on-demand and reserved instances. In *Euro-Par*, pages 242–254, 2013.

[67] Siqi Shen, Vincent van Beek, and Alexandru Iosup. Workload characterization of cloud datacenter of bitbrains. Technical Report PDS-2014-001, TU Delft, February 2014. `http://www.pds.ewi.tudelft.nl/fileadmin/pds/reports/2014/PDS-2014-001.pdf`.

[68] Siqi Shen, Vincent van Beek, and Alexandru Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *CCGRID*, 2015.

[69] Arjun Singh et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *SIGCOMM*, pages 183–197, 2015.

[70] Alexander L. Stolyar and Yuan Zhong. A large-scale service system with packing constraints: minimizing the number of occupied servers. In *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '13, Pittsburgh, PA, USA, June 17-21, 2013*, pages 41–52, 2013. doi: 10.1145/2465529.2465547. URL `http://doi.acm.org/10.1145/2465529.2465547`.

[71] Matthew J. Streeter, Daniel Golovin, and Stephen F. Smith. Combining multiple heuristics online. In *AAAI*, pages 1197–1203, 2007.

[72] David Talby and Dror G. Feitelson. Improving and stabilizing parallel computer performance using adaptive backfilling. In *IPDPS*, 2005.

[73] Vincent van Beek. Design and evaluation of a portfolio scheduler for business-critical workloads, hosted in cloud datacenters. Master's thesis, TUDelft, 2015.

[74] Vincent van Beek, Jesse Donkervliet, Tim Hegeman, Stefan Hugtenburg, and Alexandru Iosup. Self-expressive management of business-critical workloads in virtualized datacenters. *IEEE Computer*, 48 (7):46–54, 2015.

[75] VMware. vSphere 6 documentation: Monitoring and performance. Official support resources. URL `https://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-6-pubs.html`.

[76] Vladimir Vovk. Combining p-values via averaging. *arXiv preprint arXiv:1212.4966*, 2012.

[77] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75. IEEE, 2011.

[78] Philipp Wieder and Ramin Yahyapour. Grid environments: Service level agreements (slas). In *Encyclopedia of Software Engineering*, pages 347–360. 2010.

[79] Philipp Wieder, Jan Seidel, Oliver Wäldrich, Wolfgang Ziegler, and Ramin Yahyapour. Using sla for resource management and scheduling-a survey. In *Grid Middleware and Services*, pages 335–347, 2008.

[80] Gerhard J Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.

[81] Pengcheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *ICPE*, pages 271–282. ACM, 2013.

[82] Jung-Lok Yu, Chan-Ho Choi, Du-Seok Jin, Jongsuk Ruth Lee, and Hee-Jung Byun. A dynamic virtual machine allocation technique using network resource contention for a high-performance virtualized computing cloud. *International Journal of Software Engineering and Its Applications*, 8(9):17–28, 2014.