Design and Evaluation of a Portfolio Scheduler for Business-Critical Workloads Hosted in Cloud Datacenters.

Vincent Simon van Beek
DESIGN AND EVALUATION OF A PORTFOLIO SCHEDULER FOR BUSINESS-CRITICAL WORKLOADS HOSTED IN CLOUD DATACENTERS.

by

Vincent Simon van Beek

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

at the Delft University of Technology, to be defended publicly on Tuesday January 27, 2015 at 16:30 AM.

Supervisor: Assistant Prof. dr. ir. Alexandru Iosup
Thesis committee: Prof. dr. ir. D.H.J. Epema,
Assistant Prof. dr. ir. A. Iosup,
Assistant Prof. dr. ir. C. Hauff,
Ir. T. van Drimmelen, PDEng
TU Eindhoven and TU Delft
TU Delft
TU Delft
Asp4all Bitbrains

An electronic version of this thesis is available at http://repository.tudelft.nl/.
Design and Evaluation of a Portfolio Scheduler for Business-Critical Workloads Hosted in Cloud Datacenters.

MASTER THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

V.S. van Beek
born in Amsterdam, The Netherlands
Abstract

Virtualized multi-cluster, multi-datacenter datacenters are central to the digital economy, but require new techniques to address increasing scales, changing architectures, and dynamic workloads. Especially the introduction of new Business-Critical workloads that we characterize, create new resource management challenges. We propose a resource management and scheduling architecture that uses resources efficiently and reduces the risk of low performance. For this we create and evaluate a portfolio scheduler that is adapted to support multi-cluster, multi-datacenter architectures.
ACKNOWLEDGEMENTS

It took me 10 year to get to this point and it has been an interesting and enriching journey. It seems a long
time for a degree that could theoretically take you five year. While most students would spend most of their
time studying, I spent most of my time being an athlete and traveling. Besides study, sport, and traveling
I also explored the opportunities of being a freelance software developer. The final years of my university
career I found the motivation to focus on research and academic achievements. This ultimately culminated
in this master thesis project.

Before I say anything more, there are some people whom I should thank for their influence on the outcome
of my university career and this thesis. If your name has been left out, please accept my non-nominative
thanks.

First off, I want to thank Alexandru Iosup, my master thesis supervisor. It was the course on Cloud
Computing taught by Alexandru that inspired me to focus my attention on this topic. Besides the inspiration
I want to thank Alexandru for pushing me to strive for great results and helping actually finish my master
thesis in the end.

Secondly I want to thank some people at Bitbrains. To start off I want to thank Theo van Drimmelen my
daily supervisor who helped me stay on track with working on my thesis project instead of all sorts of other
interesting side projects. I also want to thank Hein Brat who introduced me to Bitbrains and gave me the
opportunity to work at a very creative company. During my project I had plenty of opportunities to consult
two very gifted experts at Bitbrains, I want to thank both Gjalt van Rutten and Mike Wessling for there patience
and time. To all other engineers that helped me getting started, and during the project many thanks for your
time.

Furthermore, I want to thank the other members of the committee, Dick Epema and Claudia Hauff, for
their interest in my research project.

Finally I want to thank my family and friends for supporting me. Especially my parents for keeping faith
that I would actually finish this journey successfully.

Vincent Simon van Beek
Delft, January 2015
4 Design of a Portfolio Scheduler for Business-Critical Workloads in Cloud Datacenters 34
4.1 Overview ........................................................................................................... 34
4.2 Requirements for the Portfolio Scheduler ..................................................... 35
  4.2.1 Restrictions on Provisioning VMs ................................................................. 35
  4.2.2 Other Requirements .................................................................................... 35
4.3 High-level Design of a Portfolio Scheduler for Business-Critical Workloads .... 36
4.4 Detailed Design Decisions ............................................................................... 36
  4.4.1 Policies for Provisioning VMs to Clusters ..................................................... 38
  4.4.2 Method of Selecting the Best Policy After Simulation ................................. 38
  4.4.3 Storage Latency Model ................................................................................. 42
4.5 Summary .......................................................................................................... 43

5 Evaluation of a Portfolio Scheduler for Business-Critical Workloads 44
5.1 Overview .......................................................................................................... 44
5.2 Multi-Cluster, Multi-Datacenter Experimental Setup ...................................... 44
  5.2.1 Process for Evaluating the Portfolio Scheduler ............................................ 47
5.3 Metrics for Evaluating the Portfolio Scheduler ................................................. 47
5.4 Results of the Simulation Based Evaluation ..................................................... 49
  5.4.1 Distribution of Load for Different Configurations ......................................... 49
  5.4.2 Distribution of Selected Policies for Different Configurations ..................... 50
  5.4.3 Performance and Scalability Evaluation of the Portfolio Scheduler ............. 51
5.5 Summary .......................................................................................................... 55

6 Conclusion and Future work ................................................................................ 56

Bibliography ........................................................................................................... 58

Appendices ............................................................................................................. 62

A Literature Study on Scheduling ........................................................................... 64
  A.1 Scheduling ..................................................................................................... 65
    A.1.1 Introduction ............................................................................................... 65
    A.1.2 Taxonomy ................................................................................................ 66
    A.1.3 Terms and Definitions ............................................................................. 67
    A.1.4 Latest Developments .............................................................................. 68
    A.1.5 Future Research Directions ................................................................... 73

B Network usage RND set .................................................................................... 74

C CPU usage RND set ............................................................................................ 76

D Memory usage RND set ...................................................................................... 77

E CPU and Memory Usage Over Time RND set ..................................................... 78

F CPU and Memory Usage Over Time RND set ..................................................... 79

G Disk Write Usage Over Time RND set .............................................................. 80

H CPU Memory Correlation RND set .................................................................. 81

I Distribution of PCC RND set .............................................................................. 82

J Peak to Mean Ratio RND set .............................................................................. 86

K Autocorrelation RND set ................................................................................... 88

L Network Usage per Cluster per Scenario During Simulation ............................ 90

M Storage Usage per Cluster per Scenario During Simulation ............................ 93

N CPU Usage per Cluster per Scenario During Simulation ............................... 96

O Mean Usage per Cluster per Scenario During Simulation ............................. 97
Spurred by a rapid development of hardware and of resource management techniques, cloud datacenters are hosting an increasing number of application types. Over a billion people access daily a diverse collection of free or paid cloud utilities, from search to business-critical financial operations, from online social gaming to engineering [1–3]. The rapid growth, and the scale, and the diversity of workloads at which computing resources are needed leads to resource management problems. In this work, we design and evaluate a portfolio scheduling technique that can be used to manage datacenter resources, with a special focus on business-critical workloads.

Datacenters house IT resources that form the basis for modern knowledge based economies. In the Netherlands there is a large datacenter market, mainly driven by the strong Internet connectivity. Amsterdam the capital of the Netherlands is one of the worlds larges Internet hubs. The connectivity and the availability of datacenters has lead to increasing demand for companies that can manage datacenters, so that business-critical workloads are efficiently and correctly run. One of such company is Asp4all Bitbrains (in short Bitbrains). Bitbrains is a mid-sized managed hosting provider in the Netherlands. The customer base of Bitbrains is mainly large enterprises in the Netherlands with an emphasis on financial institutions (banks, pension funds, and insurance companies). Bitbrains operates a multi-cluster, multi-datacenter cloud platform. Most of the customers use the fully managed services of Bitbrains. This means that Bitbrains engineers manage on behalf of Bitbrains’ customers both the operating system (OS) and the software running on top of the OS. A small portion of the Bitbrains customer base consume only Infrastructure as a Service (IaaS) offerings, and manage themselves the IT resources they lease temporarily.

Bitbrains is an interesting company to collaborate with as a researcher because they have a strong research minded engineering core. Besides the culture and the people working at Bitbrains we found good reason for collaborating with Bitbrains. The customers of Bitbrains present them with business-critical workloads of a kind that is common in the industry, but not studied in the context of clouds. This gives us an excellent opportunity to create a new body of knowledge. Business-critical workloads are workloads that combine financial applications (e.g., Monte Carlo simulators, large scale distributed Excel applications, distributed Matlab applications etc.) and traditional enterprise IT applications (e.g., data analytics, web-hosting, etc.). In this work we present a thorough statural analysis of these workloads and we compare them with known cloud, grid, and web workloads. This study show that business-critical workloads are different from previously studied workloads. We then focus on a novel method to schedule such workloads.

A common technique to provide compute, memory, storage, and network resources to enterprise customers is virtualization. With this technique multiple virtual machines (VMs) can run on one physical machine (virtualization is explained in more depth in Section 2.2). Virtualization is also used by Bitbrains to deliver services and resources to their customers. Managing the distribution of virtual resources on physical resources is a complex task that requires a smart scheduler. The scheduling of workloads, in this setting is, a complex task because of the following reasons:

1. the introduction of new workloads (Business-Critical workloads),
2. the scale at which IT providers like Bitbrains operate (multiple clusters in multiple datacenter),
3. the multi dimensional nature of the resources (compute, memory, storage, and network),
4. and the rapid growth of both the workloads and the physical infrastructure.

Evaluating designs of complex systems similar to the design we present in this work is a nontrivial task. The purpose of the evaluation is to show that our design can work in practice. The difficulty of evaluating our design is the implementation of a prototype. Implementing the prototype involves writing complex scheduling software and a simulator. We use simulation with real world workload traces to evaluate the scheduler.

In the remainder of this chapter we describe why this project is relevant, what it entails and how the research is structured. We conclude with our main findings and a reading guideline.

1.1. Problem Statement
In this section we give both theoretical and practical insights into the problem. The theoretical insight indicates the complexity of the problem as well as boundaries to what can be achieved from a theoretical stand point. The practical insights demonstrate that, besides the theoretical challenges, there are also many practical constraints that complicate the problem further.

1.1.1. Abstract Problem Model
Scheduling VMs on physical hardware is not a trivial task, as explained also earlier in the introduction. The complexity of this problem is due to solving a multi-objective (e.g., utilization and performance), multi-dimensional (e.g., CPU load, IO load, Memory, Disk space, Network load, and CPU cores) optimization problem. Scheduling VMs on physical machines can be modeled as a Vector Bin Packing problem \([4, 5]\). This model is a good abstraction of the practical scheduling problem because it can accommodate for modeling the multidimensional nature of the VM scheduling problem. The core concept of the bin-packing problem is to put a set of items in the fewest number of bins possible given that the bins have a maximum capacity and the items have individual weights smaller or equal to the largest bin. The most trivial form of the bin-packing problem is the one-dimensional bin-packing problem. The one-dimensional bin-packing problem has received a lot of attention in the past and there are good approximation algorithms, but remains NP-hard.

In Figure 1.1 we present an example of the one-dimensional bin-packing problem, it shows 4 bins with 8 items in them. The numbers in the items represent the weights of the items. In this example, the left-most IT resource is concurrently running two tasks, one incurring a load (weight) of 5, the other a load of 3. In the same example, the right-most resource is responsible for load-imbalance in the system, because its only running task has a load of 2, whereas the cumulative load of each the other IT resources in Figure 1.1 is 8. Load imbalance has been shown in scheduling literature to lead to inefficient resource use and, in some cases, also to applications that combine multiple tasks to run much slower than expected.

The multi-dimensional bin packing problem, also known as the Vector Bin Packing problem can accommodate for multiple dimensions. Each bin is represented by a vector that holds the maximum capacities for each of the dimensions. Items are also represented by vectors, the values in these vectors represent the weights for each of the dimensions. To translate the VM scheduling problem to the Vector Bin Packing problem we
1.1. Problem Statement

represent each resource (e.g., CPU load, IO load, Memory, Disk space, Network load, and number of CPU cores) as a dimension in the vectors. Solving the Vector Bin Packing problem proves to be very complex. Woeginger [6] shows that it is APX-hard, this means that there is no Polynomial-time approximation scheme (PTAS) for this problem. Therefore approaches in this domain focus on finding good heuristics to solve practical scheduling problems [4, 5].

To complicate matters further, the bin packing problems generally address only static problems, that is, problems where the bins and the items have characteristics, such as weight, that do not change over time. For the VM scheduling problem, VMs can have variable resource requirements over time. This means that for all the dimensions in the problem, the weights of the items can fluctuate over time. As a consequence, we cannot simply apply the general bin-packing approaches to the scheduling problem posed by modern datacenters.

For the VM scheduling problem this means that we need to develop a heuristics based approach for finding approximations of the optimal VM physical host mapping.

1.1.2. Practical Constraints

Besides finding a good mapping between resource requirements (represented by items) and available resources (represented by bins), there are also additional constraints that need to be taken into account. An example of such a constraint is the anti-affinity constraint, which requires that certain VMs are not allowed to reside on the same hardware (in the same bin). In practice the anti-affinity constraint is even more complex, because there are not only VMs that are not allowed to reside on the same host, but restriction also apply to clusters (set of bins) and datacenters (sets of sets of bins). We also identify an affinity constraint, that is, that VMs should reside as close to each other as possible for performance reasons.

Another practical constraint is introduced by the common resource-management practices in the large scale computing domain. In practice, datacenters are managed hierarchically, with a structure that complicates the application of traditional scheduling approaches. In Figure 1.2 we present an example of such a hierarchical structure. At the highest level we find datacenters, in the next level we find clusters, and within the clusters we find hosts (in the figure represented by bins). In the bins we find items, which represent VMs. For this structure, the scheduling problem becomes one of nested bin-packing, which is very challenging to solve within the time constraints of datacenters, where decisions need to be taken at least every few minutes.

1.1.3. Portfolio Scheduling

In previous work on workload scheduling Deng et al. [7] introduce portfolio scheduling for scientific workloads. The concept of using a portfolio of policies or algorithms is first introduced into the computing domain by Huberman et al. [8]. They show that a portfolio of algorithms can be used for solving NP-Hard problems. Portfolio scheduling uses a set (portfolio) of scheduling policies from which the best policy is selected for a workload presented to the scheduler. This approach is promising because it shows very good results for scheduling scientific workloads in the datacenter.

In this work we, adapt the work of Deng et al. [7] so that our system can manage business-critical workloads running in a multi-cluster, multi-datacenter setting. In portfolio scheduling, choosing which policy to use is
the most challenging part. Similarly to Deng et al., we use a simulator to test the performance of multiple scheduling policies. The results of the simulator are then used to make the decision which policy to use. Extending previous work, for our project we design scheduling policies tailored to the requirements of business-critical workload. We also design a simulator that can accommodate for the multi-cluster, multi-datacenter nature of the infrastructure we are targeting, which is more complicated than the single datacenter targeted by previous work. Moreover, for the scheduling of business-critical workloads it is important to consider not only one resource type (e.g., runtime is used by Deng et al.) but multiple resource types simultaneously (e.g., CPU, memory, storage, and network).

In Figure 1.3 we depict an example of workloads arriving over time. These workloads are scheduled by a portfolio scheduler with two policies (P1 and P2). The portfolio scheduler decides, based on performance predictions made by a simulator, to use P1 for the workloads in the left- and right-most blocks, and P2 for the workload in the center block. Increasing the number of types of workload patterns and the number of policies in the portfolio will quickly complicate the selection process, but lead to a realistic selection situation that exceeds the capabilities of human decision-making yet it is well within the grasp of a portfolio scheduler.

### 1.1.4. Research Questions

The main objective of this project is to deliver a proof of concept scheduler that can support engineers in making decisions about the placement of VMs for business-critical workloads in a multi-cluster multi-datacenter setting. As described in Section 1.1.1 and 1.1.2 the problem of deciding where to place VMs is non-trivial to solve. Before even attempting to build a scheduler for this purpose it is important to understand the characteristics of the workload involved.

To achieve the main objective, we set to answer the following main research questions:

1. What are the characteristics of business-critical workloads?
2. How to adapt the concept of portfolio scheduling to the placement of VMs in a multi-cluster multi-datacenter setting?
3. How to evaluate the adapted portfolio scheduler, experimentally, through the implementation of a prototype?

### 1.2. Approach

In this section we describe the research approach, for each of the three main research questions, in turn:

1. What are the characteristics of business-critical workloads?
   - We first collect workload traces from the multi-datacenter Bitbrains infrastructure. The collection is carried out by utilizing VMware tooling that is already in place. For the characterization of business-critical workloads, we use a combination of basic statistical analysis, correlation studies, and time-pattern analysis. The approach can be characterized as statistical research.

2. How to adapt the concept of portfolio scheduling to the placement of VMs in a multi-cluster multi-datacenter setting?
   - We first carry out a requirements solicitation process, using input from Bitbrains engineers. This allows us to understand both system requirements and operational system limits. We then design a portfolio scheduler that takes into account the requirements and the workload characteristics studied before,
and equip it with VM placement policies. The approach can be characterized as conceptual research.

3. How to evaluate the adapted portfolio scheduler, experimentally, through the implementation of a prototype?
   To evaluate the system we implement a prototype and carry out experimental validation to verify the workings of the portfolio scheduler. For the experimental validation we use real world workload traces. The approach can be characterized as experimental research.

1.3. **Main Contributions**

The contribution of this project is five-fold:

1. The first major contribution of this work is the characterization of business-critical workloads. Compared to previous work, our characterization contains more resource types (CPU, Memory, Network, and Storage); this is particularly relevant because, although knowledge about all types of resources is needed, most other studies only focus on CPU. Additionally, we carry out three major types of analysis (basic statistics, correlation study, and time pattern analysis), so our work is more thorough than related work. The workload analysis has been accepted for publication in the conference IEEE/ACM CCGRID 2015: **V.S. Van Beek et al., Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters, IEEE/ACM CCGRID 2015, Shenzhen, Guangdong, China, May 4-7, 2015 (acceptance ratio 25%).** The results of the study are also under submission in the ICTOPEN 2015 conference, which is a Dutch conference that aims to make academia and industry meet\(^1\). Prior to the IEEE/ACM CCGRID 2015 conference we will publish our workload traces in the Grid Workloads Archive \[9\].

2. The second major contribution is conceptual. We design a novel portfolio scheduler for business-critical workloads. This involves designing: scheduling policies, a multi-cluster multi-datacenter simulator, and an evaluation mechanism to decide which policy scored best in a simulation run. The design of the scheduler forms the basis for an article currently under submission: **V.S. Van Beek et al., Mnemos: Self-Expressive Management of Business-Critical Workloads in Virtualized Datacenters, IEEE Computer Special edition (Self-Aware and Self-Expressive Computing Systems), submitted on 15 December 2014 (expected acceptance ratio 10-30%).**

3. The third major contribution is the experimental evaluation of the designed portfolio scheduler. For this we implemented a prototype and designed an experimental setting to test and validate the workings of the portfolio scheduler. For the experimental evaluation we use real world workload traces from the Bitbrains infrastructure.

4. The results of this thesis have further resulted in invited lectures at both TU-Delft and TU/e Eindhoven, and discussions with VMware and other companies. We have received so far very positive feedback.

5. To continue the work on portfolio scheduling and workload analysis we have applied and received two research grants. For continuing the work on portfolio scheduling, we received a grant from COMMIT, which will mainly be used to valorize the work into a production system. The other grant, for an NWO/STW KIEM project will be used to continue the work on workload analysis and will help us to create benchmarks based on these workload studies.

1.4. **Reading Guidelines**

This document is structured as follows: In Chapter 2 we present background information on datacenters, virtualization, portfolio scheduling, and we present the related work. In Chapter 3 we present a thorough statistical analysis of business-critical workloads and we compare them with known cloud, grid, and web workloads. In Chapter 4 we investigate the requirements and present a design for scheduling business-critical workloads in a multi-cluster, multi-datacenter setting. In Chapter 5 we present an experimental evaluation of our portfolio scheduler for business-critical workloads.

\(^1\)ICTOPEN focuses on presentation, not on publication. This venue does not require copyright, does not publish proceedings, and accepts material published elsewhere.
In this chapter we provide basic knowledge regarding datacenters, virtualization and portfolio scheduling to prepare the reader for the remainder of this thesis.

2.1. **Primer on Datacenters**

Datacenters are facilities that house IT resources (servers, storage, and network devices) and the surrounding infrastructure (communication networks, power distribution, and cooling) to operate them. Datacenters come in all sorts of sizes and shapes. The smallest datacenters are not larger than a normal office room and the biggest datacenters span up to 600,000 square meters (see Figure 2.1 for an example of a Google datacenter). We find the smallest datacenters in enterprise office buildings, university campuses, hospitals etc. The large datacenters are dedicated datacenters, they are specially build to house computers. The remainder of this section is organized as follows:
1. **Datacenter operating models**: commercial and operational models for operating datacenters.

2. **Datacenter Tiers**: datacenters can be characterized according to their fault tolerance, this categorization is done according to four tiers.

3. **Racks for housing computer infrastructure**: racks are the basic building blocks with which we build datacenters.

### 2.1.1. **Datacenter Operating Models**

Datacenters are operated according to four operational models: privately owned datacenters (Google, Amazon, Microsoft, etc), managed hosting / cloud, colocation, and wholesale.

1. **Privately owned datacenters**: The large internet operators like Google, Amazon and Microsoft own a number of private datacenters. These datacenters are specially designed for their specific needs and are operated accordingly.

2. **Colocation**: Colocation datacenter operators sell datacenter space, this means they sell racks (racks are discussed in Section 2.1.3) or rooms where other companies can put their own privately owned computer systems. The colocation operator is responsible for the electricity and cooling of the datacenter.

3. **Managed hosting / cloud**: Datacenter space is sold on the level of application and operating systems so the end customer has no control and ownership over the underlying computer hardware.

4. **Wholesale**: Datacenter space is sold by floor space so the customer has to put in their own racks and power distribution. This is different from colocation because with colocation the datacenter owner is responsible for putting in the racks and power distribution.

### 2.1.2. **Datacenter Tiers**

The second criterium that can be used to make distinctions between datacenters is the tier level at which they operate. The Telecommunications Industry Association has developed a standard to which all datacenters in the world are categorized (the TIA-942: Datacenter Standards Overview). The main difference between the tier levels is the amount of downtime per year they are allowed to have. Designing datacenters for increasing less downtime comes with higher cost and more requirements on both cooling and power supply.

- **Tier 1**:
  - Single non-redundant distribution path serving the IT equipment
  - Non-redundant capacity components
  - Basic site infrastructure with expected availability of 99.671%
  - Tier 1 would allow 1729.224 minutes downtime

- **Tier 2**:
  - Meets or exceeds all Tier 1 requirements
  - Redundant site infrastructure capacity components with expected availability of 99.741%
  - Tier 2 would allow 1361.304 minutes downtime

- **Tier 3**:
  - Meets or exceeds all Tier 2 requirements
  - Multiple independent distribution paths serving the IT equipment
  - All IT equipment must be dual-powered and fully compatible with the topology of a site’s architecture
  - Concurrently maintainable site infrastructure with expected availability of 99.982%
  - Tier 3 would allow 94.608 minutes downtime

- **Tier 4**:
  - Meets or exceeds all Tier 3 requirements
2. Background

(a) Old datacenter setup servers on shelves.
(b) Modern datacenter constructed out of standard racks.
(c) Blade server.

Figure 2.2: Datacenter design.

- All cooling equipment is independently dual-powered, including chillers and heating, ventilating and air-conditioning (HVAC) systems
- Fault-tolerant site infrastructure with electrical power storage and distribution facilities with expected availability of 99.995%
- Tier 4 would allow 26.28 minutes downtime

2.1.3. Racks for Housing Computer Infrastructure

Servers in datacenters are not stacked on to shelves (see Figure 2.2a), instead they are mounted in special racks (see Figure 2.2b). These racks are 19-inch wide and come in different heights. The height of racks is measured in Us (one U is 1.75 inches). A standard rack is 42U high, there are however suppliers that sell taller racks.

Traditionally IT resources come in 1U to 4U shapes. Sometimes blade server enclosures are used (e.g. Figure 2.2c), these often require 10U. In these blade server enclosures actual servers are slotted that have a non standard shape.

2.1.4. Datacenter Power Supply

The power supply to IT resources is one of the two most fundamental part to get right in a datacenter (the second is cooling and is discussed next). Higher tier datacenters (tier 2 and higher) provide redundant power supply to IT resources. Tier 4 datacenter even have at least two connection to the power grid (country wide power network). Besides the redundancy it is also important to have enough power per rack. Different datacenters can offer more or less power per rack. The amount of power strongly influences the amount of IT resources that can be installed in a single rack.

2.1.5. Datacenter Cooling

Servers consume a lot of power and produce a great deal of heat. It is the duty of a datacenter operator to design and maintain the cooling infrastructure that makes sure that the IT resources remains cool enough. Datacenter operators use a variety of cooling techniques (compressors also known as chillers, vaporizers, passive cooling with the outside air, and heat exchange systems that the store and use heat/cold in the ground).

2.2. Virtualization

Virtualization is a technique used to subdivide computer resources (CPU, memory, storage and network). Traditional we would install an operating system (OS) on top of physical hardware. With virtualization we install a hypervisor on top of the hardware and install one or more virtual machines on top of the hypervisor. These virtual machines can than be used as if they are physical computers. Figure 2.3 shows an example of how virtualization can be used to subdivide a physical server, in the figure the VMware layer represents the hypervisor.

Barham et al. [10] present Xen an open source hypervisor. In the article the authors present the inner-workings of a hypervisor and they describe the tradeoffs between generic support of all operating systems and performance.
One of the reasons for introducing virtualization was that utilization of servers is very low (often under 30% \cite{11}). With virtualization we can consolidate multiple virtual servers on one physical server this will result in higher utilization and thus more efficient use of resources. With the gain in efficiency we however introduce performance issues. If for instance two virtual machines try to use the CPU at the same time this will impact the performance of both machines because the hypervisor will schedule CPU time evenly for both virtual machines.

2.3. PORTFOLIO SCHEDULING

Workload scheduling is a topic that attracts increasingly more attention \cite{7, 12-14}. Especially workload scheduling at larger scale, multiple clusters / datacenters is a topic that is very complex and requires more research to find better solutions for optimizing resource usage. One of the strategies of optimizing workload scheduling in large complex environments is portfolio scheduling. In this section we discuss the basics of portfolio scheduling to provide the necessary background to understand the remainder of this thesis.

The following topics are discussed in Sections 2.3.1-2.3.4.

1. The general concept and the rational of portfolio scheduling
2. Four stages of implementation and application
3. Portfolio scheduling operational models
4. Portfolio scheduling for scientific workloads

2.3.1. THE GENERAL CONCEPT OF PORTFOLIO SCHEDULING

The concept of portfolio scheduling originates from portfolio theorem known in the financial sector. It is used to describe a model of selecting stocks that together reduce risk and optimize profit. In the context of workload provisioning the goal is to optimize the placement of workload on compute resources. In traditional scheduling approaches one single policy is used to make the decision where workload should go. Often these policies are optimized for a certain environment or workload. The result is that when the environment or workload changes there is a big risk that the policy does not perform optimal anymore. Adapting the policy is than the only option, but this is difficult and expansive because it requires highly skilled human operators to analyze and design a new policy. By utilizing a set of policies portfolio scheduling reduces this risk. One can still profit form the specialized policy because it can still be chosen by the scheduler, but when new or different workloads are presented the scheduler has the possibility to select the best policy for the new situation. The second big advantage of portfolio scheduling is that we can use relatively simple policies and we can have multiple policies that each work best for a certain workload pattern.
2.3.2. Four Stages of Implementation and Application

Deng et al. [7] present a periodic portfolio scheduler for scientific workloads which shows very good results in both simulation and real life deployment on the DAS4 grid. In their work they describe the following four step process on how to create a portfolio scheduler.

1. creation, in this step the policies are created that are later used for scheduling
2. selection, in this step a mechanism is developed that selects policies from step 1
3. application, in this step the policy selected in step 2 is used for the actual provisioning
4. reflection, in this step the performance of the scheduler is analyzed, this can result in changes in parameters and or the portfolio (changing, removing, or adding policies)

2.3.3. Portfolio Scheduling Operational Models

The portfolio scheduling model lends it self for the following two models of operation:

1. Continuous portfolio scheduling: the scheduler evaluates all the policies for each scheduling decision.
2. Periodic portfolio scheduling: the scheduler evaluates the policies for the next time interval, during a time interval the same policy is used.

The choice of model depends on two factors: runtime of the scheduler (how much time does it take to evaluate the policies) and the inter arrival time of workload (time between the decisions that need to be made by the scheduler). If the inter arrival time is longer than evaluation time the continuous portfolio scheduling can be used. If the inter arrival time is shorter than the evaluation time the only option is the Periodic portfolio scheduling.

2.3.4. Portfolio Scheduling for Scientific Workloads

Deng et al. [7] design, implement and evaluate a periodic portfolio scheduler for scientific workloads. Figure 2.4a provides a high-level operational model of the scheduler. In the model Deng et al. describe workloads are submitted by users, these workloads end up in the Job queue. The jobs are executed by allocating resources to them, resources are provisioned using a combination of both job allocation policies and virtual machine (VM) provision policies. For each period a combination of policies is used during execution. The scheduler uses jobs in the queue and the current state of the system to run simulations to determine which combination of policies to use next. Figure 2.4b shows how the scheduler determines what the next active policy should be. Two elements that need some attention in this model are the Simulator and the Selection criteria.

The Simulator

The choice of Simulator is a non trivial, it involves balancing the accuracy of the simulated results with the runtime of the simulator. Deng et al. use the DGSim [15] simulator. DGSim is an event-driven simulator.
which makes it relatively fast. The accuracy of the simulator depends on the workload traces that are available, this is a strong property because this means we can control the accuracy our selves.

**The Selection criteria**

Deng et al. define the selection criterion as a utility function that balances user experience (job slowdown) and cost (system utilization). When we work on complex systems that are prime candidates for portfolio scheduling we often have to balance multiple optimization criteria. In portfolio scheduling we deal with this balancing act by incorporating it as part of the policy selection function.

### 2.4. Related Work

In this section we present related work on workload characterization and workload scheduling.

#### 2.4.1. Related Work for Workload Characterization

In this section we present a comprehensive comparison between our and related work, along two axes: contributions related to public datasets and contributions related to workload characterization in datacenters.

**Dataset release** Our data release complements well the few datasets that are publicly available. Many previous datacenter studies have used the workloads of distributed systems, from parallel [16] and grid [9] environments. The seminal Google workload dataset [17], released in 2011, includes only CPU and memory characteristics, and only normalized, rather than actual, values. The public SWIM workloads repository includes 5 workload traces, possibly extracted from publicly characterized Facebook MapReduce traces [18], but very short (only 1 day) and with no information about memory, network, or number of CPUs. Our main contribution here is the release of a dataset representative for a new type of workload, that is, business-critical jobs in cloud datacenters.

**Workload characterization** Table 3.1 summarizes the comparison of previous studies with our work. Overall, our study is derived from an averagely-sized dataset, but focuses on a different workload, and includes a more comprehensive resource view (four types of resources, including the rarely studied disk and network I/O). Our study also conducts a detailed study of both requested and used resources, something that most public datacenter-studies are lacking. We have already compared throughout this work, whenever possible, the results obtained in this work with results from previous studies.

Closest to our work, Reiss et al. [17] analyze the workloads of Google. They use a dataset that is limited in comparison to ours; notably, it does not cover disk and network I/O. Because the Google workloads do not match the profile of business-critical applications, we observe significantly different results. For example, in the Google trace, the actual workload is relatively stable, whereas our results indicate that CPU and memory workloads are very unpredictable for business-critical applications; we have indicated other differences throughout this work.

Also related to our work, Di et al. [19] analyze the workloads of Google (with the same dataset limitations as Reiss et al. [17]) and compare them with Grid/HPC systems regarding job length and host load. Mishra et al. [20] propose a workload classification approach and apply it to a four-days trace from a Google datacenter, and Gmach et al. [21] analyze workload demands in term of number of CPUs from an HP datacenter.

Other types of datacenter workloads are complemented by our study: Chen et al. [22] analyze MapReduce traces from Yahoo and Facebook regarding the input/output ratio, job count, job submission frequencies, etc.; Guenter et al. [23] analyze the workload traces from Microsoft’s Live Messenger, Azure, and a shared computing cluster; and Chen et al. [24] analyze the workloads of login rates and connection counts in Microsoft’s Live Messenger cluster.

#### 2.4.2. Related Work on Workload Scheduling

Workload scheduling is a well established research domain that attracts a lot of attention in recent years. Especially the sub domain of distributed systems, big-data systems, and cloud/grid computing scheduling attracts a lot of attention. The taxonomy presented next is part of a literature study we conducted: V.S. van Beek, *Survey on resource management in datacenters*, TU Delft coursework, 2014. From this report the chapter on workload scheduling can be found in Appendix A.

Figure 2.5 depicts the scheduling taxonomy, which includes four main categories in which advances in scheduling techniques regarding resource scheduling in datacenters can be categorized. We describe the four categories in more depth in sections A.1.2 until A.1.2.
2. BACKGROUND

Figure 2.5: Scheduling taxonomy.

**Specialized Scheduling**
Scheduling resources in datacenters can be done at different levels in the execution stack (see Figure 2.3). Usually the higher in the stack the decisions are made the more information is available of the actual context of the workload and the workload itself. Specialized scheduling focuses on finding scheduling policies for specific types of workloads (workflows, parallel jobs, and sequential jobs). In many cases the more information there is available the better the scheduling can be performed. So specialized scheduling mechanisms can often be found high in the execution stack (software level). This does however not mean that these policies cannot make decisions about lower levels. For instance if information is available of what is exactly running on a VM this can help to better provision VMs to hardware.

**Meta Scheduling**
Because many datacenters run a wide variety of different workloads the chance of finding a policy that works for all of them is very small. Therefore work has focused on finding ways to deal with this problem. One of the directions of research is portfolio scheduling. The portfolio scheduling model works with a set of policies, from this set of policies the best policy is chosen for every subset of the total workload. In this way less complex policies can be created that work well for subsets of the total workload without having to question whether the policy also works well for all other workloads.

**Scheduling Single-tier Applications**
Work on model based scheduling focuses on finding workload models and using the insights that these models provide to optimize scheduling future workloads. These workload models are also often used to test scheduling policies in simulators. Different modeling approaches (e.g., analytical modeling, and statistical modeling) are used to develop workload models.
SCHEDULING MULTI-TIER APPLICATIONS

Multi-tier applications often have two characteristics (multicollinearity effect, and very dynamic load patterns) that are very specific for this type of setups. Because web-applications are very often hosted on multi-tier systems (often two-tier: web server and database server) lots of research has been done on this topic. This results in a variety of approaches that use different modeling strategies (queue models, prediction based models, and analytical models) to achieve effective dynamic resource allocation for multi-tier applications.
CHARACTERIZATION OF BUSINESS-CRITICAL WORKLOADS

As part of the development process of any scheduling mechanism, it is important to understand the characteristics of the workload to schedule. In this chapter we characterize business-critical workloads, using as validation data as observed in the Bitbrains infrastructure.

This chapter answers the first research question.

**What are the workload characteristics of Bitbrains managed clusters?**

We refine this research question into 4 sub-questions:

1. How to model VM workloads with multiple resource requirements?
2. Are there correlations in resource usage between different resource types (if so, how can these correlations help with making placement decisions)?
3. Is there a difference between enterprise workloads and known scientific/grid workloads (if so, what can we learn from these differences)?
4. Which are the groupings of VMs with similar resource usage (and, if they can be found, how can this help with workload placement strategies)?

The characterization of business-critical workloads presented in this chapter forms the basis for an article accepted in IEEE/ACM CCGRID 2015 (see Section 1.3).

Business-critical workloads, are comprised of applications that have to be available for the business to not suffer significant loss. They 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. Business-critical 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. We qualify for the first time, in this work, how significantly different, business-critical workloads are from the applications that are running in datacenters used by Google’s web search/services [17], and by Microsoft’s Messenger, shared cluster, and Azure [23] datacenters.

A typical mid-size datacenter hosting business-critical workloads is managed by Bitbrains, from which we collect business-critical workload traces. Bitbrains is a service provider that specializes in managed hosting and business computation for enterprises. Customers include many major banks (ING), credit card operators (ICS), insurers (Aegon), etc. For example, a customer would request a cluster of compute nodes to run a financial risk calculation. The following requirements would come with this request: data-transfers between the customer and the datacenter via secure channels, compute nodes leased as virtual machines (VM) in the datacenter that deliver predictable performance, and high availability for running business-critical simulations.
Although actual data and knowledge about workload characteristics are often beneficial for datacenter operation, remarkably few workload traces are publicly available or have even been publicly characterized.
### Table 3.2: Business-critical workload traces collected in this work.

<table>
<thead>
<tr>
<th>Name of the trace</th>
<th># VMs</th>
<th>Period of data collection</th>
<th>Storage technology</th>
<th>Total memory</th>
<th>Total cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>fastStorage</td>
<td>1,250</td>
<td>1 month</td>
<td>SAN</td>
<td>17,729 GB</td>
<td>4,057</td>
</tr>
<tr>
<td>Rnd</td>
<td>500</td>
<td>3 months</td>
<td>NAS and SAN</td>
<td>5,485 GB</td>
<td>1,444</td>
</tr>
<tr>
<td>Total</td>
<td>1,750</td>
<td>5,446,811 CPU hours</td>
<td></td>
<td>23,214 GB</td>
<td>5,501</td>
</tr>
</tbody>
</table>

Moreover, the few existing examples, albeit seminal, are not comprehensive and, because of their data source, may not be representative for the cloud datacenter industry in general. Table 3.1 (which we will discuss in detail in Section 2.4.1) presents an overview of several of the highest-cited studies of cloud workload traces. Overall, the traces originate from Google, Microsoft, and other giant datacenter operators (column TS), and represent workloads that may be typical for the MapReduce and other operations specific to these companies (column Workload). We also observe that few studies include information about requested resources, and rarely include network and disk information at all.

To address paucity of data and knowledge about datacenter workloads, in this work we aim to characterize the workload of a distributed datacenter servicing enterprise customers with business-critical applications. Towards this end, the main contribution of this chapter is four-fold:

1. We collect long-term and large-scale workload traces from a distributed cloud datacenter (Section 3.7). The traces include information about CPU, memory, disk I/O, and network I/O, for both requested and used resources.
   
   We make available these traces (Section 1.3), which are representative for business-critical applications running in cloud datacenters, through the public Grid Workloads Archive [9].

2. We analyze the basic statistics of the requested and actually used resources (Section 3.3). We report the basic statistics, such as quartiles, mean, and standard deviation. We also contrast the basic statistics of business-critical traces with those of parallel production environments, grids, and the search and data-mining workloads of Google, Microsoft, etc.

3. We conduct a correlation study to identify possible relationships between different resources (Section 3.4). We also contrast the results with results of previous datacenter studies.

4. We investigate the time-patterns occurring in the resource consumption (Section 3.5). Specifically, we investigate the peak to mean ratio in resource usage, which we compare with previous datacenter data, and conduct an autocorrelation study of each of the recorded characteristics.

### 3.2. Dataset Collection and Method of Characterization

In this section, we introduce two traces representative for business-critical workloads collected from a distributed cloud hosting datacenter. We also present a method for characterizing such traces.

#### 3.2.1. Collected Traces

From the distributed datacenter of Bitbrains, we collect two traces of the execution of business-critical workloads. For this we use the monitoring and management tools provided by VMware, such as vCloud suite\(^1\). For each trace, the vCloud Operation tools record 7 performance metrics per VM, sampled every 5 minutes: the number of cores provisioned, the provisioned CPU frequency, the CPU usage (average usage of CPU over the sampling interval), the provisioned memory capacity, the actual memory usage (the amount of memory that is actively used), the disk I/O throughput, and the network I/O throughput. Thus, we obtain traces that cover both requested and actually used resources, for four resource types (CPU, memory, disk, and network).

We collect between August and September 2013 two traces, whose overview we present in Table 3.2. Combined, the traces accumulate data for 1,750 nodes, with over 5,000 cores and 20 TB of memory, and operationally accumulate over 5 million CPU hours in 4 operational months; thus, they are long-term and large-scale time series. The first trace, fastStorage, consists of 1,250 VMs that are connected to fast storage area network (SAN) storage devices. The second trace, Rnd, consists of 500 VMs that are either connected to the fast SAN devices or to much slower Network Attached Storage (NAS) devices. The fastStorage trace

\(^1\)For more details, we refer to the official metrics documentation: [https://www.vmware.com/support/pubs/vcops-pubs.html](https://www.vmware.com/support/pubs/vcops-pubs.html)
includes a higher fraction of application servers and compute nodes than the Rnd trace, which is due to the 
higher performance of the storage attached to the fastStorage machines. Conversely, for the Rnd trace we 
observe a higher fraction of management machines, which only require storage with lower performance and 
less frequent access.

The two traces include a random selection of VMs from the Bitbrains datacenter, using an uniform distribution 
for the probability of selecting each VM. This is motivated by the need to guarantee the absolute anonymity 
of individual Bitbrains customers and to not reveal the actual scale of the Bitbrains infrastructure. A similar 
process is used by related work presenting the workloads of Google [17, 19], where the anonymization is 
achieved through a normalization of resource scales and by a selection of only a part of the infrastructure; in 
contrast, our study is more revealing, in that it presents the full characteristics of the virtualized resources.

Our traces do not include data about arrival processes, which in a cloud datacenter could be used to 
describe the lifetime of user jobs or of VMs. We investigate in this work resource consumption, which replaces 
the notion of user jobs with resource usage counters (this also protects the anonymity of Bitbrains’ users and 
is in line with the approach taken many previous studies [17, 19]). For VMs, business-critical workloads often 
use the same VMs for long periods of time, typically over several months. Thus, we do not have a proper 
arrival process to report on (the VMs we study run throughout the duration of our traces).

### 3.2.2. Method for Workload Characterization

We conduct in this work a comprehensive characterization, of both requested and actually used resources, 
and using data corresponding to CPU, memory, disk, and network resources. Although VMs may change 
frequency during the trace, the chance of this happening is rare in our trace (under 1%), so we show only 
the initial configuration of each VM present in our traces.

For the statistical characterization we use in this study three main statistical instruments\(^2\): basic statistics, 
correlations, and time-pattern analysis. For the basic statistics, we report the min and the max, the quartiles, 
the mean and the standard deviation (SDev), and the unitless “Coefficient of variation” (CoV, defined as the 
ratio of standard deviation and mean). We also report the cumulative distribution function (CDF) of the 
values observed for all VMs, and for the CoV observed per VM (a measure of *dynamicity* that extends previous 
work [17]).

### 3.3. Characterization Using Basic Statistics

In this section, we analyze in turn the requested resources, the used CPU and memory resources, and the 
used disk and network resources. Understanding the basic statistics can lead to interesting insights into 
the operation of the datacenter and into the structure of business-critical workloads, and can help create 
benchmarks and tune resource-management approaches. Table 3.3 summarizes the results, which are further 
analyzed in this section. For Bitbrains, unless otherwise specified we present here only results obtained for 
the fastStorage trace; for Rnd results, which are very similar (for example, see Figure 3.1a), we refer to our 
technical report [34]. The main findings are:

1. Over 60% of the VM requests are for no more than 4 CPU cores and 8 GB of memory (Section 3.3.1). Q3 
of the number of requested CPU cores is 4, which indicates indicates that business-critical applications

---

\(^2\)For details, we refer to the free Statistics Textbook: [http://www.statsoft.com/Textbook](http://www.statsoft.com/Textbook)
3. Characterization of Business-Critical Workloads

(a) CDF of the number of requested CPU cores. (b) CDF of the amount of requested memory.

Figure 3.1: CDF of the amount of requested CPU cores and memory.

The resource usage for most VMs is dynamic. The mean CoV for resource usage range from around 1 to over 20. The lowest CoV is observed for CPU and memory—CoV values under 5 (Section 3.3.2).

3. On average, VMs read 4 times more than write, and use the network to send as much as they receive (Section 3.3.3).

3.3.1. Requested Resources

In this section, we analyze the requested resources (only CPU and memory, as disk and network do not record such requests). We find that VMs in our traces require on average similar amounts of CPU cores as typical grid workloads, that most of the VMs have modest requirements for CPU cores (at most 4) and allocated memory (at most 8GB), and that power-of-two requests are common.

First, we compare the CPU characteristics of VMs supporting business-critical workloads (rows labeled Bitbrains in Table 3.3) and of representative traces from grid and parallel production environments. The rows including “CPU cores” in Table 3.3 list the number of CPU cores requested (and reported as used by all resource managers) in these workloads: fastStorage and Rnd representing business-critical workloads; the DAS2, Grid5000, and NorduGrid datasets representing grid workloads [9]; and the CEA CURIE and LLNL Atlas datasets representing production parallel workloads [16]. If we view the VM as the unit of submitting workload, our workloads requires, on average, slightly more CPU cores than production grids (NorduGrid) and slightly less cores than experimental grids (DAS2 and Grid5000), but significantly less CPU cores than the parallel workloads. This may indicate that business-critical applications do not fully adopt HPC techniques.

We further characterize the requested resources, in terms of number CPU cores and bytes of memory allocated to each VM. Figure 3.1a shows the cumulative distribution function (CDF) of the number of CPU cores requested per VM. For both the number of CPU cores and the amount of memory our results show that a large percentage (more than 60%) of the VMs have low requirements (2 or 4 CPU cores for our two traces, and less than 8 GB of memory). Most of the VMs (over 90%) use power-of-two cores. Other studies [13] show the power-of-two scale-up behavior, which seems to be historically an artifact of the architecture of parallel architectures and algorithms. VMs in our datasets use from 1 up to 32 cores (small-scale HPC), but over 95% of the VMs use at most 8 cores, and over 85% of the VMs use at most 4 cores. On average, VMs in the Rnd dataset use slightly fewer cores, which we ascribe to the higher density of management VMs in the Rnd trace—typically, management VMs require only 1, rarely more cores.

Regarding memory requirements, we observe similar patterns as for CPU requirements. Figure 3.1b shows the CDF for the requested memory of each VM. Memory is often provisioned in power-of-two quantities (around 90% for memory). For the fastStorage dataset, the requested memory can range from 1GB to 512GB per VM, but most VMs use a relatively small amount of memory: over 95% of the VMs use at most 32 GB of memory, and over 70% of the VMs use at most 8 GB of memory. The VMs in the Rnd dataset demand slightly less memory than in the fastStorage dataset, which we ascribe again to the difference in management VM density—typically, management VMs use 1GB or less memory.
3.3. CHARACTERIZATION USING BASIC STATISTICS

3.3.2. CPU AND MEMORY USAGE

In this section, we analyze the CPU and memory resource usage, for which we report both the CDF observed across all VMs, and the CDF of CoV observed per VM. We find that CPU usage is low on average and can be dynamic, and much lower (around 10% for most VMs) than the requested CPU bandwidth. We also find that memory usage is even lower on average but less dynamic than CPU usage.

We study first the CDF of CPU usage, across all VMs. Figure 3.2 (top) shows the CDF, computed across all VMs, from their observed CPU usage—first-quartile (Q1), mean, third-quartile (Q3), and maximal (Max) CPU usage—and from their requested CPU bandwidth (computed as the product of the number of CPU cores and the bandwidth of each core, e.g., 4 x 2.6 GHz for 4 cores at 2.6 GHz each). For most (about 80%) of the VMs, the mean CPU usage (curve “mean” in Figure 3.2 (top)) is lower than 0.5 GHz and lower than 10% of the requested CPU bandwidth. Only for less than 5% of the VMs, the mean CPU usage is higher than 50% of the requested CPU bandwidth. About 50% of the VMs have a maximal usage (curve “max” in Figure 3.2 (top)) lower than 1.3 GHz. Only 30% of the VMs have a maximal usage higher than 2.8 GHz. These observations suggest that for most VMs the usage is low most of the time.

We study next the CDF of the CoV in the observed CPU usage, observed per VM. As Figure 3.2 (bottom) shows, the CoV for CPU usage is lower than 0.5 for half of the VMs; however, there is still a significant amount (about 20%) of VMs whose CoV for CPU usage is higher than 2—the CPU usage of these VMs is dynamic and unpredictable.

We now study the CDF of memory usage, across all VMs. We construct Figure 3.3b similarly to Figure 3.2a, but with data about used memory. We find that the memory usage is low: on average, 80% of the VMs use less than 1 GB of memory, and most (about 80%) of the VMs have maximal memory usage lower than 8 GB of memory. In Figure 3.3, the large gap between the “mean” and the “max” curves indicates that the peak memory usage of each VM is much higher than its average usage; we investigate this in more detail in Section 3.5.1.

Similarly to our study for CPU usage, we investigate next the CDF of the CoV in the observed memory usage, per VM. As Figure 3.3b shows, the memory usage is less dynamic than CPU usage: about 70% of VMs...
3. Characterization of Business-Critical Workloads

Figure 3.4: CPU and memory usage over time.

Figure 3.5: Disk read usage: CDF for all VMs, and CDF of CoV observed per VM.

We find that CPU utilization is higher than memory utilization, which is the opposite of the finding of Di et al. [19] for the Google trace. This finding is consistent with our earlier observation that business-critical workloads are more in line with grid workloads, where CPU utilization is the typical bottleneck, and indicates that different strategies for datacenter efficiency may be needed for business-critical workloads, in contrast to Google-like search and services workloads.

3.3.3. Disk and Network Usage

Similarly to Section 3.3.2, in this section we analyze the disk and network resource usage, for which we report both the CDF observed across all VMs, and the CDF of CoV observed per VM. We study, in turn, the disk read and write usage. For network usage we find similar results as for disk usage, because of space limitations graphs on network usage can be found in our technical report [34]. We find that most VMs have bursty disk and network accesses.

We study the the CDF of disk read usage, across all VMs, which we depict in Figure 3.5a. We find that most of the VMs only read sporadically: about 95% of the VMs perform three-quarters of their disk reads (“Q3” curve) at less than 0.1 MB/s. The mean value and especially the maximal value of disk reads of most VMs is much higher than the Q3 value, which indicates that disk reads are bursty. The CDF for CoV of disk reads is plotted in Figure 3.5b. The disk read usage is much more dynamic than the CPU usage: only 15% of VMs have their disk-read CoV under 1, and about 50% of the VMs have their disk-read CoV higher than 2. This may be due to application behavior, e.g. backup tools may act periodically, financial modeling tools read large volumes of financial data into memory at the start of simulations, etc.

Similarly to disk reads, we study now the disk write usage. The results, which we depict in Figure 3.6a, are similar in trend for disk reads and writes: most of the VMs do not write most of the time, but some VMs show very high peak disk write usage. On average, each VM’s disk write usage is about 0.1 MB/s, which is about one fourth of the disk read usage (0.4 MB/s). Comparing to disk reads, we observe that disk writes are less
3.4. CHARACTERIZATION OF CORRELATIONS

In this section, we analyze the pair-wise correlation between the requested resources (e.g., requested CPU and memory), the correlation between the request demand and the actual usage, and the pair-wise correlation between used resources (e.g., between the CPU and memory usage). The main findings are:

1. CPU and memory are strongly correlated for requests (Section 3.4.1), but much less correlated for usage (Section 3.4.2).

2. Request and use are very weakly correlated (Section 3.4.1).

3.4.1. CORRELATION OF REQUESTED RESOURCES

In this section, we investigate the correlation between the two types of requested resources, CPU and memory, and find a strong correlation between them. We also investigate the correlation between requested and used resources, and find a very weak correlation.

For the fastStorage dataset, the PCC and SRCC between the number of CPU cores and memory are 0.81 and 0.90, respectively. For the Rnd dataset, the PCC and SRCC are 0.82 and 0.85, respectively. This indicates that VMs with high values for the requested CPU tend to also have high values for the requested memory, especially for VMs (in the fastStorage dataset). We confirm this result through an interview with the engineers of Bitbrains, confirming that Bitbrains typically maps either 2 GB or 4 GB memory to a CPU core, depending on the physical CPU-to-memory ratio of the underlying physical infrastructure. For memory-intensive workloads, they set the memory to 16 GB per core. At the other extreme of the CPU-to-memory ratio, small VMs (1 GB or less memory) are typically management VMs that are needed to operate the customer environments.

For both the fastStorage and the Rnd datasets, the requested and the used resources are weakly correlated. This is indicated visually by the plots of Figures 3.2, 3.3, and 3.5. We analyze here the data for the former two, in turn; the analysis for the latter reveals similar trends.

Figure 3.2 shows that: about 80% of VMs have an average CPU usage lower than 10% of their allocated CPU bandwidth, and less than 5% of the VMs have a mean CPU usage that is higher than 50% of the allocated CPU bandwidth. About 50% of the VMs have a maximal usage lower than 20% of allocated CPU bandwidth and only 30% of the VMs have a maximal usage higher than 70% of allocated CPU bandwidth. These observations suggest that most of the VMs’ CPUs are idle most of the time.

Figure 3.3 shows that: 60% of the VMs have an average memory usage lower than 10% of the requested amount of memory, the most memory-intensive VM use only 80% of its requested memory; only 14% of the VMs have a peak memory usage that reaches their requested amount, about 50% of the VMs have a peak dynamic, as shown in Figure 3.6b.

Similarly to disk behavior analysis, we study the network usage, expressed in terms of received and transmitted data. Most of the VMs mean amount of data received or transmitted over the network is low. About 80% of the VMs receive less than 30 KB/s and transmit less than 10 KB/s. The large gap between the max and the other percentiles, as observed per VM, indicates the bursty nature of network traffic. The amounts of both the received and transmitted are much more dynamic than the CPU usage.
memory usage lower than 50% of the requested memory. This indicates that the memory utilization is low most of the time.

Similarly, the bursty nature and the poor visual correlation between the curves depicted in Figure 3.4, indicates poor correlation between requested and used resources.

3.4.2. Correlation of CPU and Memory Usage

We analyze the correlation of CPU usage and memory usage, for which we report an average correlation. Because both CPU and memory usage vary over time, we also report CDFs and PDFs of the correlation observed over time, per VM. We find strong correlation between high CPU and memory usage, that is, VMs that exhibit high CPU usage are very likely to also exhibit high memory usage. However, the temporal correlation is much weaker: it is less likely that VMs exhibit high CPU and memory usage at the same time. This gives, for the future, interesting opportunities to host business-critical workloads more efficiently inside the datacenter.

We analyze first the average correlation, that is, the correlation between mean CPU and mean memory usage. For the fastStorage dataset, the PCC and SRCC of the mean CPU usage and mean memory usage, per VM, are 0.83 and 0.84, respectively. For the Rnd dataset, the PCC and SRCC are 0.72 and 0.83, respectively. This indicates that VMs with high CPU usage tend to have high memory usage. Ren et al. [30] report that for the Taobao system the PCC between CPU usage and memory usage is 0.76, which falls within the same range as our overall result.

Although the average correlation indicates strong correlation, we observe that the temporal nature of both CPU and memory usage requires more in-depth analysis. We thus report CDFs and PDFs of the correlation observed over time, per VM, e.g., we collect data about the CPU-memory correlation for each sampling point (every 5 minutes, as indicated in Section 3.2.1), and we analyze the CDF and the PDF of this dataset.

In Figure 3.8 we show the distribution of PCC and SRCC for CPU usage and memory usage. The mean PCC for CPU and memory usage for the fastStorage dataset is 0.4, this is much lower than Ren et al. [30] report and also much lower than we found if we compare CPU and memory required or used on average.

From both the results of the correlation analysis and from the analysis of resource usage in Section 3.3.2, it follows that there is a big discrepancy between requests and actual usage. A similar result was found by Reiss et al. [17] for the Google trace, where only about 50% of the requested resource are actually used by jobs. As we have shown in Section 3.4.1, this ratio is even lower for business-critical workloads. This implies, for business-critical workloads, excellent opportunities for efficient resource management approaches exist.

3.4.3. Correlation of CPU and Other Resource Usage

To get a better understanding of correlations between the usage of different resource types, we conduct a comprehensive analysis of all the possible pair-wise combinations (as throughout this work, the resource types considered are CPU, memory, disk read, disk write, network transmit, and network receive). We discuss in the following a selection of these results, depicted in Figures 3.9–3.10b (all similar in construction process.
3.4. Characterization of Correlations

(a) Correlation between CPU usage and network receive usage. (b) Correlation between CPU usage and network transmit usage.

Figure 3.8: Correlation between CPU usage and network receive usage, and network transmit usage. For each group of two plots: CDF and PDF of PCC over time; CDF and PDF of SRCC over time.

(a) Correlation between CPU usage and disk read usage. (b) Correlation between CPU usage and disk write usage.

Figure 3.9: Correlation between CPU usage and disk read usage, and disk write usage. For each group of two plots: (top) CDF and PDF of PCC over time; (bottom) CDF and PDF of SRCC over time.

and structure to Figure 3.8). The complete results are presented in our technical report [34]. We find low correlations between CPU usage and the usage of other resource types, and even lower correlation between disk and network resources. We also find that about 25% of the VMs in our study exhibit strong network transmit and network receive correlation, but either strongly positive or strongly negative; the remaining VMs exhibit the low correlation trend we have observed for other resources.

We start by investigating the correlation between the usage of CPU and of all the other resource types. We find that the correlation between CPU usage and disk write, disk read, network receive, and network transmit is very low. From both Figure 3.8 and Figure 3.9, we observe that the majority of the pair-wise correlations of the type CPU usage-other resource usage are between 0.0 and 0.5. These values are much lower than, for example, the correlation values between 0.8 and 0.9 found for in Section 3.4.1.

Next, we investigate potential correlations between storage and network resources. Figure 3.10a depicts the correlation between disk read and network transmit. The observed correlations are even lower than the what we observe for CPU usage, and the usage of disk and network resources. This observation holds for all other pair-wise correlations of disk and network usage.
3. CHARACTERIZATION OF BUSINESS-CRITICAL WORKLOADS

(a) Correlation between disk read usage and network transmit usage.

(b) Correlation between network receive usage and network transmit usage.

Figure 3.10: For each group of two plots: (top) CDF and PDF of PCC over time; (bottom) CDF and PDF of SRCC over time.

Last, we investigate the correlation between the two network-related resource types, network receive and network transmit. Figure 3.10b shows the correlation between network transmit and network receive. From the figure, we observe that for the majority of the VMs the correlation between sending and receiving network traffic is very low. However, about 16% of the VMs have a strong positive correlation between sending and receiving network traffic, and about 8% of the VMs have a strong negative correlation between sending and receiving network traffic. We conclude that network receive and network transmit have more diverse pattern of correlations than other resources.

3.5. CHARACTERIZATION OF TIME-PATTERNS IN RESOURCE USAGE

In this section, we analyze the time patterns of resource usages. Understanding the time patterns of the resource usages can help to build smart predictors that estimate upcoming resource usage, and can lead to improved datacenter efficiency. The main findings are:

1. The aggregate resource usage of the VMs fluctuates significantly over time.
2. The peak CPU resource usage is 10–100 times higher than the mean (Section 3.5.1).
3. CPU and memory resource usage can be predicted in short-term.
4. The usage of disk I/O and network I/O show daily patterns, for the fastStorage dataset. (Section 3.5.2).

3.5.1. PEAK VS MEAN RESOURCE USAGE

In this section, we analyze how dynamic are business-critical workloads, and contrast our findings with previously described workloads. To this end, following [17, 35] we study the peak and mean resource usage, and their ratio, over time. We report both hourly and daily intervals, for all the resources investigated in this work. (Previous studies report this value of intervals that range from 30 seconds [23] to 1 day [17], which makes it difficult to compare results across studies.) The results for CPU bandwidth, for disk read, and for amount of network-transmitted data are summarized in Figure 3.11. Overall, we find that business-critical workloads are much more dynamic than most previously described datacenter workloads, and more in line with the volatile grid workloads. This emphasizes the opportunity to design more efficient resource management approaches, such as the dynamic change of the number of active physical resources underlying the leased VMs.

We begin with a focus on CPU usage. Figure 3.11b shows the peak and mean CPU usage, and their peak-to-mean ratio, per hour and per day. CPU usage fluctuates significantly overtime. The daily peak usage can be 10 to 100 times higher than the daily mean usage. This phenomenon is commonly observed in other related workloads: in the Google trace (daily peak-to-mean ratio: 1.3), in the Microsoft Azure trace (15 minute
3.5. CHARACTERIZATION OF TIME-PATTERNS IN RESOURCE USAGE

(a) Legend for figures 3.11b, 3.11c, and 3.11d.

(b) Peak to Mean resource usage, over time: CPU

(c) Peak to Mean resource usage, over time: Disk read.

(d) Peak to Mean resource usage, over time: Network transmit.

Figure 3.11: Peak to Mean resource usage, over time: CPU, Disk read, and Network transmit. For each group of two plots: (top) hourly data; (bottom) daily data.

sample, peak-to-mean ratio 1.7), and in the Microsoft Messenger trace (30 second sample, peak-to-mean ratio range from 2.5 to 6.0). The peak-to-mean ratios observed in business-critical workloads are even higher than the ratios observed in these traces. Josup et al. [35] analyze 5 grid traces and find hourly peak-to-mean ratios of up to 1,000:1. Similarly, Chen et al. [18] analyze 7 workload traces (from Facebook and Cloudera) and find peak-to-mean ratios ranging from 9:1 to 260:1. These ratios are more in line with the ratios we observe.

Similarly to CPU usage, we analyze the other resources, and find similarly high or even higher peak-to-mean ratios. Figure 3.11c shows the peak-to-mean ratio for disk-read usage. Both the hourly and daily ratios are much higher than the ratios observed for CPU usage: we observe 1:1000 and even 1:10,000 ratios. We find similar numbers for disk-write usage [34]. Figure 3.11d depicts the peak-to-mean ratio for network-transmit usage. We find the ratios in the same order of magnitude as for disk usage (including the occasional 1:10,000). We find similar numbers for network-receive usage [34].

3.5.2. TIME PATTERNS THROUGH AUTO-CORRELATION

In this section, we investigate the presence of time patterns in the usage of resources observed for business-critical workloads. To this end, we conduct an analysis using the auto-correlation (ACF) tool. For all resources, we identify high ACF for small lag, which indicates predictable resource usage in the short term (that is, a few hours). We also find strong daily patterns in disk and, somewhat less, in network activities.
3. CHARACTERIZATION OF BUSINESS-CRITICAL WORKLOADS

We analyze the ACF for all types of resource usage, for lag values from 0 hours up to 1 month, with 1-hour step. Figure 3.12 depicts the ACF values for three types of resource usage: CPU, disk read, and network transmit; results for the other resource usage types are presented in [34]. The ACF values for the first 10 lags ranges, for all resource usage types, from 0.7 to 0.8, which is high and indicates strong auto-correlation. This indicates that, for all resource usage types, the resource usage is predictable in the short-term (up to a few hours).

For disk read (Figure 3.12 (middle)), the ACF curve has local peaks at lag multiples that correspond to days; this indicates that the disk read has a strong daily pattern. We also observe that the disk write (not shown) and the network I/O (Figure 3.12 (bottom)) follow daily patterns, albeit less pronounced for the network I/O.

3.6. THREATS TO VALIDITY

In this section we list possible threats to the validity of this work and the measures we took to mitigate them.

Timelessness of the dataset: As datasets age, they may become unrepresentative. Our dataset is not only more recent than that of previous work, but based on our experience, rather stable over time—business-critical applications have changed relatively little in the past few years. Thus, our datasets could still be relevant for the next 5-10 years.

Dataset size: Unrepresentative datasets can lead to misleading characterization. Compared to other workload traces surveyed in this work (see Table 3.1 and Section 2.4.1), but not necessarily public, our traces are of medium size, in both the period and the number of nodes they cover. Our traces are also of medium size in comparison with the public traces collected from parallel [16] and grid [9] environments. Thus, our results suffer from this set as much as results of studies derived from other traces in the field. Because this information is not publicly available, we can only argue that the datacenter size we considered in this work is more common in the industry as a whole than the Google, Facebook, and Microsoft datacenters.

Data collection tools: The data collection tool can cast doubts on the validity of the dataset. We rely on the tools provided by VMware, which are currently used by thousands of medium and large businesses, and thus can be considered a de-facto industry standard.

Trustworthy analysis: Mistakes in analysis occur often, in many fields of applied statistics. To alleviate this problem, in lack of a validation study conducted by a third-party laboratory, our statistical analysis is conducted by two of the authors, independently; the results have matched fully.
Collaboration with an industry partner: Analysis in which a participant has a vested interest could lead to biased results. To alleviate this problem, in lack of a multi-party industry consortium, we have collected and analyzed two traces. We note that the studies presented in Section 2.4.1 have the same limitation, but most rely on a single trace.

3.7. DATA AVAILABILITY
The two traces will be made public prior to CCGrid 2015, as part of the Grid Workloads Archive [9].

3.8. SUMMARY
Understanding the workloads of cloud datacenters is important for many datacenter operations, from efficient capacity planning to resource management. In this work, we collect from a distributed datacenter hosting business-critical workloads 2 large-scale and long-term workload traces from 1,750 virtual machines. We analyze these traces for both requested resources and actual resource usage, in terms of CPU, memory, and disk I/O and network I/O; we also compare these findings with previous studies of workloads from search datacenters, parallel and grid environments, etc.

Our main findings, as reported in this chapter and detailed in a technical report [34], are:

1. More than 60% of the VMs use less than 4 cores and 8 GB of memory.
2. There is a strong positive correlation between requested CPU and memory.
3. Resource usage is low, under 10% of the requested resources, and the correlation between requested and used resources is also low.
4. Peak workloads can be 10–10,000 times higher than mean workloads, depending on resource type.
5. The CPU and memory resource usage is often predictable over the short-term. Disk and network I/O follow daily patterns.
In this chapter, we describe the design of a portfolio scheduler for business-critical workloads running in a multi-cluster, multi-datacenter setting. In addition to the high-level design, we detail the most important design aspects. This chapter answers the second research question.

**How to adapt the concept of portfolio scheduling to the placement of VMs in a multi-cluster multi-datacenter setting?**

We refine this research question into 3 sub-questions:

1. How to measure the quality of a given VM placement strategy?
2. How to simulate a multi-datacenter setting, to test placement policies?
3. How to incorporate workload models into the BBsimulator?

The design presented in this chapter forms the basis for an article that is currently under submission in a special edition (Self-Aware and Self-Expressive Computing Systems) IEEE Computer journal. In the article we present the architecture under the name *Mnemos*, referring to the Greek goddess of memory and remembrance (*MNEMOSYNE*).

### 4.1. Overview

We describe the design of the portfolio scheduler, along the following main design steps: requirements, high-level design, and detailed design decisions. To gather the requirements, we conduct interviews with Bitbrains engineers and use the results of the workload analysis.

Following the steps for designing a portfolio scheduler described in Section 2.3, we identify that an important component in our scheduler design is the datacenter simulator, which in the operation of a portfolio scheduler is the key to selecting the best scheduling policy at any given time. Every portfolio scheduler must adapt its simulator to the environment it tries to simulate. For the modern datacenter, this adaptation is non-trivial, which follows from the constraints discussed in Section 1.1.2, such as multi-optimization criteria, hierarchical datacenter structure, etc. For this work, we extend a previously existing simulator, the event-driven simulator DGSim [36], towards a datacenter simulator that can be used under the time constraints required by decision-making in the datacenter. BBsimulator incorporates the multi-cluster, multi-datacenter requirements and takes into account all the resources that are relevant to our research (CPU, memory, storage, and network). We describe the portfolio policies and the selection mechanism that we use to evaluate the results of the BBsimulator.
4.2. REQUIREMENTS FOR THE PORTFOLIO SCHEDULER

In this section we present the requirements that form the basis for the design of the portfolio scheduler for business-critical workloads. The requirements are derived from a requirements solicitation process that involved, e.g., interviewing Bitbrains engineers, detailed investigation of the infrastructure, and an in-depth analysis of business-critical workloads (see Chapter 3).

We describe next two main results of the requirements-solicitation process: the restrictions in Section 4.2.1, and the requirements in Sections 4.2.2.

4.2.1. RESTRICTIONS ON PROVISIONING VMs

There are restrictions that require attention during the development of a provisioning mechanism for VMs.

1. Memory allocation
   Memory is statically allocated. This means that oversubscription of memory is not possible. Clusters even have an operational limit lower than the total physical memory available for allocating memory to VMs. This limit ensures that daily operational tasks can take place, such as rebooting hosts when installing for instance updates.

2. CPU allocation
   CPUs cannot be oversubscribed by more than a factor 2x, because of this property load balancing for CPU loads is also important. For instance if one cluster is fully loaded for more than 100% and an other cluster has almost no load the end user would be better of if the load was balanced more evenly over the clusters.

3. Cluster load balancing
   Dynamic Resource Scheduling (DRS) is used for load balancing CPU load on cluster level. DRS is a mechanism that runs within the VMware management suite. DRS uses online migration techniques to load balance CPU and memory load over the servers within the cluster.

4. Storage allocation
   Similarly to memory allocation clusters have an operational limit for allocating storage space.

4.2.2. OTHER REQUIREMENTS

1. Load balancing
   Load balancing is the main requirement considering VM provisioning. Load balancing should be achieved at two levels, first cross cluster load balancing, and second in cluster load balancing. Load balancing can of course be considered on many different resource types (e.g., memory, network, cpu, IO, power, cooling, etc.). In the context of this project the main focus is on load balancing CPU, memory, storage, and network workloads.

2. Availability Zone Requirements
   From a customer perspective, certain systems need to be available at all times, a common technique to achieve high availability is to create an environment that runs the system on multiple physically separated servers. The the lowest form of high availability is to run the system on multiple physical servers within the same cluster. The highest form of availability that can be achieved is to run the system on servers in multiple datacenters. The scheduler has to be able to handle availability constraints set by engineers. Availability constraints are also referred to as anti-affinity constraints.

3. Affinity
   We consider in this work to two types of affinity: VM-hardware affinity and VM-VM affinity. Because of the heterogeneous nature of the infrastructure there are VMs that have specific hardware requirements. For instance VMs that run certain HPC workloads need to run on the specialized HPC servers because of performance reasons.
   The second type of affinity (VM-VM) has a lot to do with the network topology of the infrastructure, VM-VM affinity is only applicable to VMs that are part of a vCluster. Communication between servers within a clusters is obviously the fastest, and communication between servers in different datacenters is obviously the slowest. Therefore if VMs need high communication performance to communicate
amongst each other they need to be placed as close together as possible. Take for instance a system with an application server and a database server that are highly dependent and need a lot of communication, in such situations the VMs running the two servers need to be placed close together. An other setup that often has high communication performance requirements is a distributed Hadoop [37] cluster. The VMs running the Hadoop nodes should be placed close together to achieve high performance.

4. Provisioning Multiple VMs At Once

Often customer environments at Bitbrains involve multiple VMs a simple example would be a system that is split into two subsystems (a production environment and a test environment). Each of these environments could for instance exist of a web-server and a database server (running on separate VMs). If such an environment is deployed the provisioning mechanism should be able to handle the provisioning of several VMs at once. In the remainder of this work we refer to these sets of VMs as vClusters.

4.3. High-level Design of a Portfolio Scheduler for Business-critical Workloads

In this section, we present the design of our portfolio scheduler. We begin with a grand design, which is useful for the datacenter engineers that need to interact with our scheduler. We continue with the design of the portfolio scheduler itself, which is our main contribution in this work and is useful for the scheduling engineers.

We now present the grand design of our portfolio scheduler, in which we show how the portfolio scheduler fits in the operational model of a datacenter. In Figure 4.1 we depict this grand design. It shows on the top layer the user space, where users and engineers edit and use the system. The Apps component in the user space represents the software running on the VMs. In the middle layer we, find the portfolio scheduler and other components that are needed to operate a datacenter (e.g., VM manager, and system monitor). In the bottom layer, the grand design includes the physical infrastructure: datacenters, clusters, hosts, etc.

The datacenter engineers provide profile data for VMs, vClusters, and Apps. In these profiles, the engineers describe for example the number of CPUs per VM, and the workload pattern expected for the VM, and, in the vCluster, engineers can define affinity or anti-affinity requirements for VMs. The portfolio scheduler uses then the profile data to make a decision on where (on which cluster) to provision the requested VMs.

We now focus on the high-level design of the portfolio scheduler for business-critical workloads. In Figure 4.2 we present the overview of the scheduler. The numbers 1-6 represent the order of steps that take place during the process of provisioning VMs in the datacenters. The process starts with an engineer requesting the placement of one VM or a vCluster (set of VMs) (step 1). The scheduler than takes the vClusters and individual VMs from the queue following a first-in-first-out strategy (step 2). The vCluster or VM is send to the BBsimulator (described next) where the decision is made on which policy to use (step 3), the result of the chosen policy is the VM-cluster mapping (step 4). This mapping is than used to provision VMs on the actual physical clusters in the datacenters (step 5 and 6).

As in prior work in portfolio scheduling [7], we use simulation to evaluate the different policies. Each time when an engineer submits a vCluster, the BBsimulator is used to determine which of the policies is best for the given set of VMs. Figure 4.3 shows an overview of the BBsimulator. The BBsimulator uses workload traces and the current state of the datacenters (current VM mapping) to run simulations with the different provisioning policies (the policies are described in detail in Section 4.4.1). At the end of all the simulation runs, the results of all the policies are combined and, during an analysis step, the best performing policy is chosen automatically. Although the use of the simulator is not new, the BBsimulator significantly extends the state-of-the-art, in its focus, scheduling policies that it supports, multi-cluster, multi-datacenter requirements it supports, and the types of resources it supports (CPU, memory, storage, and network). A detailed discussion on how this selection process works can be found in Section 4.4.2. The final result of the simulation is a VM-cluster mapping for the VMs in the vCluster.

4.4. Detailed Design Decisions

We describe in this section a selection of the most important detailed design decisions made during the design of this portfolio scheduler. First, we describe the policies that are used by the scheduler to provision VMs during the simulation runs. We consider two broad groups of policies, for scheduling vClusters and for
Figure 4.1: Grand design, showing how the portfolio scheduler connects with the user space, the datacenter, and the datacenter operating layer (VM manager).

Figure 4.2: High-level overview of the Portfolio scheduler for business-critical workloads.
4.4.1. **Policies for Provisioning VMs to Clusters**
Starting from the scheduling requirements we have formulated in Section 4.2, we develop several scheduling policies for provisioning VMs to clusters. We make a distinction between policies for individual VMs and vClusters. The main reason for this is that the policies for individual VMs can be much simpler, because they are not required to take into account VM-VM affinity and anti-affinity requirements (see Section 4.2.2).

In the system VM-Hardware affinity is translated into a *cluster priority vector*. Each VM has such a property to express its preference of the type of clusters it can run on. VMs that are part of a vCluster also have a *VM exclusion vector*. This vector translates the VM-VM anti-affinity requirements of the VMs in the vCluster. To optimize for performance VMs are placed close together unless specified differently by the VM exclusion vector. Because of the nature of business-critical workloads (often high-availability is the key requirement) we design the scheduling policies in such that they prioritize VM-VM anti-affinity above all other requirements.

For other types of workloads (e.g., scientific computing) performance is more important than high-availability. For budget web-hosting companies optimizing for cost efficiency is more important than high-availability. The choice to prioritize VM-VM anti-affinity above all other requirements is specific to our project and would be different had we been presented with other workloads.

In Table 4.1, we provide detailed descriptions of the policies that are used by the scheduler for the placement of individual VMs. Besides a detailed description, we also provide an indication of the novelty of the policy. In Table 4.2 we, provide detailed descriptions of the policies that are used by the scheduler for the placement of vClusters.

4.4.2. **Method of Selecting the Best Policy After Simulation**
The selection of a policy is a three step process. The first two steps take place within the BBsimulator, and the third step uses the policy scores generated by the BBsimulator to make the final policy selection decision.

In Figure 4.3 we show the design of the BBsimulator. The BBsimulator contains a dataset *Simulation Results*, and a process *Result analysis* that contains the functionality for analyzing the result of the simulation run. For each vCluster or individual VM that is provisioned the BBsimulator uses all the available policies scheduling individual VMs. Second, we describe the selection process that chooses the active policy after the simulation runs have finished. Third, we describe how we create and use a latency model for emulating the effect of storage workloads on the storage layer.

![Overview of the BBsimulator at the core of our portfolio scheduler.](image)
### 4.4. Detailed Design Decisions

#### Table 4.1: In this table we provide descriptions of the policies for single VM placement. (Novelty is described with three letters: P: implementation of previous work, N: Novel work, A: adaptation of previous work.) The column Acro contains the acronyms of the policies.

<table>
<thead>
<tr>
<th>Novelty</th>
<th>Acro</th>
<th>Short description / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>RR</td>
<td><em>Round Robin policy</em>&lt;br&gt;This policy uses a round robin approach to provision VMs, RR respects the VMs cluster priority vector.</td>
</tr>
<tr>
<td>N</td>
<td>SBB</td>
<td><em>Standard before Bigmem policy</em>&lt;br&gt;This policy uses a round robin approach to provision VMs, RR respects the VMs cluster priority vector.</td>
</tr>
<tr>
<td>A</td>
<td>LML</td>
<td><em>Lowest moving average memory load policy</em>&lt;br&gt;This policy uses the moving average memory load per cluster and provisions the VM to the cluster with the lowest load. LML respects the VMs cluster priority vector.</td>
</tr>
<tr>
<td>A</td>
<td>LCL</td>
<td><em>Lowest moving average CPU load policy</em>&lt;br&gt;This policy uses the moving average CPU load per cluster and provisions the VM to the cluster with the lowest load. LCL respects the VMs cluster priority vector.</td>
</tr>
<tr>
<td>A</td>
<td>LSL</td>
<td><em>Lowest moving average storage load policy</em>&lt;br&gt;This policy uses the moving average storage load per cluster and provisions the VM to the cluster with the lowest load. LSL respects the VMs cluster priority vector.</td>
</tr>
<tr>
<td>A</td>
<td>LNL</td>
<td><em>Lowest moving average network load policy</em>&lt;br&gt;This policy uses the moving average network load per cluster and provisions the VM to the cluster with the lowest load. LNL respects the VMs cluster priority vector.</td>
</tr>
<tr>
<td>A</td>
<td>BB</td>
<td><em>Bitbrains lowest resource usage policy</em>&lt;br&gt;This policy uses the moving average CPU, network, and storage load per cluster and provisions the VM to the cluster with the lowest load. The importance of the three resource can be set with a parameter (in this work they are set to 1 for equal importance). BBpolicy respects the VMs cluster priority vector.</td>
</tr>
</tbody>
</table>

in turn to simulate the provisioning of the VMs in the datacenter. At the end of the simulation runs the simulator returns datasets with the VM mappings and the cluster resource utilization traces for each policy it has simulated (the *Simulation Results*). The *Result analysis* process uses the cluster resource utilization traces to calculate the policy scores, one score per policy. The scheduler uses all the *Result analysis* scores to decide which policy is the best policy for the given vCluster or VM. Next, we describe the three steps in more detail.

#### Step 1 Validate the VM cluster mapping:

We start by validating that the mapping suggested by BBsimulator based on the policy of a particular simulation run. The validation checks if all the requirements as described in Section 4.2 are met. During this validation step the simulator for instance checks if all the affinity and anti-affinity requirements are met.
<table>
<thead>
<tr>
<th>Novelty</th>
<th>Acro</th>
<th>Short description / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>FF</td>
<td><strong>First fit policy for vCluster</strong>&lt;br&gt;Iterate over the clusters and map each VM in the vCluster to the first cluster where the VM fits while still respecting the VM cluster priority vector.</td>
</tr>
<tr>
<td>N</td>
<td>TP</td>
<td><strong>Type priority policy for vCluster</strong>&lt;br&gt;Sort the clusters by type according to the VM cluster priority vector and by moving average CPU load.</td>
</tr>
<tr>
<td>A</td>
<td>LCLvC</td>
<td><strong>Lowest CPU load vCluster policy</strong>&lt;br&gt;VMs in the vCluster are first grouped together according to the VM exclusion vector, the groups are placed in different datacenters in clusters where the CPU load is the lowest. The LCLvC placement policy uses the VM cluster priority vector to determine which type of cluster to try first.</td>
</tr>
<tr>
<td>A</td>
<td>LMLvC</td>
<td><strong>Lowest Memory load vCluster policy</strong>&lt;br&gt;VMs in the vCluster are first grouped together according to the VM exclusion vector, the groups are placed in different datacenters in clusters where the memory load is the lowest. The LMLvC placement policy uses the VM cluster priority vector to determine which type of cluster to try first.</td>
</tr>
<tr>
<td>A</td>
<td>LNLvC</td>
<td><strong>Lowest Network load vCluster policy</strong>&lt;br&gt;VMs in the vCluster are first grouped together according to the VM exclusion vector, the groups are placed in different datacenters in clusters where the network load is the lowest. The LNLvC placement policy uses the VM cluster priority vector to determine which type of cluster to try first.</td>
</tr>
<tr>
<td>A</td>
<td>LSLvC</td>
<td><strong>Lowest Storage load vCluster policy</strong>&lt;br&gt;VMs in the vCluster are first grouped together according to the VM exclusion vector, the groups are placed in different datacenters in clusters where the storage load is the lowest. The LSLvC placement policy uses the VM cluster priority vector to determine which type of cluster to try first.</td>
</tr>
<tr>
<td>N</td>
<td>BBvC</td>
<td><strong>Bitbrains vCluster policy</strong>&lt;br&gt;VMs in the vCluster are first grouped together according to the VM exclusion vector, the groups are placed in different datacenters. The decision which clusters to use is made based on the combined CPU, storage, and network workload, and the requested memory size of the individual VMs.</td>
</tr>
</tbody>
</table>

Table 4.2: In this table we provide descriptions of the policies for vCluster placement. (Novelty is described with three letters; P: implementation of previous work, N: Novel work, A: adaptation of previous work.) The column Acro contains the acronyms of the policies.
Step 2 calculate individual resource scores:
During the simulation run we record usage statistics of the following resources: Memory, CPU, Storage, Network, Number of running VMs, Migrations. These statistics are recored per cluster and with a time interval set globally; in our experiments for example we monitor each cluster every 300 simulation seconds. This corresponds with the time interval used by the VMware tooling used in the Bitbrains production environment.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>GB</td>
<td>For each cluster we record the total amount of memory provisioned to VMs over time.</td>
</tr>
<tr>
<td>CPU</td>
<td>MHz</td>
<td>For each cluster we record the total amount of CPU used by VMs.</td>
</tr>
<tr>
<td>Network</td>
<td>MB/s</td>
<td>For each cluster we record the total amount of network traffic used by VMs.</td>
</tr>
<tr>
<td>Storage</td>
<td>ms</td>
<td>For each cluster we record the total amount of storage operations (IOps) used by VMs. The storage score is measured in latency so we translate the IOps into latency by using a storage latency model, this model is described in Section 4.4.3.</td>
</tr>
<tr>
<td>Migrations</td>
<td>#</td>
<td>For each policy we count the number of migration needed to provide the VM cluster mapping.</td>
</tr>
</tbody>
</table>

Table 4.3: In this table we provide descriptions of the monitored resources during simulation.

The memory scores and CPU scores are calculated in such a way that it favors filling clusters first before moving to the next cluster. Equations 4.1 to 4.4 describe how a resource score is calculated, and Table 4.4 shows descriptions of the used metrics and values. The score for storage is based on the latency model described in Section 4.4.3. The reason for focusing on latency is that high latency affects the overall performance of many applications severally. The migration score, is part of the total score to prevent excessive migrations. Migrations, in general, are a costly operation that should be kept to a minimum. Similar to the latency score, the migration score impacts the total score negatively. For the total policy score we use the total sum of all the individual scores divided by the number of total scores in our case 5 (memory, CPU, storage, network, migrations). In our design all the scores are considered equal, so they all have a weight of 1. In future work we will investigate how score weights can be used to optimize the scheduler.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Short</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Cluster utilization</td>
<td>Resource utilization over time, total resources used over a period of time divided by the total available resources.</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Total cluster usage</td>
<td>The total sum of resources usage over a period of time.</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Total available resources</td>
<td>The total sum of resources available over a period of time.</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Mean cluster utilization</td>
<td>The average utilization over all the clusters.</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Cluster resource score</td>
<td>The cluster resource score is a quadratic score.</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Resource score</td>
<td>The resource score is the mean of all the cluster resource scores.</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Number of clusters</td>
<td>The total number of clusters.</td>
</tr>
</tbody>
</table>

Table 4.4: In this table we provide descriptions of the variables used to calculate resource scores.

\[
\alpha = \beta / \mu \quad (4.1)
\]
\[
\delta = \Sigma \alpha / \eta \quad (4.2)
\]
\[
\gamma = (\alpha - \delta)^2 \quad (4.3)
\]
\[
\lambda = \Sigma \gamma / \eta \quad (4.4)
\]
Step 3 policy selection: Depending on the configuration of the scheduler (configuration are discussed in Section 5.2) we use the policy with the lowest or highest score. Choosing the policy with the lowest score forces better workload distribution and choosing the policy with the highest score will result in higher cluster loads. The latter is interesting, because this means that large chunks of free resource space remain for new vClusters. Equal workload distribution can make it difficult in the future to deploy large vClusters, because no cluster can accommodate the new vCluster, even when in terms of total resource space over all the clusters, there is actually enough space left.

4.4.3. Storage Latency Model
The latency in the storage stack heavily depends on both the physical setup (hardware) and the amount of storage operations (IOPS). In the model we present in this work the storage servers are physically separated from the compute servers and they are connected through a high performance low latency network (this operational model is known as Storage Area Network, SAN). In our BBsimulator, the VMs generate IOPS which we monitor per SAN. To calculate the storage score per cluster afterwards we use a model that transforms IOPS into latency. The latency model is based on behavior we have monitored in the production system. To analyze the storage layer, we recorded per SAN the number of read and write operations (IOPS) and the latency. Because all the storage servers have the same specs we merge all this data together and generate a probability model that predicts the latency for a SAN given a certain number of IOPS.

For the data collection we use VMware tooling that is in place in the Bitbrains production infrastructure. The results presented in here are specific to the hardware and software used in the Bitbrains infrastructure. Figures 4.4 and 4.5 show the latency models for read IO and write IO. On the vertical axis the figures show the frequency with which the latency occurs for the workload zones specified in the labels of the figures. On the horizontal axis the figures depict the latency in milliseconds (ms). For the latency model we characterize three IO zones (low, medium, and high), for each we specify the limits of the number of IOPS (respectively 0-99, 100-999, and >999). In the simulator the latency score is calculated by summing up all IOPS of all the VMs running on the same SAN for each time stamp. The number of IOPS determine which latency model (low, medium, or high) to use, the BBsimulator then uses a probabilistic mechanism to determine the latency per SAN per timestamp.
Virtualized datacenters provide important infrastructure for digital economies, but raise new challenges in resource management and scheduling. We propose Mnemos, a resource management and scheduling architecture for virtualized datacenters hosting business-critical workloads.

We present both the overall architecture of a portfolio scheduler as well as an in-depth design of specific policies and selection mechanism for scheduling business-critical workloads.

The architecture presented in this chapter forms the basis for the prototype implemented in the following chapter, where we will use the prototype to evaluate the use of portfolio scheduling for scheduling business-critical workloads in a multi-cluster, multi-datacenter setting.
EVALUATION OF A PORTFOLIO SCHEDULER FOR BUSINESS-CRITICAL WORKLOADS

In this chapter, we describe how we evaluate the portfolio scheduler for the Bitbrains case. For the evaluation of the portfolio scheduler we use a simulation based on recorded real-world workload traces. By comparing the results of the simulator with the workloads we have recorded in the datacenter, we get an indication of the quality of the scheduler.

This chapter answers the third research question.

How to evaluate the adapted portfolio scheduler, experimentally, through the implementation of a prototype?

We refine this research question into 3 sub-questions:

1. How to setup a realistic experiment to portfolio scheduling for business-critical workloads in a multi-cluster setting.

2. What do the results of the experimental phase tell us about the deployment in the real world?

3. What is the performance of the prototype and how does the model scale in terms of more VMs and more infrastructure?

5.1. OVERVIEW

For the evaluation we first describe the high-level architecture of the Bitbrains infrastructure and the metrics we use to evaluate the scheduler. We then describe the recorded workload traces and the evaluation process we use to validate the scheduler. We finish the chapter with a short evaluation of the performance, and the scalability of the scheduler.

The main goal of this chapter is to provide insights into the potential of portfolio scheduling for business-critical workloads. Ultimately the goal is to build a production ready scheduling system for which the designed prototype would be the blue print.

5.2. MULTI-CLUSTER, MULTI-DATACENTER EXPERIMENTAL SETUP

For the evaluation of the portfolio scheduler for business-critical workloads in a multi-cluster, multi-datacenter setting, we use the infrastructure of Bitbrains. Figure 5.1 depicts the high-level overview of the infrastructure. It shows 3 datacenters and 3 clusters. The datacenters are connected through a fiber optic ring network. In the Bitbrains infrastructure sets of 3 clusters are referred to as pods. Within pods compute servers are all of the same type, and they are connected through an low latency InfiniBand network. In the Bitbrains infrastructure we find 2 types of pods, standard pods and bigmem pods. The compute servers in the bigmem pods have a higher memory to CPU ratio than the compute servers in the standard pods. Each cluster in the Bitbrains infrastructure houses 6 Storage Area Network (SAN) devices.
Each of the SAN devices has the capacity of 10TB of storage, per cluster this adds up to 60TB. All the compute servers contain only CPU, memory, and networking, they do not have local storage. The difference between bigmem and standard compute servers is in the amount of memory that is available per CPU core. The standard servers have two 8-core CPUs and 128GB of memory. The bigmem servers have four 8-core CPUs and 768GB of memory. Clusters in a standard pod contain 16 standard servers and clusters in a bigmem pod contain 6 bigmem servers.

For the evaluation of the portfolio scheduler we run simulations with the scheduler setup with a different configurations. In Table 5.1 we describe the configurations. By running the scheduler with different configurations we obtain insights in its behavior, these insights help us to further develop the prototype into a production system.

Scheduling of workloads is always a tradeoff between performance, efficiency, cost, short term optimization, long term optimization, etc.. By no means do we provide an exhaustive evaluation of all the tradeoffs as this would be a project on its own. But we do investigate the tradeoffs most important to scheduling business-critical workloads. We investigate primarily the tradeoffs between performance and load balancing.

For the evaluation of the portfolio scheduler we use a workload trace recorded in the Bitbrains infrastructure. The trace is similar to the workload trace analyzed in Chapter 3, only this trace contains 1350 VMs. The trace spans a 2 month period from 23 August 2013 until 22 October 2013. This period is chosen because the infrastructure was stable during this period, no hardware was added or removed. Dynamically responding to changes in the infrastructure is left for future work.
**Table 5.1:** In this table we provide descriptions of the configurations investigated in this work.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replay</td>
<td>With this configuration we replay the original mapping as it was recorded in the datacenter. We present this configuration so that we can compare how the other configurations hold up against the VM provisioning as achieved by human operators.</td>
</tr>
<tr>
<td>MinScore</td>
<td>With this configuration we minimize the policy score (the policies are described in Section 4.4.1). This means that each run the policy with the lowest score is chosen as active policy. The result of this configuration is that we distribute the load as evenly as possible.</td>
</tr>
<tr>
<td>MaxScore</td>
<td>With this configuration we maximize the policy score. This means that each run the policy with the highest score is chosen as active policy. The result of this configuration is that we maximize the free space left in clusters so that we have large free chunks for provisioning future vClusters.</td>
</tr>
<tr>
<td>MinMem</td>
<td>With this configuration we isolate the Memory resource distribution score, in this case we minimize the score which results in equal distribution of memory utilization.</td>
</tr>
<tr>
<td>MaxMem</td>
<td>With this configuration we isolate the Memory resource distribution score, in this case we maximize the score which results in unequal distribution of memory utilization so that we leave large free chunks.</td>
</tr>
<tr>
<td>MinCPU</td>
<td>With this configuration we isolate the CPU resource distribution score, in this case we minimize the score which results in equal distribution of CPU utilization.</td>
</tr>
<tr>
<td>MaxCPU</td>
<td>With this configuration we isolate the CPU resource distribution score, in this case we maximize the score which results in unequal distribution of CPU utilization so that we leave large free chunks.</td>
</tr>
</tbody>
</table>
5.2.1. PROCESS FOR EVALUATING THE PORTFOLIO SCHEDULER

In this section we describe how we setup the process for evaluating the scheduler. Figure 5.2 depicts the process diagram of how we setup the evaluation process of the scheduler. In the workload trace we use, we find a combination of individual VMs and vClusters (sets of VMs). We order these items in a list by first start time. Next we setup the scheduler according to one of the configuration described in Section 5.2. We start with an empty system and add all the VMs and vClusters step by step. This means that for every entry in the sorted list we run the portfolio scheduler, including the BBsimulator. In every step we replay the results we found in the previous step and add the new VM or vCluster from the sorted list. If no more VMs or vClusters require scheduling we store the final mapping and store all the log and performance data for evaluation.

5.3. METRICS FOR EVALUATING THE PORTFOLIO SCHEDULER

For the evaluation of the portfolio scheduler, we use both commonly used metrics (e.g., resource utilization and runtime), and some specially designed metrics (e.g., risk-score and policy distribution). Next we describe the metrics we use for the evaluation of the portfolio scheduler.

We evaluate the scheduler along two axes, firstly we look at the workload distribution as achieved by the portfolio scheduler. Secondly we look at the scalability of the portfolio scheduler. Because we are working with rapidly growing workloads it is important to know how the scheduler behaves under growth conditions.

We use the commonly used box-plot\(^1\) model to summarize workload patterns that we record per cluster. We first present an artificial example of how we summarize workload patterns in box-plots. In Figure 5.3a we present the utilization of a resource of a cluster, for instance CPU utilization in cluster C1. Figure 5.3b shows the corresponding box-plot, the figure shows that more than 95% of the data falls between 70% and 98% utilization. There is one data point where the utilization was 5%. In general, in a box-plot 50% of the data is situated within the limits of the box and 95%-100% of the data is situated between the whiskers. The box is split in two by the median value. A maximum of 5% of the data can be described as outliers if the data-points do not fall within the region between the whiskers. In the box-plot we use sum signs to present the outliers.

---

\(^1\)For details, we refer to the free Statistics Textbook: [http://www.statsoft.com/Textbook](http://www.statsoft.com/Textbook)
Because of the density of information in the box-plot figures we present later, we also present the mean values of the workload patterns per cluster per configuration. In Figure 5.5 we present for instance the mean CPU load per cluster per configuration. We present box-plots and mean value distributions for all four resources types that we investigate in the experimental evaluation (CPU, Memory, Storage, and Network).

We also introduce a new metric which we call the risk-score. We use the risk-score to identify which configuration scores best on mitigating oversubscription risk. This is one of the risks we are trying to manage by using portfolio scheduling. Oversubscription risk, is the risk of running into resource scarcity. This can for instance happen if more resources are allocated to VMs than there are available in a cluster. High latency can severely impact the performance of applications, therefore we indicate high latency as a risk. High latency can for instance be caused by running too many VMs on the same SAN at the same time. In Table 5.2 we define the risk scores for CPU, Memory, Storage, and Network in more detail.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Risk-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>For the risk score of CPU load we combine two metrics into one value, we start by defining a cluster threshold. In our case 3 redundant hosts in a standard cluster and 1 redundant host in a big-mem cluster. We use the mean CPU load as a metric for the severity of the risk and the number of timestamps above the threshold as a metric for the peak risk.</td>
</tr>
<tr>
<td>Memory</td>
<td>For the risk score of memory we combine two metrics into one value, we start by defining a cluster threshold. In our case 3 redundant hosts in a standard cluster and 1 redundant host in a big-mem cluster. We use the mean memory load as a metric for the severity of the risk and the number of timestamps above the threshold as a metric for the peak risk.</td>
</tr>
<tr>
<td>Storage IO</td>
<td>For the storage IO risk we use latency as a metric for the risk, we combine the mean latency and a peak counter (timestamps that the latency is higher than 5ms).</td>
</tr>
<tr>
<td>Network</td>
<td>For the network resources we use the overall mean as a threshold, we again use both the mean workload per cluster and the total peaks above the threshold. Using the mean as a threshold encourages equal distribution of workload.</td>
</tr>
<tr>
<td>Datacenter memory balance</td>
<td>This risk metric indicates the risk of unbalancing workloads over datacenter specific to memory. If the memory distribution over datacenter becomes to unbalanced it increases the risk during a disaster (complete datacenter outage).</td>
</tr>
</tbody>
</table>

Table 5.2: Method for calculating the risk-score for the different configurations of the portfolio scheduler.

For the evaluation we further record metrics internal to the portfolio scheduler such as how often policies
5.4. RESULTS OF THE SIMULATION BASED EVALUATION

In this section, we present the results of the evaluation of the portfolio scheduler. We first report how the portfolio scheduler distributes workloads over clusters for the different configurations we introduce in Section 5.2. Secondly we report the results of metrics internal to the portfolio scheduler. Finally we report results of the evaluation of scalability and performance.

5.4.1. DISTRIBUTION OF WORKLOAD FOR DIFFERENT CONFIGURATIONS

For the comparison of the configurations we look at all four resources that we cover in this work (CPU, memory, storage and network). By comparing all resource types we can show the tradeoffs we have to make between performance and load balancing of workloads over clusters. In this section we only include the most relevant figures, the remaining figures can be found in Appendix L until O.

We first analyze the distribution of memory and CPU workloads as these are the primary resources we want to manage. Figure 5.4 depicts the distribution of mean memory utilization per cluster, per configuration. On the horizontal axis we show the clusters, on the vertical axis we show the utilization in percentages (0%-100%). Per cluster we show the 7 configurations. With the horizontal line at about 48% utilization we show the overall memory utilization. This means that if all load was evenly distributed over all clusters they all would have a mean utilization of 48%. Figure 5.5 depicts the mean utilization of CPU workloads. It directly shows that CPU usage is much lower than memory usage (overall mean CPU usage is about 12%).

In Figure 5.6 we show box-plots for memory utilization per cluster, per configuration. Similarly Figure 5.7 shows box-plots for CPU usage. In Figure 5.7 and 5.6 we show on the vertical axis the clusters. On the horizontal axis we show the utilization in percentages (0%-100%). Per cluster we show 7 box-plots, one for each configuration. Many of the clusters show box-plots that cover a large part of the utilization space, this means that for these clusters the variability in workloads is very high. Some of the box-plots however show very small variability for instance the box-plot for the replay configuration in cluster C90 for memory.
### Configurations:

<table>
<thead>
<tr>
<th></th>
<th>replay</th>
<th>MinScore</th>
<th>MaxScore</th>
<th>MinMem</th>
<th>MaxMem</th>
<th>MinCPU</th>
<th>MaxCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>1.68</td>
<td>0.31</td>
<td>2.18</td>
<td>6.16</td>
<td>1.56</td>
<td>0.26</td>
<td>15.09</td>
</tr>
<tr>
<td>CPU</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>IO write</td>
<td>2.6</td>
<td>1.53</td>
<td>0.0</td>
<td>0.0</td>
<td>1.49</td>
<td>1.69</td>
<td>1.35</td>
</tr>
<tr>
<td>IO read</td>
<td>2.31</td>
<td>1.25</td>
<td>2.99</td>
<td>1.2</td>
<td>1.23</td>
<td>2.05</td>
<td>1.41</td>
</tr>
<tr>
<td>Network send</td>
<td>1.86</td>
<td>1.18</td>
<td>2.51</td>
<td>1.13</td>
<td>1.32</td>
<td>0.9</td>
<td>1.86</td>
</tr>
<tr>
<td>Network receive</td>
<td>1.17</td>
<td>1.13</td>
<td>2.19</td>
<td>1.09</td>
<td>1.1</td>
<td>1.11</td>
<td>1.49</td>
</tr>
<tr>
<td><strong>Sub Total:</strong></td>
<td><strong>9.62</strong></td>
<td><strong>5.4</strong></td>
<td><strong>12.01</strong></td>
<td><strong>11.07</strong></td>
<td><strong>6.9</strong></td>
<td><strong>5.66</strong></td>
<td><strong>21.78</strong></td>
</tr>
<tr>
<td>Datacenter memory balance</td>
<td>0.85</td>
<td>2.69</td>
<td>1.32</td>
<td>1.78</td>
<td>1.62</td>
<td>2.1</td>
<td>2.59</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>10.47</strong></td>
<td><strong>8.09</strong></td>
<td><strong>13.33</strong></td>
<td><strong>12.85</strong></td>
<td><strong>8.52</strong></td>
<td><strong>7.76</strong></td>
<td><strong>24.37</strong></td>
</tr>
</tbody>
</table>

Table 5.3: Risk score for the different configurations of the portfolio scheduler.

(Figure 5.6) (the top most box-plot). This means that at least 95% of the memory utilization in this case is centered around 45% utilization, there are a few outliers at around 5% utilization. Translating this back to workload patterns, this means that for cluster C90, for the replay configuration the memory workload was very stable centered around 45% utilization.

High variability in workloads (in the box-plot figures represented by long stretched box-plots) can for instance be explained by services that are heavily used during the day time and not used during the night time. An other reason for high variability can be compute nodes that run for short periods of time.

To interpret all the data presented in the box-plot figures and the mean utilization figures we use the risk score explained in Section 5.3. As explained earlier a lower score is better for the Bitbrains case because it means lower risk of running into resource scarcity, which in turn would result in loss of performance. In Table 5.3 we present the results of the risk score analysis. We present both the individual resource risk scores as well as the overall risk score per configuration.

The MinScore configuration scores lowest (sub total risk score: 5.4). This means that with the policies we implemented the MinScore configuration is the best choice when it comes to reducing the oversubscription risk. The MinCPU configuration however scores similarly low (sub total risk score 5.66). In Table 5.3 we also report the datacenter memory balance score, it shows that all configurations score worse than the replay configuration. This is caused by workload imbalance between datacenters. We plan to investigate datacenter workload balancing in future work.

#### 5.4.2. Distribution of Selected Policies for Different Configurations

The distribution of policies chosen during the simulation runs provides insights to the need of having a portfolio scheduler. In Figure 5.8 we depict the distribution of the chosen policies for the 6 configurations we investigate. Figure 5.8a shows the distribution of the policies used for provisioning vClusters (sets of VMs), Figure 5.8b show the distribution of policies used to provision individual VMs.

When we compare the distribution of policies for the MinScore and MaxScore configurations we observe only small differences. More interestingly is that both strategies result in a relatively uniform distribution of policies that are used during scheduling for both vClusters and individual VMs. The configurations that focus on optimizing only one resource present much more specific choices of policies. The MaxCPU configuration for instance, selects the Lowest Storage load policy for individual VM placement almost 90% of the time, and for vClusters it selects mainly the Lowest Storage load policy or the First Fit policy.

Both MinScore and MaxScore show promising results when it comes to reducing the risk-score, it is therefore very interesting to observe that these configurations result in an evenly balanced choice of scheduling policies. This strengthens the idea that different policies working together in a portfolio leads to better scheduling decisions.

The very poorly scoring MaxCPU configuration was the result of very unbalanced choice of policies, for the vCluster policies about 80% of the choices were made in favor of two policies (LoestStorageLoad and FirstFit), and for the single VM policies more than 80% of the choices was made in favor of a single policy (LowestStorageLoad). This again strengthens the idea that a rich set of policies with a portfolio scheduler configured to use a mixture of the policies will result in better overall results.
5.4.3. Performance and Scalability Evaluation of the Portfolio Scheduler

In the portfolio scheduling there are two modes of operation, online scheduling and periodic scheduling. As explained in Section 2.3.3, the online scheduling mode the decision on which policy to use is made for every scheduling request, in our case the placement of a VM or vCluster. In the periodic scheduling mode the decision of which policy to use is made for upcoming period (time slot), often spanning multiple requests. The preferred mode is online scheduling, however this is only possible if the portfolio scheduler can make scheduling decisions faster than new request are coming in. If the incoming request rate is higher than the scheduling rate work piles up, which is undesirable because it introduces extra wait time and thus loss of performance.

The runtime performance of the scheduler is important because short runtimes allow us to use online scheduling. For the this work we implemented a parallel processing model so that we can calculate the score of all the policies at the same time per request. Because of this parallel model the runtime of the scheduler is limited by the runtime of the slowest policy. To get insight in the runtime of the scheduler we recorded the runtime of the scheduler during the evaluation of the different scheduling configurations. In Figure 5.9 we show the runtime of the scheduler against the number of running VMs during that round of scheduling. We find that the runtime scales linearly with the number of VMs. This means that there is a limit to the number of VMs we can schedule with the online model. The exact limit depends on the intake rate of new scheduling requests.
Figure 5.6: Distribution of memory workload.
Figure 5.7: Distribution of CPU workload.
5. Evaluation of a Portfolio Scheduler for Business-Critical Workloads

(a) Policy distribution for vCluster placement.

(b) Policy distribution for VM placement.

Figure 5.8: Evaluation of policy distribution for minimizing and maximizing the placement scores. On the vertical axis we show the fraction of times the policy was chosen.

Figure 5.9: Runtime of the portfolio scheduler.
5.5. **Summary**

In this chapter we present the results of the evaluation of the implemented prototype. The evaluation shows good results that strengthen the choice for using portfolio scheduling for provisioning business-critical workloads in a multi-cluster, multi-datacenter setting. It is also interesting to observe that the scheduler results in lower risk scores than human operators did in the past. This is inline with the idea that large scale complex systems are very difficult to comprehend for human operators. The MinScore configuration shows very promising results when it comes to reducing the risk score and balancing workloads.

The implemented prototype shows that it is possible to implement a linearly scalable model. This means that for the scale of the Bitbrains infrastructure we can implement an online version of the portfolio scheduler. For much larger infrastructure or for infrastructures with a much higher VM ingest rate the model needs to be redesigned to allow for online scheduling. The implementation of the scheduler however leaves room for code optimization that can reduce the runtime of the scheduler. These optimizations are not pursued in this work because the purpose of this work is to show that portfolio scheduling can work for provisioning VMs in a multi-cluster multi-datacenter setting. With the runtimes we observe and the potential for further optimizations, we are convinced that an online portfolio scheduler can work in the Bitbrains infrastructure.
Virtualized datacenters provide important infrastructure for digital economies, but raise new challenges in resource management and scheduling. We present portfolio scheduling as an approach for scheduling business-critical workloads in virtualized datacenters. In the problem statement we present three main questions that we investigate and answer throughout this work:

1. **What are the characteristics of business-critical workloads?**
   We investigate in this work the characteristics of business-critical workloads and explain there differences with other known workloads. We use the lessons learned to define a risk score that we use to evaluate the portfolio scheduler that we design. The characterization study presented in this work has resulted in an accepted article in IEEE/ACM CCGRID 2015: **V.S. Van Beek et al., Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters, IEEE/ACM CCGRID 2015, Shenzhen, Guangdong, China, May 4-7, 2015 (acceptance ratio 25%).**

2. **How to adapt the concept of portfolio scheduling to the placement of VMs in a multi-cluster multi-datacenter setting?**
   We present a design for the adaptation of the concept of portfolio scheduling, that can deal with business-critical workloads in a multi-cluster, multi-datacenter setting. In the design of the scheduler, we propose the BBsimulator. The BBsimulator is used to evaluate the scheduling policies that are at the core of the portfolio scheduler. The design of the scheduler and the BBsimulator forms the basis for an article currently under submission: **V.S. Van Beek et al., Mnemos: Self-Expressive Management of Business-Critical Workloads in Virtualized Datacenters, IEEE Computer Special edition (Self-Aware and Self-Expressive Computing Systems), submitted on 15 December 2014 (expected acceptance ratio 10-30%).**

3. **How to evaluate the adapted portfolio scheduler, experimentally, through the implementation of a prototype?**
   To evaluate the adapted portfolio scheduler we implement a prototype of both the scheduler and the BBsimulator. We use real-world workload traces to experimental evaluate the workings of the scheduler and the BBsimulator.

At the start of this project, scheduling decisions were made by human operators. This is a difficult and time consuming task that we want to automate. In this work we show that portfolio scheduling is a viable alternative to human operators. In the evaluation study we show that the portfolio scheduler can outperform human operators in terms of optimizing the workload distribution. We also show that the implemented prototype can deal with the scale (number of VMs and clusters) in terms of performance required by modern datacenters. Our discussions regarding applying these techniques at Bitbrains, and more broadly in all virtualized datacenters powered globally by the market-leader VMware, are very promising.

**Future work**

To continue the work on portfolio scheduling and workload analysis we applied and received two research grants. For continuing the work on portfolio scheduling, we received a grant from COMMIT, which will mainly be used to valorize the work into a production system. The other grant, for a NWO/STW KIEM project will
be used to continue the work on workload analysis and will help us to create benchmarks based on these workload characterization studies.

We further identify five concrete possibilities for future improvements:

1. **Rebalancing workloads**
   In this work we focus on provisioning new VMs. One of the challenges that we did not pursue is the challenge of rebalancing workloads after they have been provisioned the first time. Incorporating this in our design would require a mechanism that can determine if there are imbalances. Second if an imbalance is detected it would require VM migration to rebalance the workloads. Because this is a process that takes time and it is resource intensive and can lead to performance penalties for the VMs, it would require the development of a scoring mechanism. This scoring mechanism should make the decision if it is worth while to migrate the VMs. Our scheduler could than be use to find a new cluster for the VMs.

   A second reason for rebalancing workloads would be the introduction or removal of resources. For instance, if a cluster is removed, the workloads that were running in that cluster need to be migrated to other clusters. Or if a new cluster is introduced, this new cluster could alleviate other clusters by taking over some of their workloads.

2. **Optimizations of the portfolio scheduler**
   The newly proposed portfolio scheduler can be optimized along two axis: runtime and memory scalability, and quality improvements of the VM placement. For the later we will investigate how score weights can be used to optimize the scheduler. In Section 4.4.2 we describe how we value all resources equally, it would be interesting to experiment with other resource weights. In terms of performance optimizations there are many possibilities. The implemented prototype can currently run all the policies in parallel, but there are other code optimizations that can reduce the runtime of the scheduler and the BBsimulator. Pursuing these is important to keep-up with the growth of the infrastructure and customer base faced by Bitbrains.

3. **Scheduling policies**
   New types of hardware and workloads will require the implementation of new scheduling policies and perhaps also evaluation functions. A concrete new requirement that has recently come to light is the introduction of local hard-disks in the compute nodes. These disks are solely used for Big-Data applications so new policies are required to deal with this new architecture.

4. **Load balancing over datacenters**
   As shown in the evaluation of the portfolio scheduler, the current setup can result in load imbalances between datacenters. Further research is needed to investigate how the portfolio scheduler can be configured to take datacenter load balancing into account.

5. **Prototype to production**
   To turn the prototype, we developed for this project, into a production system the following steps need to be taken: integration with the VM manager, implementation of an API that unlocks the functionality of the portfolio scheduler, and finally we need to validate the results of the portfolio scheduler in a production system.

   For the integration with the VM manager deployed by Bitbrains we need to implement integration with the VMware vCenter API. For the validation in a production system we envision the following process: an engineer deploys a vCluster, the portfolio scheduler responds with a VM-cluster mapping, the engineer checks the mapping, and after validation by the engineer the vCluster is automatically deployed by the VM manager.


Appendices
LITERATURE STUDY ON SCHEDULING
A.1. SCHEDULING

A.1.1. INTRODUCTION

In the context of this survey, “Scheduling” is the process of provisioning resources and assigning jobs to these resources. Over the past decade much research has been done on scheduling. Most of this research focuses on finding optimal scheduling mechanisms for specific workload types. The workloads are categorized as:

1. Sequential jobs / bags-of-tasks (BoTs) (e.g., parameter sweep applications (PSAs), high-throughput computing)

2. Workflows (e.g., Scientific workflow, MarReduce)

3. Parallel jobs (e.g., High performance computing (HPC))

4. Miscellaneous (e.g., multi-tier application hosting)

Because of the complexity of scheduling problems regarding resource scheduling many authors have investigated specific practical sub problems. This often leads to good solutions for very specific cases. Scheduling of resources is often described as the theoretical bin-packing problem \[38\]. Because resources in computer systems have multiple dimensions (e.g., memory, cpu, storage, network, etc.) that often need to be considered in relationship with each other, the normal one-dimensional bin-packing problem is to limited and the multidimensional bin-packing (vector bin-packing) problem is used instead. Because the vector bin-packing problem proved to be APX-hard \[39\] there is no PTAS algorithm that can be used to approximate the problem in polynomial time. Therefore research has focused on finding good heuristics to solve practical problems \[40, 41\] which is in line with advances made in scheduling resources in datacenters during the last decades.

The remainder of this section is organized as follows:

- First (section A.1.2) the taxonomy is discussed in detail.

- Followed by (section A.1.3) a description of the most important terms and definitions.

- Section A.1.4 contains the most recent developments in each of the topics defined in the taxonomy.

- Section A.1.5 concludes the section on scheduling by identifying future research directions.
A.1.2. **TAXONOMY**

The scheduling taxonomy depicted in Figure A.1 shows four main categories in which advances in scheduling techniques regarding resource scheduling in datacenters can be categorized. The four categories listed below are described in more depth in sections A.1.2 until A.1.2.

1. specialized scheduling
2. meta scheduling
3. scheduling single-tier applications
4. scheduling multi-tier applications

**SPECIALIZED SCHEDULING**

Scheduling resources in datacenters can be done on different levels in the execution stack. Usually the higher in the stack the decisions are made the more information is available of the actual context of the workload and the workload itself. Specialized scheduling focuses on finding scheduling policies for very specific types of workloads (workflows, parallel jobs, and sequential jobs). The idea is that the more information there is available the better the scheduling can be performed. So specialized scheduling mechanisms can often be found high in the execution stack (software level). This does however not mean that these policies cannot make decisions about lower levels. For instance if information is available of what is exactly running on a VM this can help to better provision VMs to hardware. Recent developments in the three categories (sequential jobs, workflow, and parallel jobs) of specialized schedulers can respectively be found in section A.1.4 until A.1.4.
Meta Scheduling
Because many datacenters run a wide variety of different workloads the chance of finding a policy that works for all of them is very small. Therefore work has focused on finding ways to deal with this problem. One of the directions of research is portfolio scheduling. The portfolio scheduling model works with a set of policies, from this set of policies the best policy is chosen for every subset of the total workload. In this way less complex policies can be created that work well for subsets of the total workload without having to question whether the policy also works well for all other workloads. Recent developments on meta scheduling can be found in section A.1.4.

Scheduling Single-tier Applications
Work on model based scheduling focuses on finding workload models and using the insights that these models provide to optimize scheduling future workloads. These workload models are also often used to test scheduling policies in simulators. Different modeling approaches (e.g., analytical modeling, and statistical modeling) are used to develop workload models. Recent developments on model based scheduling can be found in section A.1.4.

Scheduling Multi-tier Applications
Multi-tier applications often have two characteristics (multicollinearity effect, and very dynamic load patterns) that are very specific for this type of setups. Because web-applications are very often hosted on multi-tier systems (often two-tier: web server and database server) lots of research has been done on this topic. This results in a variety of approaches that use different modeling strategies (queue models, prediction based models, and analytical models) to achieve effective dynamic resource allocation for multi-tier applications. Recent developments in the three types of modeling (queue models, prediction based models, and analytical models) approaches can be respectively be found in section A.1.4.

A.1.3. Terms and Definitions
In this section the most important terms and definitions are described that are later used to describe and compare different scheduling approaches.

The makespan of a job is the time it takes for a job to finish from the moment it is submitted to the system.

One of the concepts that appears much in the research on scheduling is fairness, many authors consider fairness in some way to be important in the context of scheduling [42–48]. What is considered to be fair in the context of scheduling is open to debate and often depends on the practical situation. Often fairness is expressed in terms of how policies affect the makespan of jobs from different users.

Some authors consider the scheduling of deadline constrained jobs, these jobs have the characteristic that they have to finish before a certain point in the future. Deadline constraints sometimes are used to define Quality of Service constraints (QoS), service providers use these to promise users that jobs will finish within a certain time margin after the short possible makespan.

QoS constraints can encompass more than only guaranteed deadlines. QoS constraints are closely related to agreements made between users and service providers. These agreements are usually defined in Service Level Agreements (SLAs).

Many applications are nowadays deployed as multi-tier systems [49]; a simple example of a two-tier system is a system with an application server, and a database server.

When analyzing the workload of multi-tier systems the problem of multicollinearity is always present. Multicollinearity is a statistical phenomenon in which two or more predictor variables in a multiple regression model are highly correlated, meaning that one can be linearly predicted from the others with a non-trivial degree of accuracy [50]. In this situation the coefficient estimates may change erratically in response to small changes in the model or the data. Multicollinearity does not reduce the predictive power or reliability of the model as a whole, at least within the sample data themselves; it only affects calculations regarding individual predictors. That is, a multiple regression model with correlated predictors can indicate how well the entire bundle of predictors predicts the outcome variable, but it may not give valid results about any individual predictor, or about which predictors are redundant with respect to others.

A problem that is closely related to multicollinearity is bottlenecks shifting. Bottlenecks shifting is the phenomenon that different tiers can become the bottleneck at different times, this often happens when one tier is scaled up or down or when the workload characteristics change.

A metric often used to define the performance of scheduling mechanisms from a hosting provider perspective is the utilization of a system, utilization is the percentage a resource is actively used over a period of time.
A method often used by datacenter operators to move workload from one physical host to another is *migration*, with migration VMs are moved from one host to another without interrupting the operating system and software running on that VM.

### A.1.4. Latest Developments

This section contains the latest developments in each of the topics defined in the taxonomy as described in section A.1.2. The publications described in this section are selected from the more prestigious conferences in the field (e.g., CCGrid, SuperComputing, JSSPP, HPDC, CLOUD, Europar).

#### Scheduling Sequential Jobs

Sequential jobs are often used for scientific and engineering workloads. The jobs in these workloads are often presented in the form of bags-of-tasks (BoTs). BoTs are sequential tasks that are submitted to a system as a group of tasks that belong to the same job. In grid computing they account for a large portion of all the jobs [51]. Much research has been done on scheduling BoTs; earlier comparisons between different scheduling policies were conducted by Braun et al [52] and Maheswaran et al [53].

Iosup et al. [51] investigate the performance of scheduling BoTs in large-scale distributed systems. They also propose a taxonomy that divides different scheduling policies in terms of the availability of information and the quality of information. They divide information classes into knowledge about the tasks in the BoTs and about the resources needed to execute these tasks. In a large-scale experiment with real-world traces of workloads, the authors show the performance of different scheduling policies under varying workloads. The main conclusion is that there is still much work to be done in finding a scheduler that effectively schedules BoTs. Main this are that existing policies vary heavily in terms of performance under changing workloads, and dynamic system information.

Netto et al [54] take a different approach compared to more traditional schedulers that work top down. Instead of having the scheduler mandating that a certain task needs to run on a certain resource, they advertise the jobs to the resources, then the resources respond by offering to execute a number of jobs within a certain time frame. This way the authors build a scheduler that can cope with deadline constraints, which is important because in this way it is possible to schedule jobs with regard to quality of service (QoS) constraints.

Laredo et al. present a BoT scheduling mechanism, that they describe a self-organized system that omits a central scheduler. Laredo et al call their system sandpile, the idea is that new jobs are submitted to resources and if a resource becomes overloaded the work is spread over neighbor resources. This mechanism of overflowing to neighbors can lead to avalanches but in the end the system will reach a new equilibrium. The proportion of the size of the avalanches and the frequency of the avalanches shows a power-law relation, this means that small avalanches happen often and large catastrophic avalanches happen very rarely. With synthetic workloads the authors show the viability of this approach. They have planed to test this approach with more realistic workloads in the future.

#### Scheduling Workflows

On of the most used workflow models is the MapReduce model whereas there are also others e.g., Triana [55], and Kepler [56]. The MapReduce model received massive interest from both the academic world and companies (e.g., Google and Facebook). Because of the popularity of MapReduce, the scheduling part of the model also received a great amount of attention.

One of the key components of a good MapReduce scheduler is that it must prevent unnecessary data transmission. This can be achieved by enhancing data locality (placement of tasks on the nodes that contain the input data). The issue of data locality has been previously addressed by Zahiria et al. [42]. The standard scheduler places jobs as they arrive, this can lead to bad data locality if no host is available that has the data locally for the arriving job. Zahiria et al. propose to hold a job in the case no host is available that has the data locally and try to reschedule the job later. The experimental results show that the delay-scheduling algorithm achieves near optimal data locality.

He et al propose a new algorithm called matchmaking that also optimizes data locality [57]. The main idea of the matchmaking approach is that local map tasks always trump non-local map tasks. The scheduler assigns local jobs to nodes, if a there is no local task for a node the node will not receive a task for that interval. If this happens two intervals in a row, a non-local task will be assigned to that node. In this way they achieve both high data locality and high cluster utilization. In the evaluation, the authors compare the matchmaking scheduler with the default FIFO scheduler and with the delayed scheduler. The evaluation shows that the matchmaking scheduler out performs standard First In First Out (FIFO) scheduling and delay scheduling, in
almost all cases. Another big advantage of the matchmaking approach over the delay-scheduler approach is that it does not require an intricate parameter-tuning process.

Often several deployments of MapReduce need to coexist on the same infrastructure. This can for instance mean that separate development environments, and production environments are needed. To facilitate this sort of multi deployment architectures while still deploying all these systems in one physical infrastructure a resource management system is needed that can deploy MapReduce clusters and schedule tasks on these deployed clusters.

Ghit et al. describe a MapReduce cluster deployer and a job scheduler on top of the Koala grid scheduler. They propose the following three scheduling policies: Grow-Shrink Policy (GSP), Greedy-Grow Policy (GGP), and Greedy-Grow-with-Data Policy (GGDP). In the system design the authors make a distinction between two types of Hadoop (an open-source implementation of the MapReduce model) nodes, core nodes and transient nodes. The first type is a normal Hadoop node that contains the data in the HDFS needed during execution. The second type is a node that does not contain data and is only used to do execution. To test the performance of the scheduling policies the authors use the real DAS-4 multi-cluster grid. As benchmark workloads the authors choose to use wordcount and sort, two well known tasks that are often used in MapReduce. The experimental results show that cpu bound application for example wordcount scale well on transient nodes while IO bound applications for example sorting suffer a high performance penalty when the number of transient nodes increases.

In current datacenters MapReduce is often deployed in virtualized environments; this is done to facilitate resource sharing, scalability, isolation, and cluster utilization optimization. All virtualization environments share the problem of performance degradation due to context switching. Especially MapReduce jobs that are data-intensive (which is the case for many MapReduce jobs), and thus have many I/O operations, suffer from performance degradation when deployed in a virtualized environment. Kang et al. modified the Xen scheduler to accommodate for a fair scheduling policy that uses group scheduling to improve performance and uses I/O batching to eliminate superfluous context switching. The interesting thing about group scheduling is that it uses MapReduce characteristics to improve scheduling at hypervisor level. Fairness is implemented at two levels, VM level and Group level, this is done because the default Xen credit scheduler results in unfair scheduling when applied on MapReduce workloads. Extensive experiments on both single-cluster and multi-cluster Hadoop deployments show that the improvements of the Xen scheduler result in more fairness and higher performance in terms of job finish time.

Scheduling Parallel Jobs

Feitelson et al. published a status report on parallel job scheduling in which they describe amongst other things two main approaches for scheduling parallel jobs. The two approaches are backfilling and gang scheduling. Backfilling is an optimization of the FIFO scheduler, the backfilling mechanism places jobs further down in the queue on resources if this does not delay the start of the job at the head of the queue. Gang scheduling uses preemption and re-scheduling to make sure that tasks belonging to a single jobs run in parallel. Feitelson et al. address a number of issues that arise with gang scheduling, among these issues are: short jobs remain in queue (fairness issue), fragmentation issues, and dealing with memory pressure issues. Papazachos et al. discuss the problem of fragmentation, they propose a migration technique to transfer tasks that are waiting in a queue to other resources so that the gang can start execution earlier. The system is only tested in simulation but shows promising results, more work has to be done to test the system in a real world scenario.

To use cloud infrastructures for HPC a scheduler is needed that can effectively schedule (in terms of both cost and performance) parallel jobs in the cloud. Moschakis and Karatza show how gang scheduling can be effectively used for scheduling HPC jobs on the Amazon EC2 cloud infrastructures. Experimental results show that both Adaptive First Come First Serve (AFBFS) and Largest Job First Served (LJFS) gang scheduling policies can be used and perform well in terms of cost. LJFS scores better in terms of performance, especially when the jobs become heavier.

Scheduling Multi-tier Applications

Multi-tier systems are used for a variety of applications, one of the most used multi-tier systems is the web stack (web server + database server). Some times these web stacks are two-tier systems but there are also deployments with more tiers, for instance if dedicated storage tiers or caching tiers are added. To model the behavior of multi-tier applications a model that uses a network of queues can be used. Each tier in this case is modeled by a queue and the queues are connected so that requests and messages can be send from one tier
to the other. Urgaonkar et al. present a queuing model for multi-tier internet applications, their model can handle amongst other things session-based workloads, and multiple heterogeneous tiers with caching in all tiers \cite{62}. The authors validate the queuing model using benchmarks on a real Linux cluster running a three tier web service as depicted in Figure A.2.

![Figure A.2: Request processing in an online auction application \cite{62}.](image)

In later work Urgaonkar et al. used the queuing model to develop a dynamic provisioning system for multi-tier internet applications \cite{63}. The system uses a mix of reactive provisioning, and predictive provisioning policies to accommodate for changing workloads. The predictive provisioning mechanisms predict the arrival rate of request up to one hour in advance, the prediction is made on the basis of historical data of the previous days. If the difference between the actual arrival rate and the predicted arrival rate is to big a new provisioning decision is made, this can either be a scale up or a scale down decision. If the system encounters inherently unpredictable arrival patterns (e.g., flash crowds) it uses reactive policies to quickly scale up or down. In a large scale experiment with both a system similar to an e-bay \footnote{http://www.ebay.com} system, and a Slashdot \footnote{http://slashdot.org} system the authors show that the dynamic provisioning system can cope with both predictable variable workloads, and with unpredictable flash crowds.

In the search for resource demand modeling for multi-tier systems much time is spend on developing regression models that use Least Squares regression (LSQ) and Least Absolute Deviations regression (LAD). For single tier systems these methods show to be very effective but when applied to multi-tier systems the problem of multicollinearity leads to very wide confidence intervals, which means that the predictive value is very low.

To solve this problem Rolia et al. \cite{64} propose a new solution which they call Demand Estimation with Confidence (DEC). DEC uses a system of linear equations to model system functionalities, to predict resource demands for a new set of functionalities DEC uses a set of benchmarks to solve the system of linear equations. If not enough data is available DEC provides this information and request more information instead of making a poor prediction. Because DEC does not rely on a single regression model it is better capable of predicting resource demands for new workload mixes (e.g., a new set of system functionalities).

Rolia et al. use the industry standard TCP-W benchmark \footnote{http://www.tpc.org/tpcw/} for benchmarking e-commerce systems to compare DEC, LSQ, and LAD. DEC outperforms LSQ and LAD in almost all situations and DEC perform especially well when the multi-tier system suffers from multicollinearity. When DEC does not have enough data to make an accurate prediction it performs quit badly the authors however show that DEC can easily be adapted to use LP relaxation in these cases to improve the results. Using LP relaxation for these workloads

\begin{enumerate}
  \item Client $\rightarrow$ HTTP (place bid on some item)
  \item HTTP $\rightarrow$ J2EE (servlet invokes EJB)
  \item J2EE $\leftrightarrow$ Database (EJB issues queries, database responds)
  \item J2EE $\rightarrow$ HTTP (EJB constructs response)
  \item J2EE $\rightarrow$ HTTP (response sent to HTTP server)
  \item HTTP $\rightarrow$ Client (response sent to client)
\end{enumerate}
improved the maximum error percentage from 86% for DEC to 8% for relaxed DEC; regression models had a maximum error percentage of 20% for these workloads.

Most large-scale multi-tier systems (especially web applications) need to deal with bursty workloads. Because most regression models are solely based on mean value analysis (MVA) it is very difficult to make accurate predictions for resource demands for systems dealing with bursty workloads. Casale et al. [65] propose to use a new model that uses three parameters (mean, index of dispersion, and 95th percentile of service demands) to predict resource demands. The index of dispersion of the service times is used to model the burstiness of the workloads.

Another problem described by Casale et al. [65] is the problem known as bottleneck switching. In multitier systems it can happen that under different workloads different tiers become the bottleneck in delivering a response to the user. Similar to Rolia et al., Casale et al. use the TCP-W benchmark to test their model. They also use a queuing model to model the multi-tier nature of their test setup. The benchmark results show that the proposed model improves on the MVA results. Especially when the load increases or when bottleneck switching happens the predictive quality of MVA drops while the newly proposed model remains accurate. Another strong characteristic of the newly proposed technique is that it can use coarse measurements, which means that the overhead of gathering these measurements in live systems is very low.

**Model Based Scheduling**

Model based scheduling policies leverage insights in workload models to improve scheduling decisions. In this section five modeling/scheduling techniques (multi-objective genetic algorithm, probabilistic SLA guarantees, classification, sliding window demand prediction) are discussed that are on the forefront of modeling workloads for the use in scheduling policies.

Singh et al. [66] use a slot-based model to advertise resources offers. To optimize the application execution in terms of cost and makespan, a multi-objective genetic algorithm is used to select the optimal set of slots (resources). To select the optimal set of resources a good indication of the runtime of an application is needed. Singh et al. use the component performance model developed by Mandal et al. [67] to predict the runtime of applications. Using trace-based simulations Singh et al. show that their approach shows to be promising when it comes to modeling and scheduling resources that are under high utilization. In a comparison with a just in time scheduling policy the provisioning approach described by Singh et al. can reduce the makespan of applications by up to 30%. Netto and Buyya [54] extended the offer based approach by developing policies for scheduling BoT’s with deadline constraints.

Bobroff et al. [68] developed a scheduling algorithm for dynamic VM placement that provides probabilistic SLA guarantees, to make it possible to control the probability of SLA violation they model resource utilization and SLA violation as a tradeoff. Their algorithm uses workload prediction to determine in advance if a VM needs to be migrated to an other physical machine to prevent future SLA violation. To indicate which VM’s are likely to benefit most from dynamic scheduling Bobroff et al. developed a classification model that uses autocorrelation, and the periodic nature of historical workload traces as workload predictors.

Song et al. [69] developed a workload model that takes into account user groups. The model is called Mixed User Group Model (MUGM). This model is especially interesting for scheduling in GRID computing where scheduling objectives are not globally given but depend on user preferences. This could however also apply to enterprise datacenters where different user groups might have different SLAs that in turn influence the scheduling policies. Song et al. characterize the users with a feature vector; these vectors are used to cluster the users in groups. After clustering the users in groups the workloads that belongs to the users are modeled using model-based density estimation [70]. Findings from this research can be used to develop new synthetic workload traces that are based on user group behavior.

Gmach et al. use a demand prediction model based on historical workload analysis to build a capacity management system which advises on the placement of VMs [71]. This publication is different from most of the publications in this field in the sense that Gmach et al. use workload traces from an enterprise datacenter, most publication on resource scheduling focus on grids, scientific computing, and scientific workloads. An interesting finding from the workload analysis is that 77% of the traces show periodic behavior with intervals of 1 day and or 1 week. They also detected other periods (3 days, 5 days, 2 weeks, 5 weeks, and 7 weeks) but these happen far less frequent. For the workload prediction model Gmach et al. use a 5-week sliding window approach, this means that workload traces of the last 5 weeks are used to fill the model with data.

Casale et al. [72] define CWS as a non-preemptive scheduling policy for workloads with a correlated job size where the jobs size s not known beforehand. The authors use Hidden Markov Models HMM [73] to predict the job size of future jobs on the basis of past scheduling decisions, and the past job durations. The
focus of Casale et al. is to prioritize jobs in the shortest job first order.

In a simulation study Casale et al. [72] compare CWS with first come first serve (FCFS), last come first serve (LCFS), and with shortest expected processing times first (SEPT [74]). The major difference between SEPT and CWS is that SEPT assumes prior knowledge of the jobs class membership (job runtime class) whereas CWS uses probabilistic models to infer job class membership. The results of the simulations show that CWS performs much better than FCFS and LCFS for medium to highly correlated workload traces, in some cases CWS produces near optimal job sequences. For un-correlated workload traces CWS scores precisely the same as FCFS which is expected. SEPT always scores better but this is also expected because SEPT has prior knowledge of job class membership which leads to optimal ordering, whereas CWS does not.

**Meta Scheduling**

As discussed earlier in section A.1.2 many datacenters have to deal with a wide variety of workloads in terms of both inter arrival time of jobs, and resource usage characteristics for example CPU, IO, and MEMORY intensity. In recent years many more researches have found their way to the datacenter, resulting in increasingly more scientific workloads having to be processed. Deng et al. describe a portfolio scheduler that uses a set of provisioning policies and a set of allocation policies to schedule scientific workloads for an entire datacenter [46]. Figure A.3 shows the operational model of their portfolio scheduler. From this model it directly follows that at least the following questions need answering:

1. Which allocation policies should be in the portfolio?
2. Which provisioning policies should be in the portfolio?
3. How should the scheduler decide which combination of policies to use?
4. How often should the scheduler make a decision, for every individual job or per time interval (and if per time interval is the choice than the next question is what time interval)?
5. How can historical data be used to improve the portfolio or other parameters used in the model?
6. How can historical data improve the selection process?

Deng et al. [46] use a portfolio with 6 provisioning policies and 2 allocation policies. For the selection process they use a simulation process that scores combinations of all the policies. At the end of the simulation a utility function is used to select which of the combinations of policies scored best. This combination of policies is then used to the actual selection. Deng et al. [46] use a periodic scheduler which in their case means that the selection of policies is done per time interval of 20 seconds. The use of historical data has been left for future work.

Deng et al. [46] test their system with both synthetic workloads and real workloads in a simulator. In the workloads considered during the evaluation of the portfolio scheduler the jobs were all CPU bound,
which according to the authors is a valid model considering the fact that it is successfully employed in grids (Worldwide LHC Computing Grid (WLCG), the French Grid’5000, and the Dutch DAS) for which the authors designed the system in the first place. The model designed and the policies selected by Deng et al. show to be effective for scientific workloads. They also show proof for the intuitive notion that the portfolio scheduler reduces risks and exploits the collective strength of the different policies.

Siqi et al. [75] investigate the cost minimization of VM selection from cloud providers. In their research they use an Integer Programming (IP) approach to build a portfolio scheduler that optimizes the use of different provisioning policies. In the research Siqi et al. use the Amazon pricing model ⁴, in the conducted experiments the authors only consider the small and large instances. To conduct the experiments the authors use real world workload traces from both grids (LCG, Grid’5000), and a gaming platform (Dotalicious). All the workloads considered in this work are CPU bound so during the allocation of jobs and the provisioning of VMs only CPU load, and cost of the VMs are considered. The experimental results show that the system performs better than existing heuristic based systems in terms of reducing the operational cost.

### A.1.5. Future Research Directions

Scheduling is a topic that has been researched to a great extend, it is also a topic that has strong ties with many other topics in the field of datacenter resource management (e.g., networking, storage, energy efficiency, etc.). Most of the research on scheduling similar to the research on storage suffers from the same problem, the single dimensionality problem. Most of the research takes one or maybe two variables and explores these in great depth but ignores other variables that in daily practice can’t be ignored.

Future research there for should focus on integrated scheduling techniques that take into account the multi dimensionality of the problem. It is time to take lessons learned from previous research and combine this into integrated scheduling mechanisms.

One example is the portfolio scheduling approach described in Section A.1.4. It can be further explored in for instance the setting of an enterprise datacenter setting. In this setting it could be used to provision and migrate VMs with different workload profiles. In the setting of an enterprise datacenter it should also take care of not only considering CPU load but also incorporating storage workload, memory workload, network workload, VM locality, and QoS constraints.

In general much research on scheduling is performed using scientific workloads, for the use in the scientific world this is fine. However enterprise workloads often differ quit significantly from scientific workloads. There for it would good for future research to also investigate enterprise workloads.

⁴http://aws.amazon.com/ec2/pricing/
NETWORK USAGE RND SET
Figure B.1: Network transmit usage: (top) CDF for all VMs, and (bottom) CDF of CoV observed per VM.

Figure B.2: Network receive usage: (top) CDF for all VMs, and (bottom) CDF of CoV observed per VM.
Figure C.1: CPU usage: (top) CDF for all VMs, and (bottom) CDF of CoV observed per VM.
Figure D.1: Memory usage: (top) CDF for all VMs, and (bottom) CDF of CoV observed per VM.
CPU AND MEMORY USAGE OVER TIME RND SET

Figure E.1: CPU and memory usage over time.
Figure F.1: Disk read usage: (top) CDF for all VMs, and (bottom) CDF of CoV observed per VM.
**Disk Write Usage Over Time RND set**

Figure G.1: Disk write usage: (top) CDF for all VMs, and (bottom) CDF of CoV observed per VM.
Figure H.1: Correlation between CPU usage and memory usage: (top) CDF and PDF of PCC over time; (bottom) CDF and PDF of SRCC over time.
DISTRIBUTION OF PCC RND SET
Figure I.1: Correlation between CPU usage and (top two plots) network receive usage, and (bottom two plots) network transmit usage. For each group of two plots: (top) CDF and PDF of PCC over time; (bottom) CDF and PDF of SRCC over time.
Figure I.2: Correlation between CPU usage and (top two plots) disk read usage, and (bottom two plots) disk write usage. For each group of two plots: (top) CDF and PDF of PCC over time; (bottom) CDF and PDF of SRCC over time.
Figure I.3: Correlation between disk read usage and network transmit usage: (top) CDF and PDF of PCC over time; (bottom) CDF and PDF of SRCC over time.

Figure I.4: Correlation between network receive usage and network transmit usage: (top) CDF and PDF of PCC over time; (bottom) CDF and PDF of SRCC over time.
J

PEAK TO MEAN RATIO RND SET
Figure J.1: Peak to Mean resource usage, over time: (top two plots) CPU, (middle two plots) Disk read, and (bottom two plots) Network transmit. For each group of two plots: (top) hourly data; (bottom) daily data.
AUTOCORRELATION RND set
Figure K.1: Auto-correlation function: (top) CPU, (middle) Disk read, (bottom) Network transmit.
NETWORK USAGE PER CLUSTER PER SCENARIO DURING SIMULATION
Figure L.1: Network send usage in packets per second per cluster per scenario.
Figure L.2: Network receive usage in packets per second per cluster per scenario.
Storage Usage per Cluster per Scenario During Simulation
Figure M.1: Storage read usage in I/O per second per cluster per scenario.
Figure M.2: Storage write usage in IO per second per cluster per scenario.
CPU Usage per Cluster per Scenario During Simulation

Figure N.1: Distribution of CPU workload.
Figure O.1: Distribution of mean CPU workload.

Figure O.2: Distribution of mean Memory workload.

MEAN USAGE PER CLUSTER PER SCENARIO DURING SIMULATION
Figure O.3: Distribution of mean IO read workload.

Figure O.4: Distribution of mean IO write workload.

Figure O.5: Distribution of mean network send workload.

Figure O.6: Distribution of mean network receive workload.