UMEÅ UNIVERSITY

Autonomous Resource Management for High Performance Datacenters

Abel Souza

Doctoral Thesis, April 2020
Department of Computing Science
Umeå University
Sweden

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

http://asouza.io
abel.souza@cs.umu.se

ISBN 978-91-7855-286-3 (print)
978-91-7855-287-0 (digital)
ISSN 0348-0542
UMINF 20.03

Printed by Cityprint i Norr AB, Umeå, 2020

"All mysteries are simple once you know what they are doing."

Abstract

Over the last decade, new applications such as data intensive workflows have hit an inflection point in wide spread use and influenced the compute paradigm of most scientific and industrial endeavours. Data intensive workflows are highly dynamic and adaptable to resource changes, system faults, and also allow approximate solutions to their models. On the one hand, these dynamic characteristics require processing power and capabilities originated in cloud computing environments, and are not well supported by large High Performance Computing (HPC) infrastructures. On the other hand, cloud computing datacenters favor low latency over throughput, deeply contrasting with HPC, which enforces a centralized environment and prioritizes total computation accomplished over-time, ignoring latency entirely. Although data handling needs are predicted to increase by as much as a thousand times over the next decade, future datacenters processing power will not increase as much.

To tackle these long-term developments, this thesis proposes autonomic methods combined with novel scheduling strategies to optimize datacenter utilization while guaranteeing user defined constraints and seamlessly supporting a wide range of applications under various real operational scenarios. Leveraging upon data intensive characteristics, a library is developed to dynamically adjust the amount of resources used throughout the lifespan of a workflow, enabling elasticity for such applications in HPC datacenters. For mission critical environments where services must run even in the event of system failures, we define an adaptive controller to dynamically select the best method to perform runtime state synchronizations. We develop different hybrid extensible architectures and reinforcement learning scheduling algorithms that smoothly enable dynamic applications into HPC environments. An overall theme in this thesis is extensive experimentation in real datacenters environments. Our results show improvements in datacenter utilization and performance, achieving higher overall efficiency. Our methods also simplify operations and allow the onboarding of novel types of applications previously not supported.

Sammanfattning

Dataintensiva workflows är en ny klass av applikationer som blivit alltmer vanliga under senaste årtiondet och har stor påverkan på hur beräkningar utförs inom flertalet forskningsområden och i industrin. Dessa dataintensiva workflows kan dynamiskt anpassa sig till ändringar i resursallokering, systemfel och kan ibland även approximera lösningar vid resursbrist. De kräver hög beräkningskraft och därtill funktionalitet som endast återfinns i datormoln och de passar därmed dåligt i dagens högpresterande datorsystem (HPC-system). Datacenter i molnet prioriterar att snabbt starta nyinkomna applikationer, vilket drastiskt skiljer sig från HPC-miljöer där hög genomströmning över tid är det främsta målet. Trots att behovet av datahantering uppskattas öka mer än tusenfallt under kommande årtioende kommer framtidens datacenter inte att ha motsvarande utveckling av beräkningskapacitet.

Denna avhandling möter dessa utmaningar genom en kombination av autonoma system och nya strategier för schedulering för att optimera utnyttjandegraden i datacenter. Detta sker utan att göra avkall på användares prestandakrav och därtill med målet att stödja ett brett spektrum av applikationer och scenarios. Ett bibliotek utvecklas för att dynamiskt anpassa resursallokering för workflows under körning, vilket innebär att även HPC-system kan stödja elastiska applikationer som tidigare bara kunde exekveras i datormoln. För miljöer med höga krav på tillgänglighet defineras en regulator för att dynamiskt anpassa hur applikationer synkroniserar tillstånd, för mer resurseffektiv aktiv replikering. Avhandlingen utvecklar även flera resurshanteringssystem baserat på schedulering med förstärkningsinlärning i syftet att förbättra stödet för dynamiska applikationer i HPC-system. Ett övergripande tema i avhandlingen är omfattande utvärderingar av de framtagna metoderna och systemen genom storskaliga experiment i verkliga datacenter. Resultaten visar förbättringar överlag av resursutnyttjande och prestanda i datacenter. De utvecklade systemen förenklar även drift och möjliggör nya typer av applikationer som tidigare ej kunnat exekveras i HPC-miljöer.

Preface

This thesis contains a brief description of datacenter infrastructures, a discussion on how to improve the performance efficiency in datacenters through the use of autonomic techniques, and the following papers.

Paper I W. Fox, D. Ghoshal, **A. Souza**, G. P. Rodrigo, and L. Ramakrishnan. E-HPC: A Library for Elastic Resource Management in HPC Environments. *Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science (WORKS, 2017)*, ACM, pp. 1-11, 2017.

Paper II **A. Souza**, A. V. Papadopoulos, L. Tomás, D. Gilbert, and J. Tordsson. Hybrid Adaptive Checkpointing for Virtual Machine Fault Tolerance. *Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E, 2018)*, pp. 12-22, 2018.

Paper III **A. Souza**, M. Rezaei, E. Laure, and J. Tordsson. Hybrid Resource Management for HPC and Data Intensive Workloads. *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2019), pp. 399-409, 2019*

Paper IV **A. Souza**, K. Pelckmans, D. Ghoshal,L. Ramakrishnan, and J. Tordsson. ASA – The Adaptive Scheduling Architecture. *This report is an extended version of a paper under the same title to appear at the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC, 2020).*

Paper V **A. Souza**, K. Pelckmans, and J. Tordsson. A HPC Co-Scheduler with Reinforcement Learning. *Submitted, 2020.*

Financial support for this work is provided in part by the Swedish Research Council (VR) project Cloud Control, the Swedish Government's strategic research project eSSENCE, by the European Union's Seventh Programme for Research (FP7), Technological Development and Demonstration under Grant Agreement No 609828, and by the Brazilian National Council for Scientific and Technological Development (CNPq) under project no 207555/2014-1.

Acknowledgements

After five long years, this PhD journey is Phinally Done. As they say: It is never a one-person's work, and therefore I want to thank the people who helped me on the way.

Firstly, I would like to thank my supervisor, Johan Tordsson, for his consistent and continuous support throughout my PhD. All opportunities for improving my research skills were only possible because of his mentorship, advice, patient supervision, and motivational and helpful catches with the Oxford commas. His wise, humble, and broad perspectives helped me strengthen my computer science view in many different areas, and for this I am very grateful. Your help is very appreciated and valuable to me as a young scientist, and is something I will take for life.

Secondly, I thank my co-supervisor Erik Elmroth for giving me general insights on how to tackle research, networking opportunities, and directions on what to do next. His wise and aimed view on important aspects of distributed systems made our group thrive, going beyond the edge. I meet you later at Lotta's.

Thirdly, to all collaborators who I have worked with and who have helped us having so many great insights and ideas. Researchers such as Kristiaan Pelckmans, who essential insights and statistical ideas are precious. Thanks to each one of you for the great conversations and discussions! I cannot forget the EU ORBIT Project team: Luis Tomás (I still have to visit you!), Simon Kollberg, and David Gilbert. Together with Alessandro Papadopoullos, they helped me write my first paper! To KTH researchers, Mohammad Rezaei and Erwin Laure, which contributions allowed us to make an impact. It is amazing what great and complementary people can achieve when they get along well! Another of the great opportunities offered to me was to work in California at the Lawrence Berkeley National Lab. (LBNL), and for this I thank Lavanya Ramakrishnan for accepting to work with us. Your long term lessons remains. Also big thanks to the collaborators who were closer to and helped me in many problems and discussions, Devarshi, Gonzalo, William, and to all the great people at the Data Science and Technology Department.

I also would like to thank the very great researchers I met at the Autonomous Distributed Systems Lab.: Ahmed, Amardeep, Chanh, Cristian, Danlami, Ewnetu, Francisco, Jakub, Lars, Mina, Monica, Magdalena, Monowar,

Muyi, Peter, Petter, P-O, Selome, Thang, Tobias, Ali, Javad, and Mohammad. Make sure to remember these people's names. Special thanks to Tomas Forsman and Mattias for providing all the magical-technical support, for which I am very grateful. This includes our friends at the Department of Computing Science and UMIT Lab., including Monika, Maitreyee, Michele, Sonvx, Nazanin, Angelika, Lili, Birgitte, Addi, Ahmad, John, Mats, Anne-Lie, Mikael Hansson, Helena, Julian, Lennart, Lili, Carl Christian, Mirko, and so on and so forth :) I cannot forget the invited people with great TED talks at Universum: Juan Carlos, Edwardinho, Merko, Marko, and João. To good Umeå friends, Katya, Gabriela, Felipe, Simone, Vladimir, Martin, Botswana, Theodor, Mahmoud, Eddie. Berkeley many friends: Tupynamba, Ezgi, and Ellis, besides the international and random groups! To my university friends, Rodrigo, Marco Gabriel, Diego, and Hudson, and all other Brazilian friends, including Rhian, Leo, Bigua, Helaynne, Alcker...

Last but not least, to my country (Brazil), who means (specially professors from the Federal Fluminense University (UFF)) helped me getting to another amazing country (Sweden). To my parents Leila and Francisco, who crafted the base of all that I am. To my brothers Rafael and Thalmo, who always believed on me and on the "FLART" potential. Together with Luquinhas, Debora, Dona Mirian, and ?. We will expand! To good friends, Lija and Geni, who together can make any travel look like super-fun. To my partner, Laura, with whom eternity is possible. Words are not able to describe what I feel about these people.

Obrigado, reader!

/Abel Souza

Contents

Chapter 1

Introduction

Datacenters are the main computing infrastructure enabling Internet services, as well as scientific endeavours addressing pressing challenges in e.g., medicine, chemistry, biology, materials design, and climate change. These infrastructures allow biological, weather, and other natural science models to be tested and evaluated, and are thus key enablers for high performance (HPC) and cloud computing [Ste+20]. However, to provide acceptable levels of performance, datacenter infrastructures consumed roughly 2% of electricity worldwide just in 2018, with projections rising it to 10% by 2030 [CP15; Jon18]. In fact, in this decade we will see datacenters handling yottabyte data amounts (= 1 trillion-terabytes), although today's data processing capacities will not increase as much. These figures show demand is soon outpacing supply, triggering serious datacenter efficiency concerns, slowing society digitization advances down [Mit16], with no clear path delineating software system solutions to handle such enormous scales in data.

Moreover, over the last decade, Artificial Intelligence (AI) has become a new paradigm in most scientific endeavours, hitting an inflection point in adoption and specially in its possibilities. Nonetheless, the easier to run state-of-the-art Machine Learning (ML) models, mostly triggered from sensors in small Internet-of-Things (IoT) devices requesting processing power from larger datacenters, the higher demand these infrastructures face. Recent software developments such as *approximate computing* leverages on the idea where applications progress depend on data estimates, and not on exact data inputs/outputs (I/O). This conceptual idea can likewise be applied to other metrics such as time to results, whereas estimates are set to produce data products rather than exact deadlines. Unfortunately, most of today's datacenter software systems do not embed applications internal dynamics into their decision-making, restricting opportunities to various types of optimizations, and resulting in low ratios of datacenter processing power engaged in actual work. From tiny sensor devices connected to mobile edge datacenters to larger cloud datacenters, if correctly integrated into datacenter management, such dynamic application features own

high potential to enable the high performance efficiency required for future datacenter realizations [Ste+20].

Thus, it is important that resource management happens in a way that satisfies datacenter end-users (also called *users*) and operators. Users are the ones using or developing and coding applications, and ultimately depend on the datacenter to run their applications, often making use of modules, libraries and runtime systems to facilitate and automate infrastructural operations. Operators maintain and provide fair means to support end-users with the access to the datacenter infrastructure through a resource management system, following a multi-tenancy policy, which describes how resources can be used and shared. At datacenter's scale, resource managers have the role of an Operating System (OS) because these are the layers of software that transparently abstracts and manages the infrastructure's resources to users and their applications. Understanding the various tradeoffs while allocating and serving resources, besides adapting applications runtime mechanisms and capacities according to the workload's variations lead to operational efficiency gains because it tailors the infrastructure to the application dynamic needs, instead of the other way around, which is having applications change to infrastructures. Finally, a timely allocation of resources to applications ensures efficiency, operational cost reductions, and shorter time to solutions.

1.1 Research Problem and Objectives

The purpose of this thesis is to propose and evaluate autonomic solutions combined with novel scheduling strategies and software architectures to increase datacenter performance and utilization, while seamlessly supporting a diverse set of applications under various and real infrastructure and operational scenarios. These scenarios may include behaviors considered anomalous, such as resource failures, resource transiency, shortened deadlines, and resource performance variability. Our aim is to minimize negative impacts on applications constraints and expected Quality-of-Service (QoS) while also improving overall datacenter efficiency and utilization. Combined with that, the main research objectives of this thesis are:

RO1 To allow users to specify application requirements to enable and improve datacenter efficiency and utilization of applications.

RO2 To develop theoretical control techniques for adapting and choosing best fault-tolerant mechanisms according to workload variations and application characteristics.

RO3 To evaluate and develop machine and reinforcement learning methods and resources capacity controllers to continuously improve datacenter throughput application performance and accuracy.

RO4 To develop adaptive scheduling techniques that choose the best performance tradeoffs according to application workload variations and time characteristics.

1.2 Methodology

The methodology used in this thesis is a combination of formal deduction with scientific experimentation, in concert with the scientific paradigm, where *a priori* and *a posteriori* knowledge about the proposed methods are sought [Ede07]. To model **RO1** and **RO2**, we set up testbeds consisting of multiple servers in use in real clusters. To generate load on the testbed we run known benchmarks tailored to harness selected computational resources, and real workflows modeling scientific experiments. When the servers are exposed to the load, we dynamically (at workloads' runtime) modify various configurations and measure the performance of running applications.

To address **RO1**, **RO2** and **RO3**, we analyze the measurements using control theory and various statistical methods in order to remove outliers, summarize multiple measurements, and evaluate hypothesis. To evaluate **RO3** and address **RO4**, we apply and design predictive methods, from machine and reinforcement learning, to process and model vast amounts of data to understand the quantities and types of resources to allocate applications with, and when and where to deploy them within a datacenter. Furthermore, we use statistical methods to correlate dependencies between application performance metrics and compute resource metrics, such as CPU and memory utilization to latency and/or throughput. We evaluate our methods through implementation and integration with real datacenter resource management software. We further conduct extensive experiments in real datacenters, often directly in HPC production systems under a variation of applications and loads, as well as tuning parameters for our methods.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of Cloud and High Performance Computing datacenters, describes the role of resource management in both infrastructures, their execution models, and a research overview. Chapter 3 explains how application performance can be supported in different situations, describing how trade-offs are achieved. Chapter 4 introduces why the autonomic management of datacenters is needed, illustrating statistical methods that can be used to support it, and its challenges. Finally, Chapter 5 concludes by discussing the research contributions presented in this thesis, discusses its limitations, and future directions.

Chapter 2

Datacenters and Resource Management

In this chapter we describe what a datacenter is, and the role of the software managing these large infrastructures, the resource manager. We also explain the various mechanisms that allow the control of a datacenter and that enable it to extend its functionality to provide new features to users. We give an overview of concepts such as collocation, isolation, resource control and scheduling, link the approaches to enabling technologies and datacenter actuators, and overview the research performed in each area.

Datacenters

Datacenters have become ubiquitous and important to nearly all aspects of communication, business, academic, and governmental information systems [WSI05; And+18]. These infrastructures are designed to respond in real time to user applications scattered around the world, using cell phones, tablets, or personal computers. Figure 2.1 outlines a high level view of a datacenter, composed of different resources such as computer servers and storage devices. These are the hardware equipment that allows a datacenter to execute computer software (application), such as data processing, storage, and communication, inside (intra network) and outside (Internet) its premises.

Resource Manager

As a central entity controlling a datacenter [JS15], the resource manager (RM) is the main system a user has to interact with in order to use resources and run services or other computations. The RM is designed to follow different organizational policies that describe how datacenters can be used. For instance, in Figure 2.1 the datacenter is grouped in three clusters with different compute capacities (Small, Medium, and High Performance Compute *clusters*), and

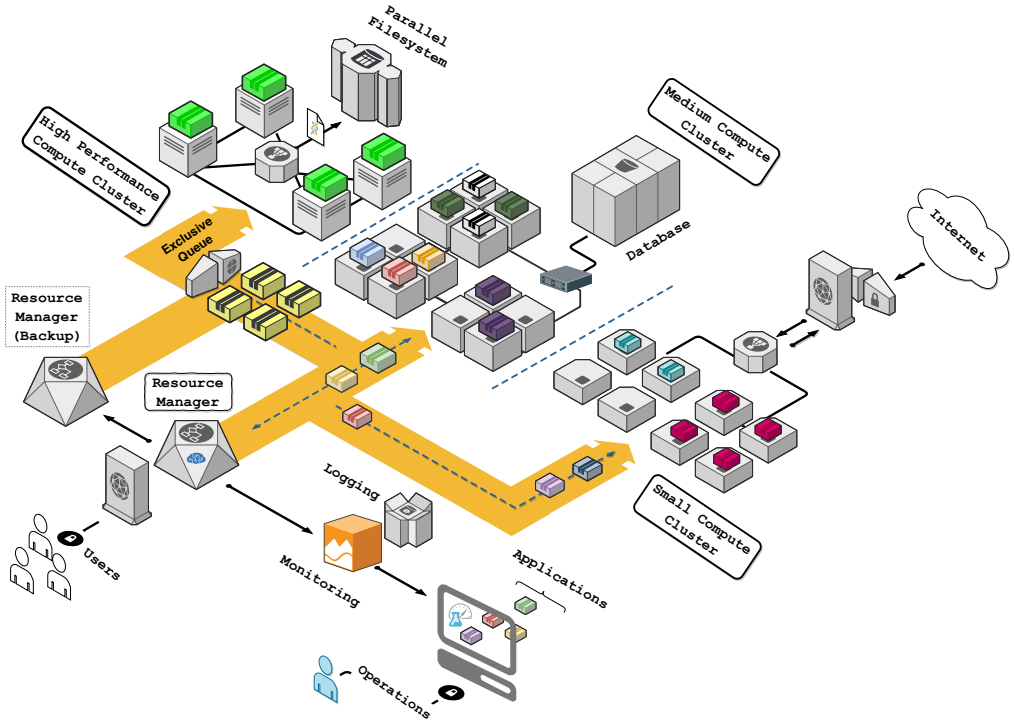


Figure 2.1: Datacenter overview: Applications (coloured boxes) are submitted by users, and requested resources allocated in cluster compute nodes by the resource manager. Operators profile and monitor applications and datacenter utilization in real time. The network connects all cluster nodes internally, to databases, and remote storage.

according to user requirements the RM schedules applications to different clusters. Current datacenters installations require RMs to be very flexible to support behaviors considered anomalous and commonly transparent to applications, such as power failure, server faults, and compute processing shortages. The RM also supports distributed applications, which is software that is able to run on multiple computers connected in a network, where instances interact through this network to accomplish a specific task. Conversely, traditional applications run on a single system only. These applications (also referred to as *jobs*) are usually encapsulated and orchestrated through virtual machines or containers, or classic Operating System (OS) processes [Bur+16].

From a system's perspective, in addition to handling the actual execution of jobs (coloured boxes in Figure 2.1), the RM is responsible for efficient job management such as maintaining high utilization and throughput (performance), as well as handling software and system faults gracefully (i.e. without the user noticing). In addition, from a user perspective, the RM should improve execution

times and fairness among different workloads, users, and projects [FR96]. The queue makespan, a metric which is defined as the total amount of time a given set of jobs takes to finish execution, is one important example of a metric users and operators want to reduce.

2.1 Applications

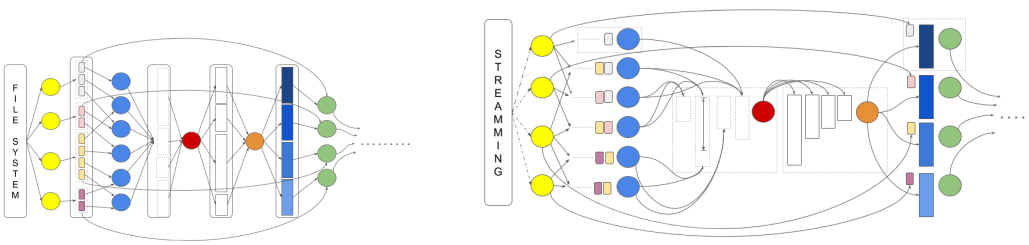
There are commonly two types of applications: interactive and batch [Reu+18; BCH13; Jha+14]. Interactive applications are commonly web requests such as ride-sharing mobile applications asking for directions, or tasks in a data analysis pipeline, which runs in datacenters hosted by providers, such as Amazon Web Services (AWS) or Google Compute Engine (GCE). Interactive applications are commonly described because of their low latency characteristics, i.e. the response to a request has to be answered very quickly, otherwise the service being offered may be considered unusable. For instance, a query in a search engine should be responded in less than 1s or users may go to the competitor.

The second type are batch applications, or background computations, i.e. a sequence of commands in a file (also known as batch file, command file, or shell script) executed by an Operating System (OS) and submitted for execution as a single unit, or process. Batch workloads are common in scientific computing, historically running in High Performance Computing (HPC) environments to enable large scale experiments. As an instance, installing an application in a workstation is considered a batch workload because many sequence of commands should be completed to successfully setup the application for use.

Scientific Workflows

Large scale experiments rely on datacenters and big computing infrastructures. Because these scientific experiments are very large and time consuming, scientists split the overall problem in independent parts, which are later combined to produce final data products [Dee+18]. Figure 2.2 illustrate two workflow pipelines, where each (coloured) stage describes a specific set of models and computational tasks organized in batch calculations (the small rectangles passing through). At a high level, these interconnected pipelines resemble the MapReduce model [DG08], where the directed acyclic graph (DAG) composition of independent stages is the actual *scientific workflow*. Moreover, scientific workflows are not only common in HPC centres, but also in virtually every sector in industry and academia. Common applications analyze and correlate data for predictions and decision support, where users can customize/sweep parameters in a model without viewing or altering any code, making them vastly flexible and reusable.

Intrinsically, a workflow pipeline structure describes the number of resources required to perform a batch (computation) task in each stage of a scientific workflow. Such a pipeline is managed by a workflow management system (WMS). The purpose of a WMS is to aid in the automation of execution of tasks and the information exchanged between these tasks, with a special focus



(a) Traditional Workflow (b) Streaming Workflow

Figure 2.2: Scientific Workflow: (a) Traditional, and (b) Streaming pipeline structures for Montage Workflow, an image mosaic software [Ber+04]. Each color in the graph describes a set tasks within a stage. Each stage produces outputs used as inputs at subsequent stages that produce the data product at the end.

on reliability. The task of ensuring acceptable performance is delegated to developers. Workflows are traditional (Figure 2.2(a)), or streaming (Figure 2.2(b)). Traditional workflows process tasks in each stage sequentially, one stage at a time. Conversely, in streaming workflows, often used for in-situ processing, tasks in a stage are processed and followed to subsequent stages as they are generated, with low latency characteristics similar to interactive applications. Finally, with increased use of workflows to process great amounts of data, closer integration between the WMS and the datacenter RM is of vital importance for improving scientific application performance [Dee+18; Com+16; Asc+18].

2.2 High Performance Computing

High Performance Computing (HPC) organizes independent compute resources in clusters that can deliver more performance and solve bigger problems than could be solved from a single personal computer or workstation [And+18]. HPC clusters solve and steer complex problems in scientific experiments, engineering, and business, and are key for innovation [She+16]. As mentioned, distributed applications use multiple nodes to accomplish a goal, usually performed through the network by means of standardized libraries such as the Message Passing Interface (MPI) [WD96]. The use of one type of network over another usually depend on the bottlenecks most workloads experience in an HPC centre. HPC clusters require users to provide a walltime (i.e. time limit) for executing jobs [Reu+16]. However, jobs not only finish execution earlier than specified walltimes, they often use less resource capacity than what is provided at job launch. Usually users cannot use more resources than what is allocated, which is commonly determined by which project or organization they belong to, to which

compute time is distributed by an allocation committee. Some tools allow users to more efficiently utilize their allocations [Ber17] by offering mechanisms for bundling jobs together in optimal ways, and mechanisms for migrating jobs to other resources without loosing completed work. However, there are still some drawbacks as important features such as state management and monitoring are not fully integrated into HPC schedulers.

2.3 Cloud Computing

Cloud computing is a model of computing where applications run on shared computing and storage infrastructure in large-scale datacenters instead of the user's own computers [JS15]. It must address many of similar issues faced in OSes in terms of resource sharing, abstraction, and common services. This happens because of the diversity of applications that clouds can accommodate: basically any type of application can run alongside with one another, which compete for and influence how resources are used. Cloud RMs such as Mesos [Hin+11], Kubernetes [Bur+16], and Yarn [Vav+13] provide APIs enabling jobs to control resource assignment, conduct state management and perform resource profiling and monitoring. Cloud RMs are developed as flexible frameworks of execution engines, which can be ported to different infrastructural contexts [JS15]. As their main objective is to maximize utilization, most cloud RMs allow application collocation with different policies support (including for HPC workloads). Although some HPC RMs supports finer grain allocations, they come with no support to enforce resource isolation when sharing resources (see Section 2.4).

Characteristics

The main difference between cloud computing and HPC lies in the model of delivery or access to computing resources as well as associated costs [Fos+08]. In traditional HPC, the institution (i.e. the operator) typically creates policies and organizes the infrastructural management following users' workload characteristics and priorities. By owning the infrastructure, the operator incurs large capital expenditures (also known as *capex* costs) e.g. cost of servers, hiring software engineers, warehouse rent, and etc. Additionally, there are recurrent operational expenditure (known as *opex* cost), e.g. power and cooling, wages, and system upgrades. This cost is constant independently of whether the infrastructure is utilized. Cloud Computing, on the other hand, is an economical model where all capex and opex costs are passed to a third party, known as cloud provider such as Amazon Web Services, Microsoft, and Google. In this model, cloud providers buys, maintains, shares, and utilizes the computer infrastructures. Users have access to a virtualized infrastructure accessed through the Internet, with a pay-per use billing scheme. Users personalize this virtual infrastructure by using various runtimes and application environments such

as containers, virtual machines, and orchestration tools such as Mesos and Kubernetes, which help them managing this complex environment.

2.4 Resource Manager Components

A RM provides four main functionalities that support job management, as illustrated in Figure 2.3: job life-cycle, resource monitoring, scheduling, and job execution [Reu+18]. Job's life-cycle management is responsible for receiving user's jobs through a user interface such as the command line or a web interface. Users provide the job's requirement (or geometry) such as specific resources (i.e. CPU cores, memory, network bandwidth, and other resources), the amount of each, and/or time constraints, such as total execution time and/or deadlines. Also, users can specify jobs requiring elastic execution such that they change their resource geometry in the middle of execution without halting execution. Job's lifecycle management thus places jobs into the appropriate queue for execution. RMs makes cluster resources (such as compute nodes and CPUs) accessible for use by jobs, while the scheduler allocates the resources to execute the job, and assigns these resources based on datacenter policies and availability. Job execution is a process in which jobs start executing on each node, after which they can be manipulated by the job lifecycle management, which in turn also allows the application to communicate directly with the scheduler (and vice-versa) in order to help it take appropriate management decisions in case of failures or workload variations. Job monitoring and profiling provides interfaces accessing application and system Key Performance Indicators (KPI), allowing techniques to statistically estimate and analyze resource demands and needs, possibly in real-time. For instance, it can enable flexible options for users to manage and use their own resource allocations, allowing users to co-schedule multiple of their jobs' tasks in varied ways not currently possible by static libraries. In this context, KPIs are user monitored metrics, specific to the service or application in question and describe its performance.

Job Characteristics

Jobs are classified by four runtime characteristics: rigid, moldable, malleable, and evolving [FR96]. Specifically, these classes differ in what can happen to the resources allocated throughout jobs' lifespan. Whereas a moldable job may have its resources changed before being launched, a rigid job cannot. From a RM's point of view, these jobs are considered the same and called *rigid* jobs. Malleable and evolving jobs are also considered the same and simply called *malleable*, because they can change resource at runtime. With AI jobs and large data processing needs becoming a norm a novel fifth class has emerged: *adaptive jobs*. Such jobs are highly dynamic and adaptable to changes and system faults. Because they need to process unprecedented amounts of data, they are known as data intensive (DI) applications. In relation to resource needs, adaptive jobs combine characteristics of malleable and evolving jobs [Pra+15].

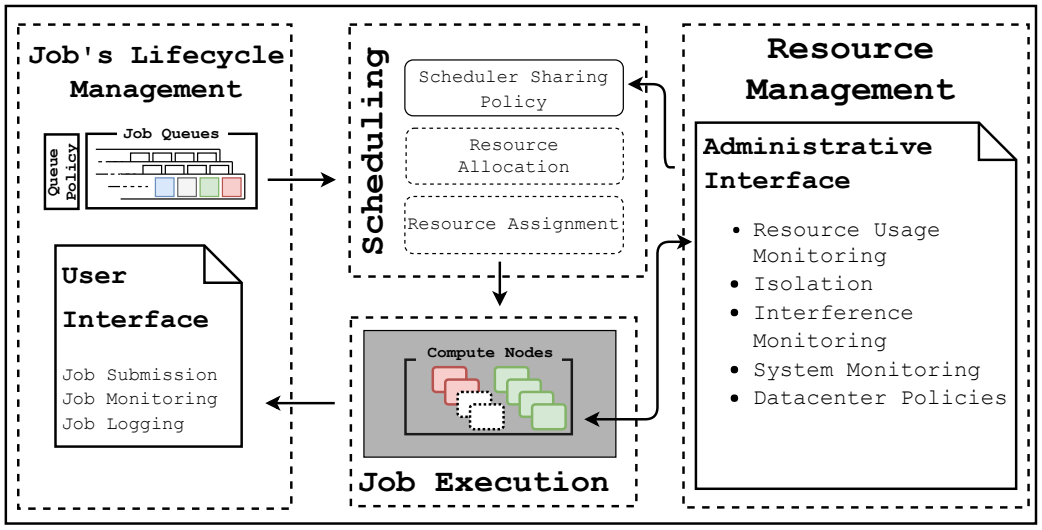


Figure 2.3: Components of a typical Resource Manager: job lifecycle, scheduling and resource management in datacenters. Users submit jobs through an interface. Jobs are queued and scheduled for execution. Each one of these steps is profiled and monitored according to the policies set in the management component. Adapted from [Reu+18].

Resource Allocation

In this context, resource allocation refers to assigning datacenter's resources to user requests (specified through jobs) according to its goals and objectives. As an example of a policy is the maximization of datacenter resource utilization. That is, the focus is on measuring how well, or how efficiently, resources in a datacenter are being utilized by applications. From the applications context, it is defined as how efficiently a given resource capacity is available to the application following its needs. Operators aim at maximizing the use resources, and such policies contradicts each other and may negatively impact users KPIs. In either context, utilization is often used as the main metric of comparison among different techniques in RM systems because it clearly relates the application performance of a given workload to the resource capacity available to the application.

Resource Sharing

Each RM uses different sharing schemes and policies for multiplexing managed resources, depending on the context applications are deployed, e.g. containers. Runtime systems and orchestrators are the software that supports the execution of containers, and they have to handle, measure, and evaluate the different organizational policies and their effect to users' KPIs. Common runtime systems

in large clusters may rely on frameworks such as Mesos [Hin+11], Slurm [YJG03], Torque [Sta06], or Kuberentes [Bur+16], to allocate and share resources to their jobs and tasks. When sharing resources such as CPU, memory, network, and file systems (I/O), resource isolation is a requirement since applications may end up competing for a common resource. Isolation specifies that minimum capacity levels are available when needed, allowing controllable behavior given to most applications. However, not all of these aspects are fully integrated into modern datacenter operations, and therefore cannot enable the extreme efficiency required by future installations.

2.5 Other Characteristics

Other characteristics like scheduling and reliability influence how resource managers and policies are combined and used. In addition, new OS capabilities also enabled the development of the new specialized resource managers, known as orchestrators.

Orchestrators

With advents of lightweight in-kernel virtualization and isolation tools such as cgroups [Men07], Linux containers (LXC) [Men07], and Docker [Mer14], resource orchestrators such as Kubernetes [Bur+16] and Docker Swarm [Rou16] have also been used in large infrastructures because of the new levels of resource management offered by these tools. By leveraging upon such tools, some RMs allow tasks within a job to also specify/request their resource geometries, enabling new features and challenges in datacenter resource management. For instance, Mesos and Kubernetes, RMs commonly used in cloud datacenters, support Docker containers and Linux namespaces [Men07], but are mainly designed to improve fine-grain resource utilization (within servers, i.e., ratio of CPU and/or memory capacities used). These RMs use finer grained resource allocation by taking into consideration fractions of resources needed to run a job.

Scheduler Objectives

In HPC most jobs are batch scripts which run for long times and occupy high parts of a cluster [Reu+18]. As such, common RMs used in HPC centres are designed mainly to achieve high resource allocation at coarse-granularity levels, such as compute nodes. In this context, resource utilization refers to the infrastructure ratio that is occupied by multiple jobs (space-sharing) at a time, and not to how efficiently allocations are utilized within these resource capacities (time-sharing), which is the case in the cloud model.

Moreover, scheduling algorithms such as Completely Fair Scheduler (CFS) and Dominant-Resource Fair Scheduling (DRF) allows for fair resource allocations in time-sharing contexts such as OSes and in clouds. CFS keeps track

of the fair share of resources (e.g., CPU) that would have been available to each process in a system [Pab09]. In modern OSs, CFS is configured by using mechanisms such as namespaces and cgroups [Men07], offering extended capabilities for isolation enforcement. DRF proposes a notion of fairness across jobs where the jobs have multi-resource (e.g., number of CPU, amount of memory and network bandwidth) requirements [Gho+11]. In HPC, Backfilling is one of the most used schedule algorithms in space-sharing contexts [Sri+02]. Its main advantages is increased utilization at the cluster level, lowering queue waiting time.

Reliability and Availability

Given their importance, mission critical applications such as banking, hospital, and airline services must be highly available for use. Availability is the percentage of time an infrastructure or a service is running in an operable state. Outages severely impact services reputation and overall users' satisfaction, and thus most datacenters typically aim for at least 99.99% uptime, which is approximately one hour of downtime over a year period.

Leased resources in a datacenter are subject to Service Level Agreements (SLA) that specify what a service provider general complies with. On the other hand, some users are often more concerned about Service Level Objectives (SLOs) for alloted resources, which are performance indicator levels such as "$latency < X(units)$ or $throughput > Y(units)$", and that make up the SLA. The SLA is the overall contract, where the SLO are the performant terms the provider aims to deliver. Thus, the SLO can be either uptime, or throughput, or something else.

Reaching high availability or zero fault operations on large systems is hard and even more difficult at extreme scales. It is possible to prevent failures in a collection of thousands of servers at costs of deploying hardware resource replicas, that is duplicating the execution of a system of interest. Common high availability software techniques use checkpoint and restart mechanisms and/or user request duplication, where one request is executed two or more times, and in case of failures, a secondary replica takes the execution over the failed component. Consequently, workloads and runtime systems are designed to gracefully tolerate and adapt to component faults with negligible impacts to SLAs and SLOs.

Chapter 3

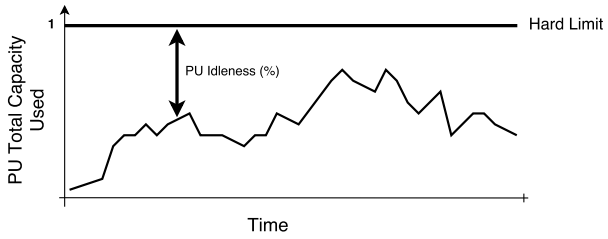
Resource Management Trade-offs

In this chapter we talk about trade-offs and approaches to improve the performance efficiency of datacenter resources. By now we have been looking e how different systems behave when working in isolation. In reality, though, these resource subsystems do not work independently (i.e. in isolation), they rather depend on each other. We give an overview on concepts such as consolidation, throttling and scheduling, link the approaches to enabling technologies and datacenter actuators, review the research performed in each area, and present the associated challenges and limitations.

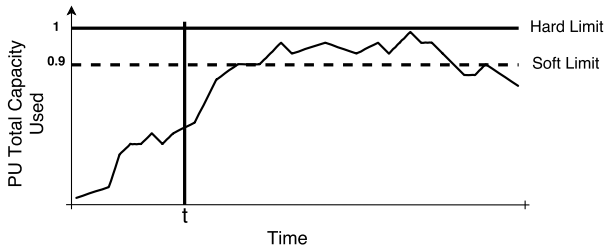
3.1 Performance Trade-offs

Resource assignment can be a challenging problem, specially in clusters with heterogeneous resources, where compute nodes with different configurations and architectures are combined. For heterogeneous environments, dynamic RMs are commonly used since they are able to cope with variations and faults within the infrastructure [Reu+18; Ahn+14; Jha+14]. In traditional HPC RMs, allocation is the exclusive assignment of resources to execute a job [Jha+14], which means that the resource request describes the exact amount of resources the RM allocates to the job, which is the common SLO that most HPC clusters support.

For different goals and objectives (i.e. policies), the way resources are shared are very important. *Time-sharing* and *space-sharing* refer to the way resources of a machine or a cluster are shared among jobs. In time-sharing, several processes typically take turns in accessing the resource (e.g., a compute server or a CPU). On the contrary, when resources are assigned exclusively to individual processes, different processes share the infrastructure spaces (the resources). A time-sharing RM such as Slurm [YJG03] with a space-shared



(a) Isolation run



(b) Collocated run

Figure 3.1: Resource utilization in two allocation strategies: isolated and collocated. The soft limit sets resource limits which eventually might be crossed. Hard limits are physical, and if 'crossed' may result in applications performance degradation.

policy assures predictable resource performance to jobs at the cost of higher queue-makespan [Amv+18].

Resource Utilization

Utilization is often used as the main metric of comparison among different techniques in distributed computing systems, and even different research areas. It is defined as how efficiently a given resource capacity is used by a workload, where operators often try to maximize the benefits for those who scheduled its needs to specifics resources. When a resource is shared with any given constraint, such as maximum number of users per second, researchers go towards on ways to optimize its use. Even though it is very intuitive, utilization hides many aspects of what is happening under the hoods in a system. Figure 3.1 generically illustrates the utilization of a processing unit (PU) in two different scenarios: isolated (Figure 4.1(a) and collocated (Figure 4.1(b)). Although in the collocated scenario the utilization is higher, this metric per-se does not indicate how the performance experienced by the application in the isolated run is affected. Systems often depend on other systems with different utilization ratios and would then answer to similar requests quite differently, depending on time and on the workload. A common PU metric is the CPU utilization (CPU %), which measures the time a processor is waiting for memory I/O, and results in the

processor not making forward progress with instructions. By using fine grain monitoring, modern OSes can observe this metric on a per-process basis.

3.2 Strategies and Mechanisms

In order to enable the performance tradeoffs, strategies such as consolidation and mechanisms such as isolation should be combined in a way that do not disrupt the developer's workflow, nor users QoS experience.

Consolidation

Consolidation (also known as co-scheduling) is a common technique in cloud datacenters, though operators tend to use it with care due to the performance interference caused by node sharing. However, with finer granularity in resource allocations, one can expect higher resource utilization [Zha+13; Reu+18], and in large clusters this can have high impacts due to the lower fragmentation of unused resources per node. Additionally, because of resource sharing, a method for enforcing resource isolation among jobs is essential [Zha+13; Bur+16].

Isolation

Isolation mechanisms need to be combined with interference detection techniques [DK13] that use, e.g., classification to weight the impact of different resources for each job, and use this knowledge to select candidates for collocating jobs. In such scenarios, on-line models can also be used to detect, control, and avoid performance interference [NKG10; Yan+13], or to take actions such as throttling low-priority jobs to mend the interference [Zha+13].

Most traditional HPC RMs do not let jobs to dynamically change their placement at the level of nodes inside a cluster, let alone to throttle or to perform low-level resource control in order to enable different isolation mechanisms among multiple jobs and tasks. Although RMs like Slurm can also allocate resources with finer granularity (e.g., CPU-cores), they do not provide the necessary application programming interface (API) and capabilities for application elasticity at runtime (change on the resource geometry), nor mechanisms for enforcing isolation between jobs [Hin+09]. For instance, these capabilities are essential for doing load-balacing in streaming workflows, which demand capabilities such as task migration, or changing resource allocations at runtime [Hin+09; Asc+18].

There are various ways to enforce isolation between co-located tasks (processes) within a node:

- *Operating System Schedulers*: An OS scheduler can be used to dynamically control the prioritization given to jobs while also monitoring application performance. Unfortunately this may not provide enough guarantees for

memory operations because of Last Level Cache (LLC) evictions that could cause severe interference problems [Zha+13].

- *Using a monitoring Agent on the nodes*: This could be implemented by having an exclusive hardware profiler. Though a very promising approach [Sch+13], it needs a specific system architecture for communicating with the RM and it can be hardware dependant [KTC01], limiting its adoption.

- *Linux Cgroup*: Cgroup [Men07] is a set of mechanisms to enforce isolation between containers where processes share resources such as CPU, memory, I/O and network bandwidth. Cgroups also control the way the Linux CSF scheduler calculates weights in container execution. Linux's cgroup resource isolation mechanisms is one of the most available and robust ways to make sure processes, encapsulated as containers (namespaces) do not consume more of the resource capacity than what has been assigned to them.

- *Resource pinning*: This mechanism enables the binding and unbinding of a container, a process, or a thread to a specific resource location, such as CPU, or to a range of CPUs. In this way, the container and its spawned applications execute only on the designated resources (e.g. CPU(s)). Its main use is for load-balacing, which is a strategy to divide the workload equally among available resources, so no one resource has more load than any other resource.

Throttling

Throttling aims at controlling the computational performance ratios an individual process or container receives from a resource, such as CPU or Graphical Processing Units (GPUs). This technique is a combination of the consolidation and isolation techniques, and together they create the computational controlling effect experienced by the target application. For instance, a RM can simultaneously use resource pinning together with cgroups to prioritize CPU intensive processes over I/O intensive processes. It is often used when resources are not fully utilized by hosted processes, i.e. when the resource has a low utilization level.

The challenges concerning throttling approach, that still need to be investigated, are connected with the power-performance tradeoffs. There is still a lack of models of the relationship between server configurations and application performance, to be used for optimisation of multiple co-located applications. Moreover, Mann points out that it is important to investigate how server throttling techniques interact with virtual machine placement algorithms [Man15].

3.3 Multi-Level Scheduling

Modular RMs with multi-level scheduling objectives are proposed to support a diverse set of jobs in a datacenter while also enforcing space sharing to ensure fair-share usage and to meet constraint requirements [Inc16; Sch+13; Hin+11]. In multi-level schedulers, the process of deciding how to schedule and share available resources are given to applications instead of following an unique policy. Static schedulers found in Mesos [Hin+11], Torque, or Omega [Sch+13] expect users to specify resource reservations, for instance how many CPU-cores and memory the application needs. For example, Mesos, processes resource requests and, based on availability and fairness, makes resource offers to individual application frameworks (e.g. Hadoop), which can accept or reject these offers depending on application requirements [Hin+11]. Mesos handles heterogeneity by acting as a meta-scheduler for a whole cluster, with conceptual resource abstractions for CPU, memory, I/O and other infrastructure resources being used and exposed in the same way an OS does in a single computer. This enables a set of new capabilities like elasticity and fault-tolerance to distributed applications [Reu+18], a concept now known as Datacenter Operating System (DC/OS) [Zah+11; Hin+11].

In principle, the idea of allowing multiple distributed applications, which were all developed independently and have their own scheduling policies and requirements, to share resources is very complex. In particular, having a single monolithic scheduler that has to encompass the scheduling decisions from many applications would be particularly complex and not scalable. Mesos simplifies the problem by using an abstraction to separate the allocation of resources and the scheduling of tasks. Similarly, Flux [Ahn+14] proposes an unified layer where applications can decide from what is available within an hierarchical model. Ultimately, the operating system (OS) is the layer and channel providing the real management from which RMs can profile applications to trace dynamic control flow and identify hot-spots for optimizations. By utilizing specific resource controllers to understand and adapt capacity according jobs needs, multi-level schedulers can be harnessed to improve overall resource utilization while also meeting most job contraints.

Thus, to be efficiently managed by a HPC RM, adaptive jobs often combine multiple scheduling policies in a same job and thus require higher degrees of integration with schedulers than what monolithic managers offer. These dynamicity and adaptability are often not fully supported in HPC centers, which demand full resource control to keep-up with Service Level Objectives (SLOs), for e.g. deadlines.

3.4 End Goals

The resource manager is responsible for making important decisions regarding a datacenter's internal operations. Therefore, such decisions affect the perfor-

mance, functionality, and maintenance costs of a datacenter. Softwares such as auto-scalers, elasticity engines, and schedulers provide some solutions to research challenges such as understanding how much and what type of resources to allocate, and when and where to deploy them inside datacenter infrastructures [JS15; Jha+14]. These systems are designed to follow models of application and performance based on few KPIs such as response time and throughput (for applications) and utilization of servers, CPU, memory, bandwidth as well as energy expenditure and heat (for servers) [FR96; Sta06]. However, all these software have limitations because they usually aim for only few aspects of resource management and do not holistically correlate different organizational policies to applications' KPIs [Kat+11; Jha+14]. Integrated to resource management, KPIs present a varied number of interesting and important problems that are being studied in this thesis and elsewhere as well [Kat+11].

As it can be seen, efficient and improved resource management in a datacenter can happen at different layers: application, runtime, and lower-level systems. Public and private datacenters are growing in size, and even though the dynamic and "virtual" nature of cloud computing infrastructures in its inception is more focused towards dynamic applications, recent providers are also supporting HPC applications, therefore illustrating an infrastructural convergence to how such jobs are deployed [Asc+18]. Irrespectively of a strategy steering datacenter efficiency, substantial resource allocation control entails performance degradation of running applications. However, providing a high and expected performance to applications is one of the most important goals in datacenter operations.

Therefore, one of the main focus of this thesis is to understand and control the performance tradeoffs in datacenter servers. To this end, performance trade-off strategies are methods that may degrade application performance in exchange of higher efficiency in another end, for instance to the overall number of running applications with the same amount of datacenter resources.

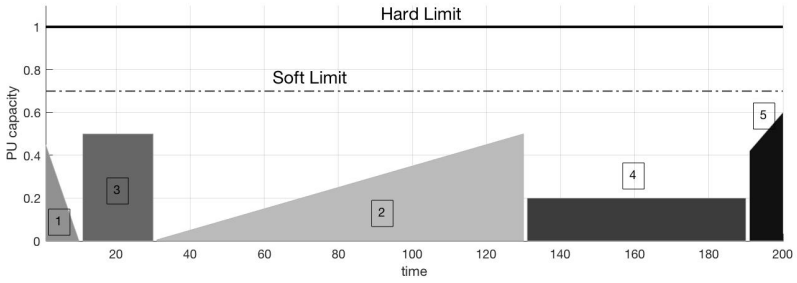
Chapter 4

Autonomous Systems for Resource Management

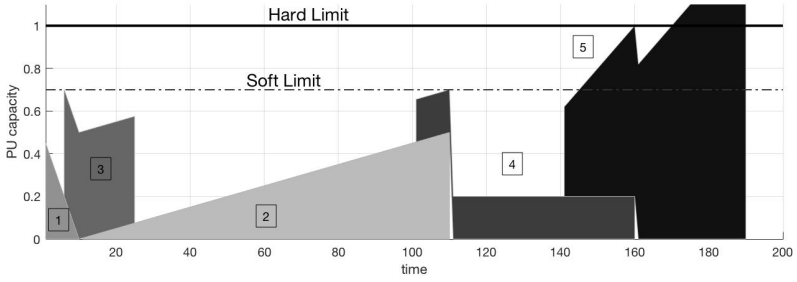
In this chapter we explain the high complexities involved in resource management and how they can be systematically studied to ease the development and evaluation of policies that support and enable the deployment of new and current types of applications. The rapid growth and wide spread use of dynamic applications demands important developments in the intelligence and heuristics coded in the software managing these datacenters. With great amounts of operational data available, the use of statistical techniques such as Machine Learning (ML) and Reinforcement Learning (RL) help to achieve insights on how to improve datacenter operations and efficiency. In fact, the combination of large amounts of data together with statistical learning techniques and feedback-loop to manage datacenters form the basis of autonomic systems for resource management [KC03].

4.1 Efficiency in Resource Management

There are multiple ways to improve datacenter efficiency in datacenters. For instance, assessing resources and their utilization rates often reveal resources that are performing single, not frequent, or small tasks, which suggest there is space for consolidation. Figure 4.1 illustrates a case where consolidation can be applied. Figure 4.1(a) shows space-sharing jobs do not fully utilize the PU capacity. By consolidating at earlier times (Figure 4.1(b), we see increases in the PU utilization. Jobs might end up competing for PU capacity at times, and monitoring this behavior may be useful to adjust the soft limits imposed to less prioritized collocated jobs. Consolidation potentially reduces total number of resources by collocating more applications on fewer machines, resulting in less spare resources, energy, and operational costs.



(a) Isolated allocations



(b) Consolidated allocations

Figure 4.1: Resource utilization assessment with two consolidation strategies. In (a) isolated (space-sharing) allocations, PU capacity has no interference. However, when consolidating (b) and due to time-sharing, some jobs (3, 4, and 5) face performance interference, and may need larger PU capacity and longer to finish execution.

In addition to supporting high-level policies and efficient execution, RMs also offer applications with communication interfaces for handling states, workload, and resource transiency variations [Sch+13; Dee+18]. Due to high dynamicity in a datacenter environment, and applications using such features, RMs have to be able to quickly adapt to workload variations at which current and new applications operate [Bur+16]. For instance, if a legacy application is latency sensitive and reliability is important, then simultaneously running two replicas of the same, synchronized application may be important as well. However, to keep these replicas consistent, the state synchronization of the two replicas may only happen when they do not seem to be producing similar outputs given similar inputs. Since latency correlates to applications workloads, the fault-tolerance synchronization mechanism used can be controlled over time to adapt the runtime system to such variations. This allows the runtime system to efficiently use available resources according to application workloads, improving overall use of datacenter resources. Likewise, if a scientific workflow describes the amount of resources needed, a management system can interact with the RM, which can then schedule appropriate number of resources to each workflow stage when needed, dynamically at runtime.

4.2 Analytics for Resource Management

Although users understand fundamental resource job requirements such as amounts of CPUs and memory, internal infrastructural utilization data and system dynamics are often visible only to cluster operators. When combined with utilization traces, operation logs, and reliability, observations can be analyzed jointly to discover patterns not easily derived without using autonomic techniques [Ste+20]. However, due to increased complexity and configurations, heuristically tweaking a RM is still today a very challenge task.

Generally, datacenter infrastructures utilize the latest performance optimized equipment. However, data often show jobs do not fully utilize allocated resource capacities [CB16; Eti+10]. In addition, a very common observation in HPC centres is that users tend to make poor estimates about parameters like total execution time and total number of resources needed [FR96]. Log traces can also be used to understand the state of a cluster as a whole and give better informed decisions for resource management or usage reporting [Sit18]. Once derived, these relationships can be used to increase cluster resource utilization by allocating spare resources to additional jobs. Predicting system utilization of parallel jobs have been studied extensively [NKG10; Mar+11; DK14; Yan+13], but adding certainty (or confidence intervals) to these predictions have not been prevalent. One main focus of a RM design is to enable decisions with confidence and to reduce false positives when detecting performance interference (while sharing resources), which is essential in HPC infrastructures. Autonomic techniques such as control theory and reinforcement learning may indicate a way to tackle such problems.

Machine Learning

Machine learning (ML) is an application of artificial intelligence (AI) enabling systems to automatically learn and improve from observed data without being explicitly programmed. ML systems focuses on the development of intelligent software that can access information extracted from data and use it to predict future behaviors of a managed system.

Reiforcement Learning

Reinforcement learning is defined as a problem with states, actions, and rewards, with state transitions that are affected by the current measured state, chosen actions, and environment's rewards. These are embedded in RL's definition, which is formulated as a Markov Decision Process [Mon82]. Generally there are no stateless variants, but some related stateless problems such as bandit optimization. Solutions for bandit optimizations allow for learning of reward based on actions, and can be optimised by selecting the best action. Main optimization problems are knowing the best action, and what total reward can be accumulated whilst still testing for which is the best action. Another common example might be advert selection online for anonymous visitors to a

website. Although there is often plenty of data available, a practical approach is to treat the probability of a click through as only depending on the choice of content, which is then the site's action.

In resource management terms, we basically have a set of resources onto which jobs needs to be assigned according to an object function. This function can be any anything that can be optimized, such as an strategy to minimize the expected queue waiting time for a given job geometry (i.e. number of resources and total runtime). The scheduler's (learner's) task is to model the behavior of each available resource and/or application by interacting (exploring) with a system and observing its reactions.

Control Theory

Highly dynamic environments and unpredictable workload variations demand new techniques to adapt resource capacity in ways to applications needs, without under or over utilizing them [Fil+15]. Runtime adaptation mechanisms are thus required to deploy robust software that operates correctly despite a lack of design-time knowledge. Control theory was initially designed to deal with the handling of continuously operating dynamical systems in engineered processes and machines. The objective is to design a model for controlling such systems using a control action in an optimum manner without delay or overshoot and ensuring control stability after a reasonable amount of time. With feedback, a system's signal outputs are used as inputs in part of a chain of cause-and-effect that forms a circuit or loop. One of these, a proportional–integral–derivative controller (PID controller) is a feedback mechanism technique widely used for cloud control systems. A PID controller continuously calculates an error value *e(t)* as the difference between a desired *setpoint* and a measured process variable and applies a correction based on proportional, integral, and derivative terms.

4.3 Challenges

Due to large demand, datacenters have been facing unprecedented challenges in its management. It is not only a problem about hardware capacity, but mainly at allowing applications to harness the infrastructure in the most optimal way.

Reliability

In clouds and HPC, datacenter system reliability directly relates to KPIs because applications and services may not run properly when failures and other anomalous behaviors occur. If a system does not support any fault tolerant mechanism, applications may need to be restarted from scratch, possibly affecting user web requests or scientific experiments results. Even if a system does support fault tolerant mechanisms, there is still a chance that application performance degrades, mainly due to re-computing times, and also the time taken to restart the failed system and ensure that the failed application is

consistent again. Therefore, it is important to accurately estimate the reliability of a datacenter system in order to better mitigate faults and thus effectively utilize its resources to improve application KPIs and achieve the SLAs agreed with cloud users.

Data Intensive Applications

There are many ways performance can be evaluated and measured, specially when comparing different infrastructures with different goals and policies. In order to evaluate a RM, one needs to understand its design goals and history, how it communicates with its users, how its resources are managed, and what tradeoffs were made to achieve its goals. Differently from most OSs that are designed to run on a certain category of processor, or to be used by a specific group of users, datacenter RMs evolved over time to operate on multiple systems and support different goals. New classes of adaptive DI jobs [HTT+09] and the convergence of many different workloads, jobs, and infrastructure management systems, combined with needs for fast and efficient resource assignment, are key challenges for modern datacenters [Asc+18]. This is particularly true for clusters with heterogeneous resources or with applications that have varying SLOs and diverse workloads with unknown variations.

However, today's HPC platforms are primarily designed to support monolithic MPI applications and to provide a static allocation model i.e., the resource allocation is fixed for the duration of the entire job [Jha+14; Reu+18; Sch+13; Com+16]. Current methods result in loss of efficiency, and the problems are likely to be exacerbated with next-generation dynamic workflows. Thus dynamic resource management models that allow workflows to dynamically increase and decrease the amount of resources used at runtime is key not only to current workflows, but also future streaming workflows.

Probabilistic Scheduling

Traditional RMs have been unable to properly manage highly dynamic and adaptive jobs because most of them are expected to execute immediately (low latency scheduling) as they often have shorter duration in comparison to more traditional batch jobs that runs for longer. Dynamic jobs are also complex to manage as they more easily adapt to different resource geometries, such as those caused by resource revocations and faults. Monolithic schedulers traditionally used by HPC clusters were designed to centrally maintain the complete state of the jobs and cluster infrastructure, while also performing workload placement logic. These design choices limits scalability and make it hard to introduce the new features and capabilities today's workloads need, and furthermore results in poor resource utilization (capacity-wise). On the other hand, characteristics from dynamic, non-monolithic schedulers (e.g. Mesos, or Kuberentes) such as resource state management, data locality and low-latency scheduling, as well as easier extensibility, are yet to be fully supported in HPC centres [Asc+18].

To illustrate with a simple arithmetic example, the difference in result between $(1/3.0) * 3 = 1.0$ (estimate, faster) and $0.3333 * 3 = 0.9999$ (accurate, slower) seems small and may not impact the progress of two applications, though each fraction is calculated differently and does affect their compute time and energy expenditure. Using similar approaches allow a diverse and intelligent exploitation of the space between the accuracy required by users (1.0 or 0.9999?) and the compute power available in datacenters. Such techniques rely on applications allowing selective approximations to be used during various computing phases, while still behaving correctly and generating consistent results. However, are not well integrated into modern datacenters, and

Data Analytics for Resource Management

Use cases for applying data analytics into RM come from best effort jobs that can use spare resources to progress computations. Given their needs of low-latency scheduling and other characteristics as fault tolerance support. The growth in the size of data sets and complexity of tasks has caused DI jobs to evolve to a point that their resource requirements are similar to traditional HPC jobs [Asc+18; Man15]. However, DI applications have different performance goals compared to traditional HPC applications. SLOs guaranteed by HPC RMs are different from what cloud providers guarantee. Whereas cloud providers are concerned with offering high levels of availability, HPC centres want applications to have minimal to non-existent interference among different users and jobs. Furthermore, there are types of resource requests typical in HPC that cannot be supported by cloud providers under their existing SLOs. For instance, it is unfeasible to allocate whole partitions of resources inside a cloud datacenter for exclusive use, which is a critical demand for running large scale scientific experiments. Similarly, it is very challenging to efficiently run HPC jobs in cloud infrastructures where RMs multiplex datacenter resources unwarily of what type of jobs that are sharing the resources.

These contrasts create opportunities that if properly explored will create means for the infrastructural convergence needed for advancing RMs in HPC and cloud computing [Asc+18; Jha+14]. First, utilizing existing HPC infrastructures shared by many users by the dynamic DI and elastic jobs is a great opportunity. Second, a solution should not interfere with already deployed infrastructures, nor it should alter the job submission workflow of current HPC infrastructures. Third, in HPC, scalability and predictability of resources are as important as utilization because they simplify application performance portability and debugging, respectively.

However, scalability and predictability can at times be contradictory to utilization. In the one hand, increases in utilization can cause severe application performance interference. On the other hand, if scalability in an application is not linear and performance does not improve linearly with additional resources, these extra resources will be underutilized [PZK16; Eti+10]. These three should

be considered when proposing new approaches for resource management because they relate to the stability and efficiency of a system.

As such, in order to allow such diverse SLOs in a same cluster to co-exist, two of the main components in a RM (see Figure 2.3) needs to be addressed: resource scheduling and performance control [Com+16]. Although there is a proliferation of new libraries, tools and scripts workarounds [Ber17], all targeting dynamicity in HPC, few efforts integrate such ideas and mechanisms directly into the RM [Dee+18]. Furthermore, any combination of static and dynamic RMs must be simple and scalable, and must detect and gracefully deal with job interference. Integrating and bridging the different constraints and requirements within the High Performance and Data Analysis (HPDA) and HPC communities is one of the aims of this thesis.

Chapter 5

Summary of Contributions

In this thesis, we present various tools and mechanisms that directly or indirectly improve resource utilization from either the user or the system perspective. For users, applications can spend less resources overall by using information extracted from, for instance, scientific workflow descriptors. In this way, a larger job can be composed of many multiple intermediary jobs of various sizes following resource requirements (geometry) for each specific task to be performed. From a system perspective, where the same amount of work (workload) is submitted to the job queue, utilization is improved if the workload is completed earlier with negligible delays in job runtimes.

The papers in this thesis are ordered following a top-down approach, that also follows the order of the specific research objectives in Section 1.1. In Paper I [Fox+17], we design a library that communicates with the resource manager and allow users to specify the resource requirements following the specific workflow stages. In Paper II [Sou+18], we focused on extending a hypervisor system (the software emulator that performs hardware virtualization in datacenters) to dynamically adjust the fault-tolerant mechanism to use according to the workload faced by the application. In Paper III [Sou+19], we investigate the monitoring of processor counters to enable finer grained resource allocation on HPC infrastructures via a two-level scheduling architectural approach. Finally, in Papers IV [Sou+20] and V [SPT20] we design two reinforcement learning algorithms to enable autonomic schedule of applications, resulting in extensive resource utilization improvements.

5.1 Paper I

W. Fox, D. Ghoshal, **A. Souza**, G. P. Rodrigo, and L. Ramakrishnan. E-HPC: A Library for Elastic Resource Management in HPC Environments. *Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science (WORKS, 2017)*, ACM, pp. 1-11, 2017.

Paper Contributions

Next-generation data-intensive scientific workflows need to support streaming and real-time applications with dynamic resource needs on HPC platforms. The static resource allocation model of current HPC systems that was designed for monolithic MPI applications is insufficient to support the elastic resource needs of current and future workflows. In this paper, we discuss the design, implementation, and evaluation of Elastic-HPC (E-HPC), an elastic framework for managing resources for scientific workflows on current HPC systems. E-HPC considers a resource slot for a workflow as an elastic window that might map to different physical resources over the duration of a workflow. Our framework uses checkpoint-restart as the underlying mechanism to migrate workflow execution across the dynamic window of resources. E-HPC provides the foundation necessary to enable dynamic resource allocation of HPC resources that are needed for streaming and real-time workflows. E-HPC has negligible overhead beyond the cost of checkpointing, and can minimize turnaround time of workflows core-hour utilization for common workflow resource use patterns. It thus provides an effective framework for elastic expansion of resources for applications with dynamic resource needs.

Authors Contributions

In this paper, I entered on an ongoing project in connection with exchange studies at the Lawrence Berkeley National Lab. (LBNL). I have implemented parts of the library enabling its use with any job scheduler and multiple queue submission, as well as designed and performed all experiments (with the in-depth implementation work). I have also written the parts of the paper concerning the experiments evaluation and discussion, and introductory schematics. The first two authors are data scientist respective postdoc who worked on the project before I came, Gonzalo P Rodrigo was a postdoc at LBNL and contributed to some parts related to his previous research at Umeå University, and Lavanya Ramakrishnan leads the group at LNBL and acted as supervisor.

5.2 Paper II

A. Souza, A. V. Papadopoulos, L. Tomás, D. Gilbert, and J. Tordsson. Hybrid Adaptive Checkpointing for Virtual Machine Fault Tolerance. *Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E, 2018)*, pp. 12-22, 2018.

Paper Contributions

Active VM replication is an application independent and cost-efficient mechanism for high availability and fault tolerance, with several recently proposed implementations based on checkpointing. However, these methods may suffer from large impacts on application latency, excessive resource usage overhead, and/or unpredictable behavior for varying workloads. To address these problems, in Paper II we propose a hybrid approach through a Proportional-Integral (PI) controller to dynamically switch between periodic and on-demand checkpointing. The mechanism proposed automatically selects the method that minimizes application downtime by adapting itself to changes in workload characteristics. The implementation is based on modifications to QEMU, LibVirt, and OpenStack, to seamlessly provide fault tolerant VM provisioning and to enable the controller to dynamically select the best checkpointing mode. Our evaluation is based on experiments with a video streaming application, an e-commerce benchmark, and a software development tool. The experiments demonstrate that our adaptive hybrid approach improves both application availability and resource usage compared to static selection of a checkpointing method, with improved application performance of up to 10% and neglectable overheads.

Authors Contributions

This project is a result of work in the ORBIT project [ORB], from which the idea also came. The first author (Abel Souza) did the bulk of the implementation (especially in regards to integration), all experiments and wrote all the text in the article. The second author (Alessandro Papadopoulos, Mälarden University) helped designing the software controller. The third author (Luis Tomás, RedHat Inc) was the project leader and helped with some technical issues. The fourth author (David Gilbert, RedHat Inc) did the COLO implementation in the Linux kernel and gave many advises about the experiments. And the fifth author (Johan Tordsson, Umeå University - supervisor) gave feedback on experiments, presentation of data, and the article at large. This article became "best paper runner up" at IEEE International Conference on Cloud Engineering 2018.

5.3 Paper III

A. Souza, M. Rezaei, E. Laure, and J. Tordsson. Hybrid Resource Management for HPC and Data Intensive Workloads. *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2019), pp. 399-409, 2019*

Paper Contributions

Traditionally, High Performance Computing (HPC) and Data Intensive (DI) workloads have been executed on separate hardware using different tools for resource and application management. With increasing convergence of these paradigms, where modern applications are composed of both types of jobs in complex workflows, this separation becomes a growing overhead and the need for a common computation platform for both application areas increases. Executing both application classes on the same hardware not only enables hybrid workflows, but can also increase the usage efficiency of the system, as often not all available hardware is fully utilized by an application. While HPC systems are typically managed in a coarse grained fashion, allocating a fixed set of resources exclusively to an application, DI systems employ a finer grained regime, enabling dynamic resource allocation and control based on application needs. On the path to full convergence, a useful and less intrusive step is a hybrid resource management system that allows the execution of DI applications on top of standard HPC scheduling systems.

In this paper we present the architecture of a hybrid system enabling dual-level scheduling for DI jobs in HPC infrastructures. Our system takes advantage of real-time resource utilization monitoring to efficiently co-schedule HPC and DI applications. The architecture is easily adaptable and extensible to current and new types of distributed workloads, allowing efficient combination of hybrid workloads on HPC resources with increased job throughput and higher overall resource utilization. The architecture is implemented based on the Slurm and Mesos resource managers for HPC and DI jobs. Our experimental evaluation in a real cluster based on a set of representative HPC and DI applications demonstrate that our hybrid architecture improves resource utilization by 20%, with 12% decrease on queue makespan while still meeting all deadlines for HPC jobs.

Authors Contributions

I and the second author (Mohammad Rezaei, KTH Royal Institute of Technology) defined the problem together. First author did the bulk of the implementation (all integration with scheduler and subsystems, etc., and Mohammad contributed with some code to the analytics bits). Abel conducted the experiments and wrote the article. Erwin Laure (KTH Royal Institute of Technology) contributed with feedback and advises on HPC perspectives as well as related

work and the text overall. Johan Tordsson gave feedback on the article writing as a whole as well as the experiments.

5.4 Paper IV

A. Souza, K. Pelckmans, D. Ghoshal,L. Ramakrishnan, and J. Tordsson. ASA – The Adaptive Scheduling Architecture. *This report is an extended version of a paper under the same title to appear at the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC, 2020).*

Paper Contributions

In HPC infrastructures, resources are controlled by batch systems and may not be readily available, which can negatively impact applications with deadlines and long queue waiting times. In particular, this is noticeable for data intensive and low latency workflows where resource planning and timely allocation are key characteristics for efficient processing. On the one hand, allocating the maximum capacity expected for a scientific workflow guarantees the fastest possible execution time, at the cost of spare and idle infrastructural resources, as well as extended queue waiting times and costly resource usage. On the other hand, dynamically allocating resources according to specific workflow stage requirements optimizes resource usage, although it may also negatively impact the total workflow makespan. With the aim of enabling new scheduling strategies and features for scientific workflows, we propose ASA: the Adaptive Scheduling Architecture, a novel and convergence proven scheduling method to reduce perceived queue waiting times as well as to optimize resource usage and planning in scientific workflows. The algorithm uses reinforcement learning to estimate queue waiting times, and based on these estimates pro-actively submits resource change requests, with the goal of minimizing total workflow inter-stage waiting times, idle resources, and makespan. The algorithm takes into consideration both learning (the waiting times), and acts on what is learnt thus far, and thus handles the exploration-exploitation trade-off. Experiments with real scientific workflows in two real supercomputers show that ASA combines the best of the two aforementioned approaches for resource allocation, with average workflows' queue waiting time and makespan reductions of up to 10% and 2% respectively, with up to 100% prediction accuracy, while obtaining near optimal resource utilization.

Authors Contributions

In this paper I defined the problem, and created the main algorithm together the second author (Kristiaan Pelckmans, Uppsala University). I did the whole of the implementation – integration with scheduler and subsystems, etc. –, besides designing and conducting all experiments, and article writing. Kristiaan Pelckmans proofed the algorithm convergence. Lavanya Ramakrishan and Devarshi Ghoshal helped at reviewing, scoping, and suggesting some related work. Johan Tordsson followed all the paper development and gave feedback on the article writing as a whole as well as the experiments.

5.5 Paper V

A. Souza, K. Pelckmans, and J. Tordsson. A HPC Co-Scheduler with Reinforcement Learning. *Submitted, 2020.*

Paper Contributions

HPC datacenters process thousands of diverse applications, supporting many scientific and business endeavours. Although users understand fundamental resource job requirements such as amounts of CPUs and memory, internal infrastructural utilization data and system dynamics are often visible only to cluster operators. Besides that, due to increased complexity, heuristically tweaking a batch system is even today a very challenge task. When combined with applications profiling, infrastructure data enables improvements to job scheduling, and also better support for QoS metrics such as queue waiting time and total execution time. Targeting improvements in utilization and throughput, in this paper we evaluate and propose a novel reinforcement learning co-scheduling algorithm that combines capacity utilization with application performance profiling. We first profile a running application by assessing its resource utilization and progress by means of a forest of decision trees, enabling our algorithm to understand the application's resource capacity usage. We then use this information to estimate how much capacity from the current allocation can be used for co-scheduling additional applications. Our algorithm learns from incorrect estimations and evaluates when co-scheduling decisions results in QoS degradation, such as application slowdown. For our implementation, our co-scheduling architecture use a handful metrics to help minimizing performance degradation, enabling improvements on utilization of up to 25% even when the cluster is experiencing high demands, with 10% average queue makespan reductions when experiencing low loads.

Authors Contributions

In this paper I defined the problem, and created the main algorithm together the second author (Kristiaan Pelckmans, Uppsala University). I did the whole of the implementation – integration with scheduler and subsystems, etc. –, besides designing and conducting all experiments, and article writing. Kristiaan Pelckmans proofed the algorithm convergence. Johan Tordsson followed all the paper development and gave feedback on the article writing as a whole as well as the experiments.

5.6 Limitations

Relentless demand for greater computing capabilities makes cost efficiency the main metric of interest in the design of datacenters. Thus, datacenter operators' primary objective concerns minimizing operational costs by maximizing utilization while simultaneously minimizing applications performance degradation due to resource sharing.

As noted in the summary of papers, performance is not measured as a single quantity because it can be obtained in different ways. One metric is *overhead*, the extra resource cost of implementing an abstraction presented to, or used by applications. A related metric is efficiency, the low overheads cause by an abstraction. RMs need to allocate resources among applications, and this affects the system performance as perceived by the end user. Focusing on a small number of metrics may bring fairness issues among multiple applications running on the same cluster or machine. Questions regarding equal divisions or prioritized access to resources among different users and applications are often raised.

Predictability, a related performance metric is whether the system's response time (or other metric) is consistent over time. Predictability is most of time more important than average performance, because it accurately estimates confidence intervals for how long repeated application experiments take in a system. As job constraints are compounded, the predictability of some our methods may be hurt, although we tackle some of these on Paper IV and V. Other areas for improvement relate to doing a deeper mathematical analysis of the consequences of using one method over another.

5.7 Future Work

Research in resource management for large scale clusters has always had a duality between increasing resource utilization, and guaranteeing predictability of allocated resources. For HPC workloads, the focus is on the later, though recent challenges on converged infrastructures are demanding new solutions because in Cloud datacenters, the focus in on utilization with efficiency. To mix job classes with different SLOs is a main challenge in such infrastructures, and any solution has to provide mechanisms for intra-node isolation of collocated jobs, outlier and interference detection, and a mechanism to handle interference, which could be specified as a set of general rules and datacenter specific.

In these directions, Paper III targets dynamic execution in HPC clusters, and as mentioned, one of resource management's main goals is to have predictable resource assignment of jobs. Thus our goal is to extend on Paper IV and V approaches by considering ways to minimize performance interference and/or false positives in our co-locations while making use of job prioritization as it happens in traditional HPC environments. One can also combine the two algorithms proposed in both papers and create a new level of probabilistic

scheduling with reinforcement learning. In doing so, the architecture unifying the last three papers would be able to support an unrestricted range of workflows. Resources would now be viewed as statistical entities, where its capacities would be guaranteed to be within an acceptable range. Applications could then extend on this architecture to schedule its tasks transparently with no changes in its workflow, and the side effect would be datacenter higher throughput and utilization.

Finally, the varied and new combined ways for achieving extreme-efficiency at scale is influencing all layers of a datacenter system stack [Jha+14]. One way towards this direction regards the use of compute specialization, where specialized hardware is used to compute specific types of operations. This has been replacing the computer industry economies of scale dependency on Moore's law [TW17], with new computing paradigms being proposed. For instance, recent developments such as *approximate computing* leverages on the idea where applications' progress depend on data estimates, and not on exact data inputs/outputs. Using such approaches allow a more diverse and intelligent exploitation of the space between the accuracy required by users and the compute power available in datacenters. As this could be done in different ways, we can also extend on the combined ideas of Paper V for improving the efficiency of future datacenter realizations by integrating and supporting approximate applications together with resource management. Potential use cases for this can impact scientific and industrial real-time applications, such as the ones found in aviation and autonomous vehicles. A possible outcome would be aimed at future datacenter realizations, which would be able to support new power performance tradeoffs by using autonomic tools and methods to enable approximate applications to achieve the high efficiency and performance needed in such infrastructures.

Bibliography

[Ahn+14] Dong H Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. "Flux: a next-generation resource management framework for large HPC centers". In: *2014 43rd International Conference on Parallel Processing Workshops*. IEEE. 2014, pp. 9–17.

[Amv+18] George Amvrosiadis, Jun Woo Park, Gregory R Ganger, Garth A Gibson, Elisabeth Baseman, and Nathan DeBardeleben. "On the diversity of cluster workloads and its impact on research results". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association. 2018, pp. 533–546.

[And+18] Georgios Andreadis, Laurens Versluis, Fabian Mastenbroek, and Alexandru Iosup. "A reference architecture for datacenter scheduling: design, validation, and experiments". In: *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2018, pp. 478–492.

[Asc+18] M Asch, T Moore, R Badia, M Beck, P Beckman, T Bidot, F Bodin, F Cappello, A Choudhary, B de Supinski, et al. "Big data and extreme-scale computing: Pathways to Convergence-Toward a shaping strategy for a future software and data ecosystem for scientific inquiry". In: *The International Journal of High Performance Computing Applications* 32.4 (2018), pp. 435–479.

[BCH13] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines". In: *Synthesis lectures on computer architecture* 8.3 (2013), pp. 1–154.

[Ber+04] GB Berriman, JC Good, AC Laity, A Bergou, J Jacob, DS Katz, E Deelman, C Kesselman, G Singh, M-H Su, et al. "Montage: A grid enabled image mosaic service for the national virtual observatory". In: *Astronomical Data Analysis Software and Systems (ADASS) XIII*. Vol. 314. 2004, p. 593. URL: `http://montage.ipac.caltech.edu/`.

[Ber17] Evan Berkowitz. "METAQ: Bundle Supercomputing Tasks". In: (2017). arXiv: 1702.06122 [physics.comp-ph]. URL: https://github.com/evanberkowitz/metaq.

[Bur+16] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. "Borg, omega, and kubernetes". In: *Queue* 14.1 (2016), pp. 70–93.

[CB16] Marc Casas and Greg Bronevetsky. "Evaluation of HPC applications' memory resource consumption via active measurement". In: *IEEE Transactions on Parallel and Distributed Systems* 27.9 (2016), pp. 2560–2573.

[Com+16] Isaıas Comprés, Ao Mo-Hellenbrand, Michael Gerndt, and Hans-Joachim Bungartz. "Infrastructure and api extensions for elastic execution of mpi applications". In: *Proceedings of the 23rd European MPI Users' Group Meeting*. ACM. 2016, pp. 82–97.

[CP15] Gary Cook and David Pomerantz. "Clicking clean: A guide to building the green Internet". In: *Greenpeace International, Tech. Rep.* (2015).

[Dee+18] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. "The future of scientific workflows". In: *The International Journal of High Performance Computing Applications* 32.1 (2018), pp. 159–175.

[DG08] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[DK13] Christina Delimitrou and Christos Kozyrakis. "Paragon: QoS-aware scheduling for heterogeneous datacenters". In: *ACM SIGPLAN Notices*. Vol. 48. 4. ACM. 2013, pp. 77–88.

[DK14] Christina Delimitrou and Christos Kozyrakis. "Quasar: resource-efficient and QoS-aware cluster management". In: *ACM SIGPLAN Notices* 49.4 (2014), pp. 127–144.

[Ede07] Amnon H Eden. "Three paradigms of computer science". In: *Minds and machines* 17.2 (2007), pp. 135–167.

[Eti+10] Maja Etinski, Julita Corbalan, Jesus Labarta, and Mateo Valero. "Utilization driven power-aware parallel job scheduling". In: *Computer Science-Research and Development* 25.3-4 (2010), pp. 207–216.

[Fil+15] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, et al. "Software engineering meets control theory". In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press. 2015, pp. 71–82.

[Fos+08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. "Cloud computing and grid computing 360-degree compared". In: *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee. 2008, pp. 1–10.

[Fox+17] William Fox, Devarshi Ghoshal, Abel Souza, Gonzalo P. Rodrigo, and Lavanya Ramakrishnan. "E-HPC: A Library for Elastic Resource Management in HPC Environments". In: *Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science*. WORKS '17. 2017.

[FR96] Dror G Feitelson and Larry Rudolph. "Toward convergence in job schedulers for parallel supercomputers". In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 1996, pp. 1–26.

[Gho+11] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types." In: 11.2011 (2011), pp. 323–336.

[Hin+09] Benjamin Hindman, Andrew Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Scott Shenker, and Ion Stoica. "Nexus: A common substrate for cluster computing". In: *Workshop on Hot Topics in Cloud Computing*. 2009.

[Hin+11] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." In: 11.2011 (2011), pp. 295–308.

[HTT+09] Tony Hey, Stewart Tansley, Kristin M Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*. Vol. 1. Microsoft research Redmond, WA, 2009.

[Inc16] Google Inc. *Google Data Centers, Efficiency: How we do it*. 2016. URL: `https://www.google.com/about/datacenters/efficiency/internal/` (visited on Jan. 2, 2017).

[Jha+14] Somesh Jha, Jian Qiu, Andre Luckow, Pradeep Mantha, and Geoffrey C Fox. "A tale of two data-intensive paradigms: Applications, abstractions, and architectures". In: *IEEE BigData Congress, 2014*. 2014, pp. 645–652.

[Jon18] Nicola Jones. "How to stop data centres from gobbling up the world's electricity". In: *Nature* 561.7722 (2018), pp. 163–167.

[JS15] Brendan Jennings and Rolf Stadler. "Resource management in clouds: Survey and research challenges". In: *Journal of Network and Systems Management* 23.3 (2015), pp. 567–619.

[Kat+11] Gregory Katsaros, Georgina Gallizo, Roland Kübert, Tinghe Wang, Josep Oriol Fitó, and Daniel Henriksson. "A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments." In: *CLOSER*. 2011, pp. 232–239.

[KC03] Jeffrey O Kephart and David M Chess. "The vision of autonomic computing". In: *Computer* 36.1 (2003), pp. 41–50.

[KTC01] Wendy Korn, Patricia J Teller, and G Castillo. "Just how accurate are performance counters?" In: *2001. IEEE International Conference on Performance, Computing, and Communications*. IEEE. 2001, pp. 303–310.

[Man15] Zoltán Ádám Mann. "Allocation of Virtual Machines in Cloud Data Centers – A Survey of Problem Models and Optimization Algorithms". In: *ACM Computing Surveys (CSUR)* 48.1 (2015), p. 11.

[Mar+11] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations". In: (2011), pp. 248–259.

[Men07] Paul B Menage. "Adding generic process containers to the linux kernel". In: *Proceedings of the Linux Symposium*. Vol. 2. Citeseer. 2007, pp. 45–57.

[Mer14] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (2014), p. 2.

[Mit16] Sparsh Mittal. "A survey of techniques for approximate computing". In: *ACM Computing Surveys (CSUR)* 48.4 (2016), pp. 1–33.

[Mon82] George E Monahan. "State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms". In: *Management science* 28.1 (1982), pp. 1–16.

[NKG10] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. "Q-clouds: managing performance interference effects for QoS-aware clouds". In: *Proceedings of the 5th European conference on Computer systems* (2010), pp. 237–250.

[ORB] ORBIT. *Business Continuity as a Service (ORBIT)*. URL: `http://www.orbitproject.eu`.

[Pab09] Chandandeep Singh Pabla. "Completely fair scheduler". In: *Linux Journal* 2009.184 (2009), p. 4.

[Pra+15] Suraj Prabhakaran, Marcel Neumann, Sebastian Rinke, Felix Wolf, Abhishek Gupta, and Laxmikant V Kale. "A batch system with efficient adaptive scheduling for malleable and evolving applications". In: *2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2015, pp. 429–438.

[PZK16] Natalie Perlin, Joel P Zysman, and Ben P Kirtman. "Practical scalability assesment for parallel scientific numerical applications". In: *arXiv preprint arXiv:1611.01598* (2016).

[Reu+16] Albert Reuther, Chansup Byun, William Arcand, David Bestor, Bill Bergeron, Matthew Hubbell, Michael Jones, Peter Michaleas, Andrew Prout, Antonio Rosa, et al. "Scheduler technologies in support of high performance data analysis". In: *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*. IEEE. 2016, pp. 1–6.

[Reu+18] Albert Reuther, Chansup Byun, William Arcand, David Bestor, Bill Bergeron, Matthew Hubbell, Michael Jones, Peter Michaleas, Andrew Prout, Antonio Rosa, et al. "Scalable system scheduling for HPC and big data". In: *Journal of Parallel and Distributed Computing* 111 (2018), pp. 76–92.

[Rou16] M Rouse. *Docker swarm*. 2016.

[Sch+13] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. "Omega: flexible, scalable schedulers for large compute clusters". In: *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM. 2013, pp. 351–364.

[She+16] Arman Shehabi, Sarah J. Smith, Dale A. Sartor, Richard E. Brown, Magnus Herrlin, Jonathan G. Koomey, Eric R. Masanet, Nathaniel Horner, Inês L. Azevedo, and William Lintner. *United States Data Center Energy Usage Report*. 2016. URL: `https://eta.lbl.gov/publications/united-states-data-center-energy` (visited on June 2, 2019).

[Sit18] Richard L Sites. "Benchmarking" Hello, World!"". In: *Queue* 16.5 (2018), pp. 54–80.

[Sou+18] Abel Souza, Alessandro Vittorio Papadopoulos, Luis Tomas, David Gilbert, and Johan Tordsson. "Hybrid adaptive checkpointing for virtual machine fault tolerance". In: *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2018, pp. 12–22.

[Sou+19] Abel Souza, Mohamad Rezaei, Erwin Laure, and Johan Tordsson. "Hybrid Resource Management for HPC and Data Intensive Workloads". In: *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2019, pp. 399–409.

[Sou+20] Abel Souza, Kristiaan Pelckmans, Devarshi Ghoshal, Lavanya Ramakrishnan, and Johan Tordsson. *ASA – The Adaptive Scheduling Architecture*. Research Report. Extended report. Umeå University, 2020. URL: `http://umu.diva-portal.org/smash/record.jsf?pid=diva2%3A1423086`.

[SPT20] Abel Souza, Kristiaan Pelckmans, and Johan Tordsson. *An HPC Co-Scheduler with Reinforcement Learning*. Research Report. Umeå University, 2020. URL: `http://umu.diva-portal.org/smash/record.jsf?pid=diva2%3A1423087`.

[Sri+02] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and P Sadayappan. "Characterization of backfilling strategies for parallel job scheduling". In: *International Conference on Parallel Processing Workshops*. IEEE. 2002, pp. 514–519.

[Sta06] Garrick Staples. "TORQUE resource manager". In: *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM. 2006, p. 8.

[Ste+20] Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. *AI for Science*. Tech. rep. Argonne National Lab.(ANL), Argonne, IL (United States), 2020.

[TW17] Thomas N Theis and H-S Philip Wong. "The end of moore's law: A new beginning for information technology". In: *Computing in Science & Engineering* 19.2 (2017), pp. 41–50.

[Vav+13] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Owen O Malley, Benjamin Reed, et al. "Apache Hadoop Yarn: Yet another resource negotiator". In: *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM. 2013, p. 5. ISBN: 9781450324281.

[WD96] David W Walker and Jack J Dongarra. "MPI: A standard message passing interface". In: *Supercomputer* 12 (1996), pp. 56–68.

[WSI05] WSIS. *Declaration of Principles: Building the Information Society: a Global Challenge in the New Millennium*. 2005.

[Yan+13] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. "Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers". In: *ACM SIGARCH Computer Architecture News*. Vol. 41. 3. ACM. 2013, pp. 607–618.

[YJG03] Andy B Yoo, Morris A Jette, and Mark Grondona. "Slurm: Simple linux utility for resource management". In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2003, pp. 44–60.

[Zah+11] Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi, Anthony D Joesph, Randy Katz, Scott Shenker, and Ion Stoica. "The datacenter needs an operating system". In: *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*. USENIX Association. 2011, pp. 17–17.

[Zha+13] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. "CPI 2: CPU performance isolation for shared compute clusters". In: *8th ACM European Conference on Computer Systems*. ACM. 2013, pp. 379–391.