UMEÅ UNIVERSITY

Application-Aware Resource Management for Datacenters

Abel Pinto Coelho de Souza

Licenciate Thesis, December 2018
Department of Computing Science
Umeå University
Sweden

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

http://people.cs.umu.se/abel
abel@cs.umu.se

ISBN 978-91-7601-971-9
ISSN 0348-0542
UMINF 18.14

Printed by UmU Print Service, Umeå University, 2018

Abstract

High Performance Computing (HPC) and Cloud Computing datacenters are extensively used to steer and solve complex problems in science, engineering, and business, such as calculating correlations and making predictions. Already in a single datacenter server, there are thousands of hardware and software metrics – Key Performance Indicators (KPIs) – that individually and aggregated can give insight in the performance, robustness, and efficiency of the datacenter and the provisioned applications. At the datacenter level, the number of KPIs is even higher. The fast growing interest on datacenter management from both public and industry together with the rapid expansion in scale and complexity of datacenter resources and the services being provided on them have made monitoring, profiling, controlling, and provisioning compute resources dynamically at runtime into a challenging and complex task. Commonly, correlations of application KPIs, like response time and throughput, with resource capacities show that runtime systems (e.g., containers or virtual machines) that are used to provision these applications do not utilize available resources efficiently. This reduces datacenter efficiency, which in term results in higher operational costs and longer waiting times for results.

The goal of this thesis is to develop tools and autonomic techniques for improving datacenter operations, management and utilization, while improving and/or minimizing impacts on applications performance. To this end, we make use of application resource descriptors to create a library that dynamically adjusts the amount of resources used, enabling elasticity for scientific workflows in HPC datacenters. For mission critical applications, high availability is of great concern since these services must be kept running even in the event of system failures. By modeling and correlating specific resource counters, like CPU, memory and network utilization, with the number of runtime synchronizations, we present adaptive mechanisms to dynamically select which fault tolerant mechanism to use. Likewise, for scientific applications we propose a hybrid extensible architecture for dual-level scheduling of data intensive jobs in HPC infrastructures, allowing operational simplification, on-boarding of new types of applications and achieving greater job throughput with higher overall datacenter efficiency.

Preface

This thesis contains a brief description of datacenter infrastructures, a discussion on improving performance efficiency in datacenters, 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. *ACM, Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 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. *IEEE International Conference on Cloud Engineering (IC2E)*, pp. 12-22, 2018.
Paper III	**A. Souza**, M. Rezaei, E. Laure and J. Tordsson. Hybrid Resource Management for HPC and Data Intensive Workloads. *Submitted for publication, 2018.*

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, and the Brazil National Council for Scientific and Technological Development (CNPq) under project no. 207555/2014-1.

Contents

1 Introduction **1**
1.1 Research Problem and Objectives . . . 3
1.2 Methodology . . . 4

2 Resource Management in Datacenters **5**
2.1 High Performance and Cloud Computing . . . 7
2.2 Components and Characteristics . . . 9
2.2.1 Classes of Jobs . . . 9
2.2.2 Scientific Workflows: Job Templates and Pipelines . . . 10
2.2.3 Resource Managers and Sharing . . . 12
2.2.4 Reliability and Availability . . . 13
2.3 Challenges . . . 13
2.3.1 Collocation . . . 14
2.3.2 Isolation . . . 15
2.3.3 Multi-Level Scheduling . . . 16
2.4 Data Analysis for Resource Efficiency . . . 16

3 Summary of Contributions **19**
3.1 Paper I . . . 19
3.2 Paper II . . . 20
3.3 Paper III . . . 21
3.4 Future Work . . . 22

Paper I **29**

Paper II **53**

Paper III **69**

Chapter 1

Introduction

A datacenter is the main physical infrastructure behind many local and global services, a key enabler for high performance (HPC) and cloud computing [41]. Effective datacenter operations is a crucial aspect for many organizations around the world because customers satisfaction and datacenter costs are directly linked to optimal and performant operational services, a concept also known as Information Technology (IT) performance management. Many important applications and services from both industry and academia could be mentioned: Google, Facebook and YouTube, as well as scientific applications enabling researchers to address pressing challenges in e.g., medicine, chemistry, biology (bioinformatics), materials design, natural disasters and climate change. A datacenter is composed of different hardware equipment for performing computations and data handling (Input/Output or I/O): data processing (compute servers), data storage, and communication (network). Together, this hardware equipment respectively processes, stores, and transmits digital information (the data) around the world. Datacenters have become ubiquitous and important to nearly all aspects of communication, business, academic, and governmental information systems [44].

As the engines that make industry services and most scientific applications run, datacenters processing power and reliability depend on very large infrastructures, remotely accessed through the Internet. It is very complex to operate and maintain such infrastructures in an efficient and cost-effective way, and this challenge is subject of much research, including this thesis. Demands for and on datacenters have been consistently increasing over the years and thus, commercial providers are building newer and even bigger datacenters in many regions across the world. Furthermore, datacenters generate lots of data which can be used to automatize most of resource management problems, like overall system efficiency, and software failures. Combined with statistical learning techniques, this data can lead to automatic remediation and reductions in the time to solution for such management problems, which otherwise require significant time and effort.

The software managing datacenters is called Resource Manager (RM) [47]. A RM is responsible for making important decisions regarding the datacenter's internal operations, which affect the whole datacenter's performance, functionality, and maintenance costs. Datacenter RM softwares such as auto-scaling or elasticity engines and schedulers provide partial 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 [21, 23]. Traditionally, such systems have been designed following models of application and system performance based on a selected few metrics such as response time and throughput (for applications) and utilization of servers, CPU, memory, bandwidth as well as energy expenditure and heat (for servers) [13, 40]. 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' Key Performance Indicators (KPIs) [24, 23]. In this context, KPIs are monitored values specific to the service or application in question and describe its performance. Integrated to resource management, KPIs present a varied number of interesting and important problems that are yet to be studied [24]. Here holistically means that the global view of a system and supported applications have to be systematically considered before trying to improve any of the system parts.

There are commonly two types of applications being managed: interactive and batch [35, 3, 23]. Interactive applications are commonly web requests or tasks in a data analysis pipeline, which runs in cloud datacenters. Interactive applications are commonly described because of their low latency characteristics. The second type, batch applications, are 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. Batch workloads are common in scientific computing, historically running in HPC environments to enable large scale experiments. Because these experiments are time consuming, scientists can split the overall problem in independent parts, pipelined in order to produce final data products and are called *scientific workflows* [9]. Scientific workflows allow users to customize/sweep parameters in an experiment without viewing or altering its inner code, making them vastly flexible and reuseable. Besides that, some mission critical applications residing in datacenters must achieve high availability (HA), which is the percentage of time the infrastructure or web service is in an operable state. Outages can severely damage datacenter services reputation and overall users' perception with web services, and thus most datacenter services typically aim for at least 99.99% uptime (available services, according to the Service Level Agreement), approximately an hour of downtime (unavailable services) over a year. HA is achievable by using replication, that is duplicating the execution of the system of interest. One way of implementing HA is through fault tolerance mechanisms, which commonly use checkpoint and restart techniques and/or hardware and software request duplication, where one request is executed two or more times, and in case of failures, a secondary replica takes over the request execution or failed component.

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 are the ones maintaining and providing fair means to support end-users the access to the datacenter infrastructure through a RM software system, following an organizational policy on how to access and share available resources. At datacenter's scale, the RM has the role of an OS [19], since it is the layer of software abstracting and managing the infrastructure's hardware resources for users and their applications, running in a wide range of compute servers, invisible to end users. Understanding the various tradeoffs in allocating and requesting resources, besides adapting applications runtime mechanisms and capacities according to the workload's variations lead to operational efficiency in datacenters because it tailors the infrastructure to the application dynamic demands.

1.1 Research Problem and Objectives

In this thesis, we investigate and map resources capacities to workloads with the goal of maximizing application performance and resource utilization. Thus we analyse the tradeoffs in performance, reliability, and costs in datacenters, create models that capture these tradeoffs, and propose controllers and tools to optimise datacenter infrastructures taking these tradeoffs into consideration. The main research problem investigated in this thesis is how to increase the efficiency of datacenter taking into account the applications performance tradeoffs.

Therefore, our main objective is to perform a more efficient management of the computing infrastructure by continuously adjusting the number of resources, their capacity and the their runtime mechanisms that affect resource usage and allocations in face of what the workloads demand from allocated resources. We do this by controlling the performance tradeoffs in datacenter servers through the application, runtime, and operating systems levels of software configurations. The main research objectives of this thesis are:

RO1 To enable developers to specify scientific workflow resource requirements in order to improve overall performance efficiency of applications and datacenters.

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

RO3 To use and develop statistical learning methods to enable resource capacity controllers for servers to continuously improve datacenter throughput (e.g., number of completed jobs) without sacrificing application performance.

1.2 Methodology

The methodology used in this thesis is mostly experimental. To model **RO1** and **RO2**, we set up testbeds consisting of multiple servers and/or use real clusters. To put a load on the servers we use both benchmarks generating utilization of selected computational resources and real applications modeling scientific methods/experiments and handling web-requests from users. When the servers are exposed to the load, we dynamically (at workloads' runtime) modify various configurations and measure the performance of hosted 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. Furthermore, we use statistical methods to model dependencies between application performance metrics and server metrics, like CPU and memory utilization.

Chapter 2

Resource Management in Datacenters

In this chapter we describe the role of a resource manager (RM), which is the software system that manages the infrastructure and application scheduling in a datacenter according to organizational policies that specify how such infrastructures can be used. Novel classes of applications require RMs to be more dynamic in order to allow runtime systems and orchestrators to handle, measure, and evaluate how different organizational policies affect application KPIs experienced by users. We also discuss challenges and approaches to improve the efficiency of RMs. We give an overview of concepts such as collocation, isolation, resource control and scheduling, link the approaches to enabling technologies and datacenter actuators, review the research performed in each area, and present the associated limitations and challenges.

A RM is the middleware managing infrastructure resources, such as supercomputers and cloud datacenters [21], also known as clusters. The RM has the ability to run distributed applications on a cluster (Figure 2.1). These applications (sometimes referred to as jobs) are usually encapsulated and orchestrated through virtual machines and containers (herein also called runtime systems), or classic Operating System (OS) processes [6]. From a system's perspective, in addition to handle the actual execution of jobs coming in a queue (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 hardware faults in a graceful manner. In addition, from a user perspective, the RM should improve execution times and fairness among different workloads, users, and projects [13]. The queue makespan, a metric which is defined at the total amount of time a given set of jobs takes to finish their execution, is one important metric users and operators want to minimize.

Besides handling high-level policies, like fairness and efficient execution, more recently RMs also have to provide applications with communication in-

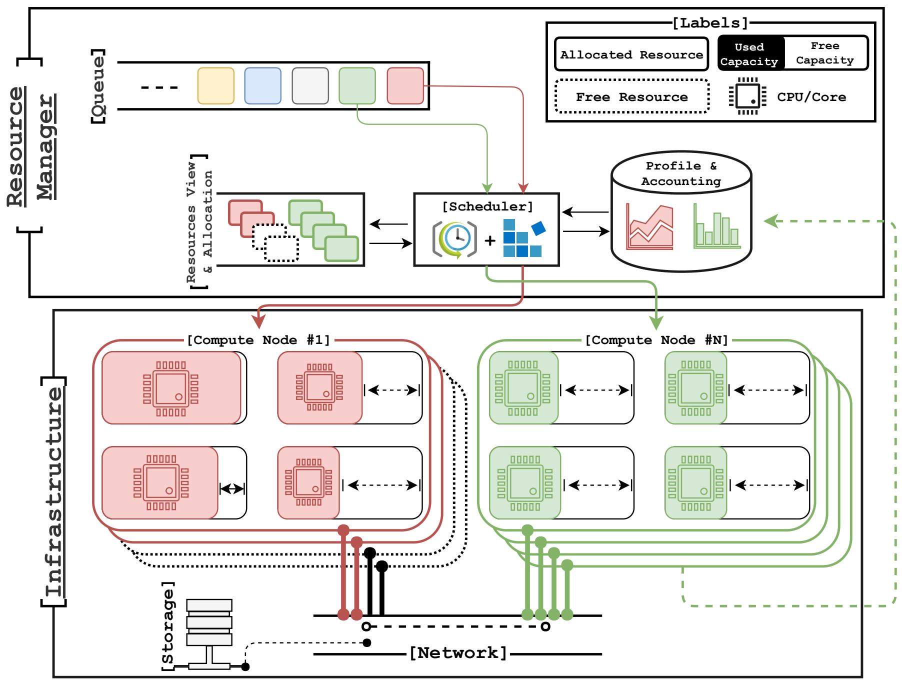

Figure 2.1: Resource Manager in a datacenter cluster: (Coloured) jobs are queued (submitted by users), and requested resources allocated in the compute nodes by the scheduler (big red and green parallel rectangles in the Infrastructure). During job execution, resources' capacity utilization and idleness can be monitored and profiled (white areas within compute nodes). The underneath network connects all nodes and remote storage.

terfaces for handling state, load, and resource capacity variations, the latter for different and complex workflows that now are emerging from academia and industry [38]. Applications and runtime systems are aware of measures like workload variation, progress, and on what type of remediation is needed in order to mitigate system faults, a feature also known as fault-tolerance support [9]. Because of this high dynamicity in the datacenter and new classes of applications using such features, RMs have to be able to quickly adapt to workload variations at which current and new applications operate [6]. For instance, if a legacy application is latency sensitive and reliability is of great concern, then it may be worth to simultaneously run two replicas of the same application and synchronize the two replicas only when they do not seem to be producing similar outputs given similar inputs. Since latency is linked to applications workloads, the fault-tolerance synchronization mechanism used (e.g. by a virtual machine) can be switched over time in order to adapt the runtime system to such variations. This allows the runtime system to more efficiently utilize available resources according to application loads, thus improving overall datacenter resource usage. Likewise, if a scientific workflow describes the amount of resources the tasks demand, a workflow management system can communicate with the RM, which can then allocate the appropriate number of resources to each of the workflow stage when needed, dynamically at runtime.

As a building block for datacenter RMs, the lowest system layers (e.g. OS) provide a new channel of management from which RMs can relate hardware metrics such as processor counters to application KPIs [6]. Processor counters are CPU hardware registers that count events such as CPU utilization, number of instructions executed, cache-misses, or branch mispredictions. At the OS level, these metrics form a basis for profiling applications to trace dynamic control flow and identify hot-spots. Based on continuous job profiling, RMs can utilize specific resource controllers to understand and adapt resource capacity according to what jobs actually need, thus improving overall resource utilization.

As it can be seen, efficient and improved resource management in a datacenter can happen at different levels of the stack: application, runtime, and lower-level systems. For different goals and objectives (i.e. policies), the way resources are shared plays a very important role. *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). These processes can be seen as sharing only if observed that resources separately run the different processes in the same infrastructure, thus the term *space-sharing*.

2.1 High Performance and Cloud Computing

High Performance Computing (HPC) organizes independent compute nodes in clusters that can deliver more performance than could be delivered from a typical personal computer (PC) or workstation. HPC clusters are used to solve and steer complex problems and experiments in science, engineering, and business, and is key for innovation [41]. The reason for having a HPC system is because independent nodes working together can (in a timely manner) solve problems bigger than an independent PC could ever solve. To work together, cluster nodes need to communicate with each other. This is usually performed through the network by means of standardized libraries such as the Message Passing Interface (MPI) [43]. The use of one type of network over another usually depend on the bottlenecks most workloads experience in an HPC centre. HPC clusters usually are managed by static RMs like Slurm [22] or Torque [40], which commonly asks users to provide a deadline for launching jobs [36]. Jobs can finish earlier than the specified deadlines, and more importantly, can use less resources than what was required by the users at job submission. Usually users cannot use more resources than what has already been allocated to them, 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 [4] 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 like job state management and monitoring are not fully integrated into HPC schedulers. A time-sharing RM such as Slurm with a space-shared policy assures predictable resource performance to jobs at the cost of higher queue-makespan [1].

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 [21]. 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 like Mesos [19], Omega [38] and Yarn [42], 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 [21]. 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 (like Slurm) supports finer grain allocations, they come with no support for enforcing resource isolation when sharing resources (see Section 2.2).

The main differences between cloud and HPC lie in the model of delivery or access to computing resources as well as associated costs [14]. In traditional HPC, the operator (say, an university) will typically owns, share, and organizes the infrastructural system management, in a way that satisfies users' workload characteristics and priorities. By owning the server, the operator incurs capital expenditures,[1] e.g. cost of servers, hiring admins, compute room rent, and etc. Besides that, there are recurrent operational expenditure, e.g. power and cooling, admin wages, software and hardware upgrades.[2] This cost is almost constant regardless of whether the server is fully utilized or not. Cloud Computing, on the other hand, is really an economical definition for delegating the management of the IT hardware infrastructure to a third party (Cloud Provider), who buys, maintains, shares and utilizes the computer infrastructure. The user has access to a virtual infrastructure through the public Internet, with a pay-per use billing scheme. Users personalize this virtual infrastructure by using various runtimes or application environments such as containers, virtual machines and orchestration tools like Mesos and Kubernetes [6].

[1]Known as *capex* cost
[2]Known as *opex* cost

2.2 Components and Characteristics

A cluster system's architecture provides four main functions to job (applications) management, as illustrated in Figure 2.2: job life-cycle, resource monitoring, scheduling, and job execution [35]. 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 hardware resources (i.e. CPU cores, memory, network bandwidth and other resources), the amount of each, and/or time constraints, like 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 required to execute the job, and assigns these resources based on datacenter policies and resource 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 KPIs, 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. An example would be allowing users to co-schedule multiple of their jobs' tasks in varied ways not currently possible by static libraries.

2.2.1 Classes of Jobs

Historically, jobs have been classified in four types: rigid, moldable, malleable, and evolving [13]. Specifically, these classes differ in what can happen to a job's resource geometry throughout its life-cycle. Whereas a moldable job can have its resource geometry changed before its execution, a rigid job cannot. These jobs are sometimes considered the same, and simply known as *rigid* jobs since none of them allow resource changes at runtime. Malleable and evolving jobs, which are often considered the same and simply called *malleable*, can change resource geometry throughout execution, the difference between the two is which entity requests the change: the *scheduler* for malleable jobs, or the *user* for evolving jobs respectively. With large data processing needs becoming a norm both in industry and in scientific communities, a fifth class has emerged: *adaptive jobs*. These jobs are characterized by being highly dynamic and adaptable to changes and system faults, and for having data intensive (DI) requirements as they process unprecedented amounts of data. In relation to resource needs, adaptive jobs are a combination of malleable and evolving jobs [34]: malleable for how many resources they require, and evolving

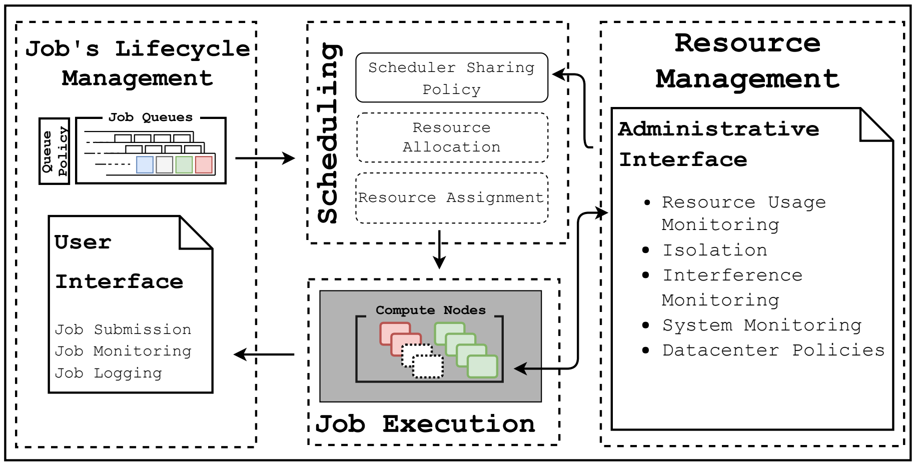

Figure 2.2: Components of a typical Resource Manager: Job Lifecycle, Scheduling and Resource Management in datacenters. Users submit jobs through an interface. Jobs are queued and scheduled for execution. Each one of these steps is profiled and monitored according to the policies set in the management component. Adapted from [35].

because the number of resources can vary at runtime according to system and workload contexts.

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 an objective is datacenter resource utilization, that is focused on measuring how well, or how efficiently, resources in a datacenter are being used. From the applications context, it is defined as how efficiently a given resource capacity is used by the application, where operators often try to maximize the use of specific resources without (negatively) impacting user application 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. Even though it is very intuitive, utilization hides many aspects of what happens internally in a system. A system is often intertwined with other system(s) which may have different utilization ratios and would then perform differently to similar requests, depending on time and on the workload at that time.

2.2.2 Scientific Workflows: Job Templates and Pipelines

Large scale experiments rely on big computing infrastructures available to scientists. Because these experiments are very large and time consuming, scientists

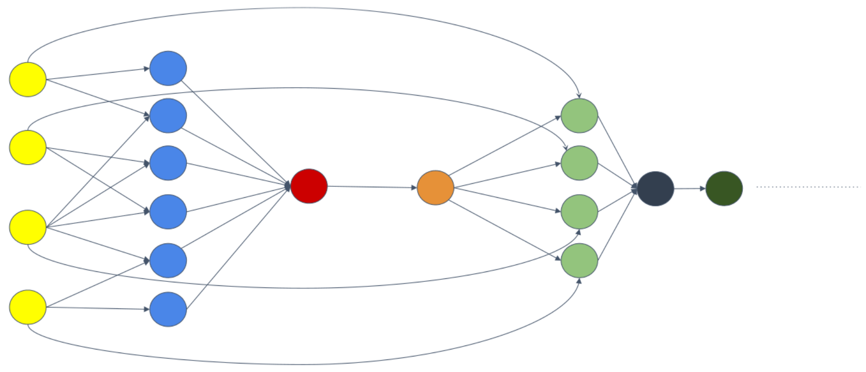

Figure 2.3: Scientific Workflow: Pipeline structure (snippet) for the Montage Workflow, an astronomical image mosaic engine used at NASA [5]. Each color in the graph describes a set of specific tasks within a workflow stage. Each stage produces outputs used as inputs at subsequent pipelined stages that produce the data product (final result) at the end.

split the overall problem in independent parts, which are later on combined in order to produce final data products. Such time consuming scientific campaigns with high complexity can be organized in pipelines, where each pipeline stage describes a specific set of models and computational tasks organized in batch calculations. At a high level, these interconnected pipelines composed of independent stages is what defines a *scientific workflow* (Figure 2.3). Moreover, scientific workflows are not only common in HPC centres, but also in virtually every sector in industry and academia to analyze and correlate data for predictions and decision support.

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 on reliability. The task of ensuring good performance during execution is delegated to the developers, but with increased use of workflows to process big amounts of data, a closer integration between the WMS and the datacenter RM is of vital importance for improving scientific application performance [9, 8, 2]. Today, 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 [23, 35, 38, 8]. Current methods result in loss of efficiency or utilization, and the problems are likely to be exacerbated with next-generation dynamic workflows. Thus dynamic resource management models that allows 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.

2.2.3 Resource Managers and Sharing

Each RM uses different sharing schemes and policies for multiplexing managed resources, depending on the context applications are deployed. Common RMs in large clusters usually use frameworks such as Mesos [19], Slurm [46], Torque [40], or Omega [38], to allocate resources to their jobs. When sharing resources like CPU, memory, network and file systems (I/O), isolation is a requirement for most applications competing for a common resource. Isolation specifies that minimum levels of a resource capacity will be available when the application needs it, allowing predictable behavior given similar workloads and amount of resources.

With advents of lightweight in-kernel virtualization and isolation tools such as cgroups [28], Linux containers (LXC) [28], and Docker [29], resource orchestrators such as Kubernetes [6] and Docker Swarm [37] have also been considered and 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 [28], 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. In HPC, though, most jobs are batch scripts which run for long times and occupy high percentages of a cluster [35]. As such, Slurm and Torque, common RMs used in HPC centres, are designed mainly to achieve high resource utilization at coarse-granularity like compute nodes (cluster level, i.e., ratio of occupied/booked servers). In such context and higher level granularities, resource utilization refers to the percentage of the whole infrastructure that is occupied by several jobs (space-sharing) at some point in time, and not to how efficiently job allocations are using provided resource capacities (time-sharing) as is the case in the cloud context.

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 [32]. In modern OSs, CFS is configured by using mechanisms such as namespaces and cgroups [28], 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 [15]. In HPC, Backfilling is one of the most used schedule algorithms in space-sharing contexts [39]. Its main advantages is increased utilization at the cluster level, lowering queue waiting time.

2.2.4 Reliability and Availability

In cloud computing, services are subject to a Service Level Agreement (SLA), which generally specifies the availability levels within a time frame the Service Provider guarantees. In another hand, specific users, like those running HPC applications, are often more concerned about Service Level Objectives (SLOs) for alloted resources, which are the terms describing overall performance levels (such as "$latency < X(units)$ or $throughput > Y(units)$") that make up the SLA. Additionally, web services must achieve high availability, which is the measure of percentage of time the infrastructure is in an operable state. Reaching zero fault operations on large hardware and software 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 fault tolerance mechanisms such as RAID (Redundant Array of Independent Disks) and mechanisms for software such as storage replication, Virtual Machine duplication, migration, and etc. Consequently, workloads and runtime systems have been designed to gracefully tolerate and adapt to component faults with little to no impacts on service level performance and availability. A recent model of software development, known as DevOps, advocates for shorter development cycles and reduced costs, separates application logic from its infrastructure management, which enables increased flexibility in regards to system faults, a norm in large datacenters.

All these changes created new demands for modern datacenter RMs. In clouds and HPC, datacenter system reliability directly relates to resource performance metrics because applications and services may not run properly when failures occur. If a system does not support any hardware or software 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 a still chance that application performance degrades, mainly due to re-computing time and also the time taken to restart the failed system and ensure that the failed application works properly 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.

2.3 Challenges

There are many ways performance can be evaluated and measured, specially when comparing different infrastructures with overall 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 were designed to run on a certain hardware, a category of processor, or be used by a specific group of users, datacenter RMs evolved over time to operate on multiple systems and support different and multiple hardware and users. With new classes of adaptive DI jobs [18] and the convergence of many different workloads, job and infrastructure management, combined with needs for fast and efficient resource assignment, are key challenges for modern datacenters [2]. This is particularly true for clusters with heterogeneous resources or with applications that have varying SLOs and diverse workloads with unknown variations.

Traditional RMs have been unable to properly manage these 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 dynamic workloads need, and furthermore results in poor resource utilization (capacity-wise). On the other hand, characteristics from dynamic, non-monolithic schedulers (e.g. Mesos, Yarn, Omega) such as resource state management, optimization for data access and low-latency scheduling, as well as easy extensibility for supporting new policies and classes of workload requirements, are yet to be fully supported in HPC centres [2].

Thus, to be properly managed by a HPC RM, adaptive and DI 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.

2.3.1 Collocation

Resource assignment can be a challenging task 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 [35, 23]. In traditional HPC RMs, allocation is the exclusive assignment of resources to execute a job [23], 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.

Collocation (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 [49, 35], 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 [49, 6].

2.3.2 Isolation

Isolation mechanisms need to be combined with interference detection techniques [10] 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 [30, 45], or to take actions such as throttling low-priority jobs to mend the interference [49].

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 [20]. For instance, these capabilities are essential for doing load-balacing in workflow stream-processing, which demand capabilities like task migration, or changing resource allocations at runtime [20, 2].

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 [49].

- *Using a monitoring Agent on the nodes*: This could be implemented by having an exclusive hardware profiler. Though a very promising approach [38], it needs a specific system architecture for communicating with the RM and it can be hardware dependant [25], limiting its adoption.

- *Linux Cgroup*: Cgroup [28] 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. The LLC problem can already be addressed by cgroup in newer architectures by assigning exclusive portions of the LLC to processes [17], guaranteeing strong isolation enforcement. The main problem with cgroups is that within the frame of a container, tasks may not use all available resources, though idle capacities can be used by other processes.

2.3.3 Multi-Level Scheduling

To support adaptable jobs while at the same time enforcing space sharing and ensuring fair-share usage and that job deadlines are met, modular RMs with multi-level scheduling are proposed [16, 38, 19]. In multi-level schedulers, the process of deciding how to schedule and share available resources are given to applications instead of following an organizational unique policy. RMs like Mesos [19], Torque, and Omega [38] expect workloads (users) to specify resource reservations, for instance how many CPU-cores and memory the application needs. For example, Mesos[3], 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 [19]. 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 [35], a concept now known as Datacenter Operating System (DC/OS) [48, 19].

2.4 Data Analysis for Resource Efficiency

Large datasets such as utilization traces, application logs, and reliability measurements can be analyzed jointly to discover patterns that could not be derived without using data analytic techniques. Generally, HPC infrastructures utilize the latest performance optimized, or domain specific hardware. However, these data analytic techniques often show that jobs do not fully utilize allocated resource capacities [7, 12]. 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 [13]. These 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. 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 [30, 27, 11, 45], 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

[3]Its earlier project was called Nexus [20].

reduce false positives when detecting performance interference (while sharing resources), which is essential in HPC infrastructures.

Use cases for applying data analytics for RM in HPC come from DI and adaptive jobs that can make use of spare resources 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 hardware requirements are very similar to those of traditional HPC applications [2, 26]. 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. What's more, there are types of resource requests typical in HPC that cannot be handled by cloud providers SLOs. For instance, it is unfeasible to allocate whole partitions of resources inside a cloud datacenter for exclusive use and for long times, which is a critical demand for conducting large scale scientific experiments. These contrasts make running DI jobs in HPC infrastructures with existing HPC RMs challenging. Similarly, it becomes more 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 some opportunities that if properly explored will create means for the infrastructural convergence that is now needed for advancing RMs in HPC and cloud computing [2, 23]. First, utilizing existing HPC infrastructures which are being shared by many diverse users (as common, e.g., in academic research) 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 respectively simplify application performance portability and debugging. However, scalability and predictability can at times be contradictory to utilization. In one hand, increase 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 [33, 12]. These three should be considered when proposing new approaches for HPC RM because they relate to the stability and efficiency of a system. In order to allow such diverse SLOs in a same cluster to co-exist, two of the main components in a HPC RM (Figure 2.2) needs to be addressed: resource scheduling and performance control [8]. Today we see a proliferation of new libraries, tools and scripts workarounds [4], all targeting dynamicity in HPC, but with little efforts on integrating such ideas and mechanisms directly into the RM. 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. Furthermore, any combination of static and dynamic

RMs must be simple and scalable, and must detect and gracefully deal with job interference.

Chapter 3

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 neglectable 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 and in Chapter 2. In Paper I, we developed in a tool for enabling a better understanding of the application in order to allow it to communicate with the resource manager and use resources more appropriately in a scientific workflow scenario. In Paper II, we focused on extending the runtime system (the emulator that performs hardware virtualization, virtual machine) to dynamically adjust which fault-tolerant mechanism to use according to application workload. Finally, in Paper III, we investigate how to monitor processor counters to enable finer grained resource allocation on HPC infrastructures via a two-level scheduling architectural approach.

3.1 Paper I

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.

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.

3.2 Paper II

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.

This project is a result of work in the ORBIT project [31], 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.

3.3 Paper III

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.

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.

3.4 Future Work

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 as it can be obtained in several different ways. One metric is *overhead*, the extra resource cost of implementing an abstraction presented to applications. A related metric is efficiency, the lack of overhead in an abstraction. RMs need to allocate resources among applications, and this affects the system performance as perceived by the end user.

Focusing on small number of metrics may bring fairness issues among multiple applications running on the same cluster or machine. Should resources be divided equally between different users and applications, or should some get preferential treatment? 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.

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, but recent challenges on converged infrastructures are demanding new solutions. 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.

In these directions, Paper III targets dynamic execution in HPC clusters, and one of RMs main goals is to have predictable resource assignment of jobs. Thus our goal is to extend on Paper III approach by considering ways to minimize in different ways interference and/or false positives in our co-locations while making use of job prioritization. This is understandable since, for instance majority of HPC jobs are not fault tolerant or are not designed to deal with stragglers between tasks. On the other hand, dynamic and elastic jobs are generally, and by design, developed to deal with those issues.

As such, varied and new combined ways for achieving extreme-efficiency at scale at all layers of the system stack are needed [23]. Since this could be done in different ways, we will be extending on ideas of Paper III for improving future resource utilization in order to perform probabilistic co-schedule of applications.

Bibliography

[1] George Amvrosiadis et al. “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.

[2] M Asch 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.

[3] 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.

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

[5] GB Berriman 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.

[6] Brendan Burns et al. “Borg, omega, and kubernetes”. In: *Queue* 14.1 (2016), p. 10.

[7] 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.

[8] Isaıas Comprés et al. “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.

[9] Ewa Deelman et al. “The future of scientific workflows”. In: *The International Journal of High Performance Computing Applications* 32.1 (2018), pp. 159–175.

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

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

[12] Maja Etinski et al. "Utilization driven power-aware parallel job scheduling". In: *Computer Science-Research and Development* 25.3-4 (2010), pp. 207–216.

[13] 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.

[14] Ian Foster et al. "Cloud computing and grid computing 360-degree compared". In: *Grid Computing Environments Workshop, 2008. GCE'08.* Ieee. 2008, pp. 1–10.

[15] Ali Ghodsi et al. "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types." In: 11.2011 (2011), pp. 24–24.

[16] *Google Data Centers, Efficiency: How we do it.* 2016. URL: `https://www.google.com/about/datacenters/efficiency/internal/` (visited on 01/02/2017).

[17] Andrew Herdrich et al. "Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family". In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2016, pp. 657–668.

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

[19] Benjamin Hindman et al. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." In: 11.2011 (2011), pp. 22–22.

[20] Benjamin Hindman et al. "Nexus: A common substrate for cluster computing". In: *Workshop on Hot Topics in Cloud Computing.* 2009.

[21] 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.

[22] Morris A. Jette, Andy B. Yoo, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management". In: *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003.* Springer-Verlag, 2002, pp. 44–60.

[23] Somesh Jha et al. "A tale of two data-intensive paradigms: Applications, abstractions, and architectures". In: *IEEE BigData Congress, 2014.* 2014, pp. 645–652.

[24] Gregory Katsaros et al. "A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments." In: *CLOSER.* 2011, pp. 232–239.

[25] 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.

[26] 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.

[27] Jason Mars et al. "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations". In: (2011), pp. 248–259.

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

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

[30] 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.

[31] *ORBIT Project*. URL: `http://www.orbitproject.eu`.

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

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

[34] Suraj Prabhakaran et al. "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.

[35] Albert Reuther et al. "Scalable system scheduling for HPC and big data". In: *Journal of Parallel and Distributed Computing* 111 (2018), pp. 76–92.

[36] Albert Reuther 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.

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

[38] Malte Schwarzkopf et al. "Omega: flexible, scalable schedulers for large compute clusters". In: *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM. 2013, pp. 351–364.

[39] Srividya Srinivasan et al. "Characterization of backfilling strategies for parallel job scheduling". In: *International Conference on Parallel Processing Workshops*. IEEE. 2002, pp. 514–519.

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

[41] *United States Data Center Energy Usage Report*. 2016. URL: `https://eta.lbl.gov/publications/united-states-data-center-energy` (visited on 01/02/2017).

[42] Vinod Kumar Vavilapalli 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.

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

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

[45] Hailong Yang et al. “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.

[46] 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.

[47] Andrew J Younge et al. “Efficient resource management for cloud computing environments”. In: *2010 International Green Computing Conference*. IEEE. 2010, pp. 357–364.

[48] Matei Zaharia et al. “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.

[49] Xiao Zhang et al. “CPI 2: CPU performance isolation for shared compute clusters”. In: *8th ACM European Conference on Computer Systems*. ACM. 2013, pp. 379–391.