# LEVERAGING DISAGGREGATED ACCELERATORS AND NON-VOLATILE MEMORIES TO IMPROVE THE EFFICIENCY OF MODERN DATACENTERS

AARON CALL BARREIRO

Dissertation submitted in partial fulfillment of the requirements for the Doctoral Degree in Computer Architecture

Universitat Politècnica de Catalunya - BarcelonaTECH

May 2022

Aaron Call Barreiro : *Leveraging Disaggregated Accelerators and Non-Volatile Memories to Improve the Efficiency of Modern Datacenters*
Dissertation submitted in partial fulfillment of the requirements for the Doctoral Degree in Computer Architecture, September 19, 2022.

SUPERVISORS:
David Carrera
Jordà Polo

TUTOR:
Yolanda Becerra

AFFILIATION:
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya - BarcelonaTECH

LOCATION:
Barcelona

## COLOPHON

The front cover picture shows the north face of mount Eiger, Switzerland. Unlicensed and free-to-use picture from Piyapong Sayduang.

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*".

*Somewhere between the bottom of the climb*
*and the summit is the answer to the mystery why we climb.*
Greg Child

# ABSTRACT

Traditional data centers consist of computing nodes that possess all the resources physically attached. When there was the need to deal with more significant demands, the solution has been to either add more nodes (scaling out) or increase the capacity of existing ones (scaling-up). Workload requirements are traditionally fulfilled by selecting compute platforms from pools that better satisfy their average or maximum resource requirements depending on the price that the user is willing to pay. The amount of processor, memory, storage, and network bandwidth of a selected platform needs to meet or exceed the platform requirements of the workload. Beyond those explicitly required by the workload, additional resources are considered stranded resources (if not used) or bonus resources (if used).

Meanwhile, workloads in all market segments have evolved significantly during the last decades. Today, workloads have a larger variety of requirements in terms of characteristics related to the computing platforms. Those workload new requirements include new technologies such as GPU, FPGA, NVMe, etc. These new technologies are more expensive and thus become more limited. It is no longer feasible to increase the number of resources according to potential peak demands, as this significantly raises the total cost of ownership.

Software-Defined-Infrastructures (SDI), a new concept for the data center architecture, is being developed to address those issues. The main SDI proposition is to disaggregate all the resources over the fabric to enable the required flexibility. On SDI, instead of pools of computational nodes, the pools consist of individual units of resources (CPU, memory, FPGA, NVMe, GPU, etc.).

When an application needs to be executed, SDI identifies the computational requirements and assembles all the resources required, creating a composite node. Resource disaggregation brings new challenges and opportunities that this thesis will explore. This thesis demonstrates that resource disaggregation brings opportunities to increase the efficiency of modern data centers. This thesis demonstrates that resource disaggregation may increase workloads' performance when sharing a single resource. Thus, needing fewer resources to achieve similar results. On the other hand, this thesis demonstrates how through disaggregation, aggregation of resources can be made, increasing a workload's performance.

However, to take maximum advantage of those characteristics and flexibility, orchestrators must be aware of them. This thesis demonstrates how workload-aware techniques applied at the resource management level allow for improved quality of service leveraging resource disaggregation. Enabling resource disaggregation, this thesis demonstrates a reduction of up to 49% missed deadlines compared to a traditional policy. This reduction can rise up to 100% when enabling workload awareness.

Moreover, this thesis demonstrates that GPU partitioning and disaggregation further enhances the data center flexibility. This increased flexibility can achieve the same results with half the resources. That is, with a single physical GPU partitioned and disaggregated, the same results can be achieved with 2 GPU disaggregated but not partitioned.

Finally, this thesis demonstrates that resource fragmentation becomes key when having a limited set of heterogeneous resources, namely NVMe and GPU. For the case of an heterogeneous set of resources, and specifically when some of those resources are highly demanded but limited in quantityt. That is, the situation where the demand for a resource is unexpectedly high, this thesis proposes a technique to minimize fragmentation that reduces deadlines missed compared to a disaggregation-aware policy of up to 86%.

# ACKNOWLEDGMENTS

Primer de tot voldria mostrar el meu absolut agraïment als meus director de tesis, David i Jordà. Sense vosaltres això no hauria estat mai una realitat. Gràcies per la paciència infinita en fer-me entendre què tot el que feia valia la pena. També moltes gràcies a en Josep Lluís que en la part final d'aquesta tesis ha estat fent tasques de director i ha resultat molt més fàcil pulir alguns detalls finals. Sense la teva ajuda no hi hauria hagut recursos per acabar la tercera contribució.

Thanks as well to the reviewers and jruy members for the effort made in reading and assessing this thesis. Julita, Jordi, Vicenç, Waheed, Marcelo and Toni.

Gràcies a tots els companys d'oficina, passats i presents, que d'alguna forma han fet que tot aquest procés hagi resultat més fàcil. Els dinars amb debats eterns i acudits tontos ajuden a oblidar-se dels problemes. També agraïr al grup d'amics de muntanya, Flor, Marco, Carlos, Armando, David, Olga, Alberto, Minerva, Edu, Maria, Eva, Sophie, Danila, Domenico, Xavi, Laures, Sergi, entre molts altres que costa enumerar. Tots ells m'han ajudat a relaxar-me i a oblidar-me de tots els mals i poder desconnectar.

Mil gràcies als amics per escoltar-me en els moments de crisi. Moltes gràcies a l'Andreu per sempre estar disposat a prendre alguna cosa i aguantar-me. Mil gràcies a Alvaro, Nando, Dolors, Vero, Alberto, Maialen, Lluís per acompanyar-me i fer-me distreure dels moments delicats. Especial menció a l'Eli que porta sent una bona aimga des de fa més de 15 anys tot i els obstacles que hem tingut. Sempre ajuda saber que hi ha algú que et pot ajudar quan ho necessitis. Especial menció a que sempre respons els suggeriments de fer un soparet en 2 minuts però per tota la resta de coses trigues 4 dies, quins ovaris.

Gràcies Alberto que hem compartit tots els dolors i mal tràngols del final de les nostres respectives tesis. Ha estat un plaer compartir aquest camí junts i és tota un honor poder acabar i celebrar-ho plegats.

Moltes gràcies per fer-me distreure infinit i donar-me totes les estones de riure infinites que necessitava Lucía, Patrícia, Hector i Carme. Som uns conquistadors de l'inútil que t'hi cagues.

Muchas gracias Cris que me has acompañado y ayudado en todos los momentos en los que dudaba y quería dejarlo. Sin ti esta tesis nunca hubiera ocurrido. Ha sido un honor compartir tantos momentos altos y bajos a tu lado. Aunque contigo siempre acabo sudando mucho y me descubres deportes caros, así que no sé, igual no deberías estar aquí. Y encima me echas muchas especies picantes sin previo aviso.

Finalment agraïr als meus pares el seu suport incondicional, de vegades absurd ja que no acabaven d'entendre res. És sorprenent com algú sense entendre res de res pot confiar en tu cegament i dir-te amb convicció que te'n sortiràs, així, sense més, perquè sí.

# CONTENTS

# ACRONYMS

HPC High-Performance Computing
GPU Graphics Processing Unit
FPGA Field Programmable Gate Array
NVMe Non-Volatile Memory Express
NVM Non-Volatile Memory
SLA Service Level Agreement
SDI Software-Defined Infrastructure
ROI Return on Investment
Intel RST Intel Rapid Storage Technology
PCIe Peripheral Component Interconnec Express
PCI Peripheral Component Interconnec
NVMeOF NVMe over Fabrics
RDMA Remote Direct Memory Access
NFS Network File System
HPC High-Perforamnce Computing
QoS Quality of Service
vGPU Virtual GPU
GPFS General Parallel File System
API Application Programming Interface
SSD Solid-State Drive
EDF Earliest Deadline First
DRF Dominant Resource Fairness
vGPU Virtual GPU
VM Virtual Machine
VMP Virtual Machine Placement

# LIST OF FIGURES

# LIST OF TABLES

# 1

# INTRODUCTION

## 1.1 THESIS CONTEXT AND MOTIVATION

Traditional data centers consist of computing nodes that possess all the resources physically attached. When there was the need to deal with more significant demands, the solution has been to either add more nodes (scaling out) or increase the capacity of existing ones (scaling-up). The drawback of this scheme is that while re-configuring the infrastructure, there is a period when several resources are unusable. This period is relatively short, and thus the method has been able to deal with the demands for a few years successfully.

Workload requirements are traditionally fulfilled by selecting compute platforms from pools that better satisfy their average or maximum resource requirements depending on the price that the user is willing to pay. The amount of processor, memory, storage, and network bandwidth of a selected platform needs to meet or exceed the platform requirements of the workload. Beyond those explicitly required by the workload, additional resources are considered stranded resources (if not used) or bonus resources (if used).

The conventional data center design target is to include enough infrastructure capacity to meet peak demands or to arrange the data center platform for bursting when needed while keeping in mind the total cost of ownership. Both of those methods depend on having enough foresight about what peak demands will be and the trickier problem of predicting what possible bottlenecks might arise when applications approach capacity. Both are risky and do not account for the unexpected.

Meanwhile, workloads in all market segments have evolved significantly during the last decades. Today, workloads have a larger variety of requirements in terms of characteristics related to the computing platforms. Workloads are tuned to work optimally on specific computing, memory, and storage configurations. On top of that, current workloads tend to be composed of multiple phases with different behaviors and resource requirements. Those workload requirements include new technologies (i.e., GPU, Field Programmable Gate Array (FPGA), NVMe, etc.). Moreover, the workloads are set to run under specific conditions of storage, memory, bandwidth, etc. When those requirements are not met, their performance

drops drastically. However, those resources are expensive. The traditional approach tells us to place the same resources on all computational resources. However, this is no longer feasible under newer environments such as the edge cloud. Under edge computing small devices gather and compute data for different purposes. To list a few examples, security can be enhanced in airports through feeding cameras input to small devices with face recognition workloads targeting individuals of interest. Sensors can be installed across city to measure traffic and improve mobility. Small weather stations can carry some forecast predictions in-situ to prevent catastrophic events in key areas such villages close to the sea or frequented climbing routes in the mountains. All this devices carry computationally expensive works and at the same time they are unable to host large amount of resources that could speed up the process. The solution for this is to move part of the data and computation to the cloud. However, those cloud datacenters are often city datacenters or regional datacenters, with limited space and budget. Moreover traditional cloud computing is no longer sufficient to meet all demands [11].

Consequently, a heterogeneous set of resources is limited, and not all computing nodes possess it. The decision on where to place them is non-trivial. As a result, only a few computational nodes will have access to specialized resources. Thus, some computational nodes will be primarily dedicated to handling such workloads. However, if there is peak demand for those workloads, it is sure the data center won't be able to satisfy all their requirements and fulfill the Service Level Agreement (SLA).

SDI, a new concept for the data center architecture, is being developed and among others, can address those issues. The main SDI proposition is to disaggregate all the resources over the fabric to enable the required flexibility. On SDI, instead of pools of computational nodes, the pools consist of individual units of resources (CPU, memory, FPGA, NVMe, etc.). When an application needs to be executed, SDI identifies the computational requirements and assembles all the resources required, creating a composite node.

Disaggregation provides higher flexibility and malleability. Under a disaggregated data center, instead of pools of nodes possessing all the needed physical hardware, those nodes get their hardware through pools of resources. Those pools contain units of resources (storage, GPUs, FPGAs, etc.). When an application needs to be executed, the data center identifies its resource requirements and pools the resources into an available node. Consequently, the need to find a node containing both enough CPU cores and the resources disappears, as the resource can now be pooled. If we move into a disaggregated data center, the lack of specialized resources can be solved, as all computational nodes may gain access to those resources. When the node no longer needs to run the workload, the resource may go into another one. Although the theory seems straightforward, it poses several questions and challenges.

> Research challenge 1: we need to find out whether disaggregation has a negative impact on the performance of workloads when resources are far away from the computing node where they are being executed on. If so, there is a need to find a solution to it. Then we need to figure out how to leverage disaggregation benefits to improve the efficiency of individual workloads.

On the other hand, although disaggregation provides flexibility we still have a limitation on the number of resources. As a consequence, the orchestrator must know about disaggregation to take full advantage of it. It brings a new challenge:

> Research challenge 2: we must decide where to place a resource when there is peak demand for a specific resource. It is of utmost importance to know whether it is good that a computing node to have attached more than one disaggregated resource or not, and to know whether or not should the data center has computing nodes specific for some workloads. The orchestrator must also act accordingly when the demand changes.

Finally, since the specialized resources are heterogeneous, their characteristics and offerings under disaggregation are also different. Not all resources provide the same level of flexibility when disaggregated. It leads us to the final research challenge this thesis attempts to address.

> Research challenge 3: there is a need to discover the characteristics of accelerators and non-volatile memories under disaggregation. There is a need to have different strategies for each resource to leverage its maximum benefit. Moreover, when deciding how to allocate those resources into computing nodes, we must be aware of the different levels of flexibility they provide under disaggregation.

The industry is already pursuing the disaggregated data center based on SDI. Intel has been developing a new software-defined infrastructure framework called Intel Rack Scale [33]. Intel Rack Scale design promises to address the need for scaling to larger resources placed anywhere in the world and quickly and dynamically adapt to the new requirements. It goes even further, enabling faster response to business demands for Return on Investment (ROI) and higher performance, all while lowering costs and simplifying complexity for IT and the overall business. Recently there has been an initial commercial deployment of the architecture [23]. Other companies are bringing their own SDI data centers, such as Facebook [22] and HP [32].

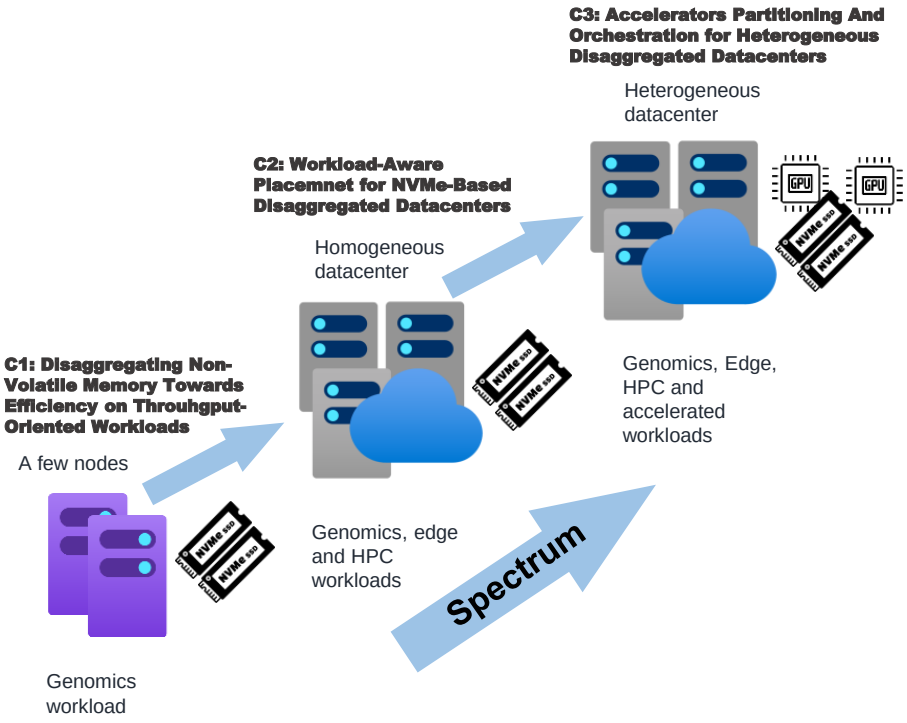



Figure 1.1: Thesis contributions

## 1.2 THESIS CONTRIBUTIONS

### 1.2.1 *Thesis statement*

The exposed contents motivated the following thesis statement:

*"It is possible to improve the efficiency of data centers by leveraging the additional freedom that resource disaggregation provides."*

To achieve this goal, we divided the work into three main contributions (depicted in figure 1.1):

C1: Disaggregating Non-Volatile Memory Towards Efficiency on Throughput-Oriented Workloads

C2: Workload-Aware Placement for NVMe-Based Disaggregated Datacenters

C3: Accelerators Partitioning And Orchestration for Heterogeneous Disaggregated Datacenters

### 1.2.2 *Disaggregating Non-Volatile Memory Towards Efficiency on Throughput-Oriented Workloads*

The first contribution analyzes how disaggregation of NVMe can be leveraged to improve the efficiency of throughput-oriented workloads. In particular, it utilizes a genomics workload bandwidth-sensitive. First, we characterize the workloads' properties and their behavior when sharing NVMe or composing many of them into a single one. Secondly, it allows observing that we can share the resource without taking significant degradation up to a certain threshold. While the latter aids in increasing this threshold. Finally, this contribution shows how having disaggregated resources may improve overall performance by allowing even higher levels of sharing.

The achievements of this contribution are:

- Explored how to increase resource usage through resource sharing.
- Explored resources' performance when sharing.
- Compose resources out of a few units, enhancing its capabilities.
- Leverage disaggregation to improve workloads' performance.

### 1.2.3 *Workload-Aware Placement for NVMe-Based Disaggregated Datacenters*

The gathered knowledge is used in the second contribution, allowing to orchestrate the data center in a workload-aware fashion, showing the benefits of proper orchestration of disaggregated NVMe. Disaggregation brings the flexibility of assigning or re-assigning a resource to the nodes who need it. This allows meeting workloads SLA under tight conditions much better than workload-unaware policies.

In this contribution, we enable orchestrators to leverage this knowledge in data center management. In this contribution, a policy is made to decide when and where to pool disaggregated resources for the workloads depending on the current observed demands. The achievements of the contribution are:

- Two model-aware placement policies. Maximize composition and minimize fragmentation. The former leverages resource composition and sharing, while the latter intends to minimize the effects of fragmentation.
- A disaggregation-aware scheduling policy that decides which of the placement policies fits best under the current data center load.
- A comparative analysis of the benefits of resource disaggregation versus physically-attached resources.

### 1.2.4 *Accelerators Partitioning And Orchestration for Heterogeneous Disaggregated Datacenters*

In the first two contributions, it is shown how disaggregated resources improve the efficiency of the data center. However, disaggregation still has one limitation: while a resource is pooled in a node, it can only be used by workloads within that node. That is, pooled resources can't be shared with other nodes until they are detached and pooled to them. To do so, however, workloads using the resource must be completed. In this contribution we leverage accelerators partitioning, concretely GPU partitioning, to expose portions of the same GPU to several nodes simultaneously. We explore how this partitioning impacts performance on accelerated workloads and how it can be leveraged to enhance data center efficiency further. Finally, we add NVMe-based workloads to manage a heterogeneous set of disaggregated resources into the mix. This contribution shows the impact of fragmentation when dealing with a heterogeneous group of resources and proposes a policy to manage the data center effectively.

- An study of the performance impact of resource partitioning on accelerated workloads.
- An allocation policy leveraging disaggregated GPU partitions to enhance data center efficiency.
- A placement policy dealing with the fragmentation induced by heterogeneous resources in the data center.

# 2

# BACKGROUND

## 2.1 SOFTWARE-DEFINED INFRASTRUCTURES

Typical cloud data centers encompass a set of nodes holding all the physical resources. When a task requests a set of resources (typically CPU, memory, storage and bandwidth) the cloud orchestrator finds a suitable physical machine having enough of those resources and assigns a Virtual Machine (VM) to it. Moreover, cloud providers are set to fulfill an SLA agreement. Thus, it is not always acceptable that a task must wait to have enough physical resources available. Consequently, cloud providers are set to have more resources than would be strictly necessary to run all tasks if no SLA had to be met. This results into datacenters with a high percentage of resource slack of up to 56% in some providers [13]. On the other hand, a task may have all the available resources in the datacenter, but not all of them together in a single physical machine. Since resources are physically-attached, the task must wait before it can run, be assigned on a sub-optimal machine and underperform or a combination of both.

SDI is designed to solve, at least partially those issues. In SDI, compute platforms are constructed on the fly and are connected within a topology to fulfill the requirements of the software (or workload). The built composite platform will meet the specific platform requirements of the workload.

The traditional definition of a platform evolves to a composite node or composite platform. Unlike existing solutions, resources that traditionally have been attached to the local platform are disaggregated in the data center. As depicted in Figure 2.1, resources are distributed in different resource pools. When an application needs to be executed, the orchestration layer and the SDI manager work together to identify and assemble resources in the SDI pools to satisfy application/user requirements. Once all the resources are assembled , creating the composite node, the operating system is booted, and the application is instantiated. Unlike previous mechanisms to access remote resources (such as General Parallel File System (GPFS) [71] or Remote Direct

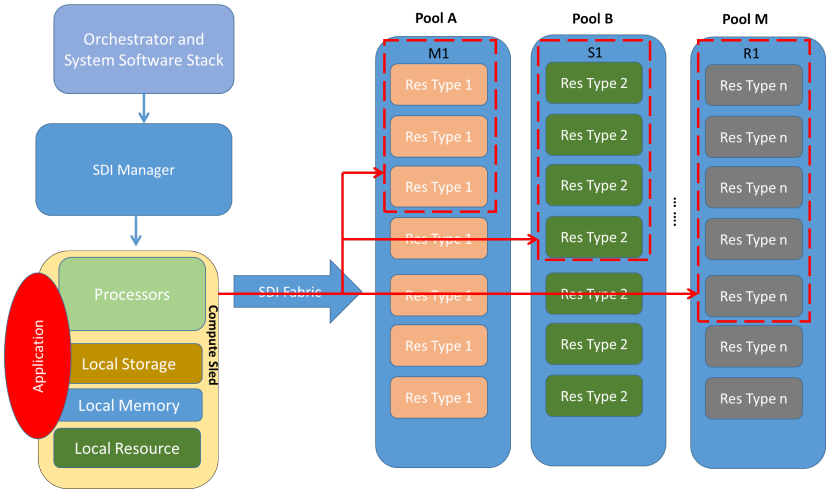



Figure 2.1: Resource Disaggregation in SDI architectures

Memory Access (RDMA)), resources are exposed as local resources to the software stack running on the composite node. Despite being distributed across the data center in multiple pools, the SDI architecture provides the illusion that applications are executed in a traditional platform with the exact requirements that the application/user requested. Once the application finishes its execution, the node is decomposed, and resources are placed back in their corresponding queues.

In the last years, conceptual visions and prototype implementations of SDI data centers have increased. In [37], the authors provide an abstract definition focused on scalability and dynamic infrastructure changes depending on workload demands. Additionally in [5], the need to scale over heterogeneous groups of resources is highlighted with special emphasis on the role of the SDI manager.

On the implementation side, Intel has developed Rack Scale [33], one of the first SDI frameworks that are closing the gap between academic research and real data centers. Rack Scale allows dynamic composition of nodes, fully disaggregating its resources in pools, such as CPU, storage, memory, FPGA, GPU, etc. Recently it has released a first commercial product as well, Falconwitch [23].

The work in [1] proposes an infrastructure solution to compose nodes while ensuring resources' requirements of workloads are met. They show two approaches on how to build an infrastructure to build the composite nodes.

Facebook has engaged with Intel to explore this SDI implementation, developing the Facebook Disaggregated Rack [22]. On the other hand, HP is also conducting its own SDI data center implementation with "The Machine" [32].

Additionally, the dRedBox project [38] presents a disaggregated architecture similar to SDI. In this prototype, there are many microservers using pools of resources. Hence

there is no physical node composition. On the other hand, the linking of resource pools is carried through a hypervisor. Thus, it becomes a software connection rather than a physical one as it happens on a pure SDI data center.

Other prototypes allowing disaggregation of Peripheral Component Interconnec (PCI) devices are ExpEther[19]. ExpEther relies on ethernet protocol as a networking fabric. ExpEther enhances the Peripheral Component Interconnec Express (PCIe) bus by extending connection distances, supporting device numbers beyond the limitation of local PCI port counts, and alleviating localized electrical power and heat dissipation restrictions that a conventional box-type computer would encounter. In addition, with the full support of the PCI Hot-Plug protocol, dynamic reconfiguration of the computing platform is also possible.

In [75], an optical architecture is proposed leveraging slotted TDMA/WDM switching to realize dynamic resource allocation with sub-wavelength granularity, thus realizing a low cost and power consumption, scalable data center network. Dynamic reconfiguration of the slotted network vouches for low latency operation of the data plane, and hence, it fulfills the requirements of the envisaged disaggregated data center infrastructure.

## 2.2 DIRECT-ATTACHED NVME AND NVME OVER FABRICS

NVMe [74] is a logical device interface specification for accessing direct-attached non-volatile storage media via a PCIe bus. The NVM acronym stands for non-volatile memory, and it is commonly flash memory that comes in the form of Solid-State Drive (SSD)s. NVMe, as a logical device interface, has been designed from the ground up to capitalize on the low latency and internal parallelism of flash-based storage devices [61], mirroring the parallelism of contemporary CPUs, platforms, and applications.

By design, NVM Express allows host hardware and software to exploit the levels of parallelism existing in modern SSDs entirely. As a result, NVM Express reduces I/O overhead brings various performance improvements compared to previous logical-device interfaces, including multiple, long command queues and reduced latency.

NVMeOF defines a typical architecture that supports a range of storage networking fabrics for the NVMe block storage protocol over a networking fabric. It includes enabling a front-side interface into storage systems, scaling out to large numbers of NVMe devices, and extending the distance within a data center over which NVMe devices and NVMe subsystems can be accessed [84]. Newer versions now allow connecting through RDMA, eliminating middle software layers. Thus, NVMe over fabrics brings the

opportunity to disaggregate storage over fabric that can be used either as storage or memory expansion.

In [41] they introduce a software to enable disaggregation of NVMe. This proposal was prior to the appearance of NVMeOF, and therefore there are big differences both in the implementation side and the performance. The main difference between their work and NVMeOF is that their software happens on top of the operating system and heavily relies on TCP protocol. While NVMeOF is a kernel driver and allows RDMA protocol to bypass the overheads of using TCP. Moreover, the proposed work requires two software layers: one on the clients pooling the NVMe, and a second on the server hosting the physical devices. Since all these software layers are on top of the operating system and on top of TCP or UDP protocols, the overall overhead compared to using IB with RDMA protocol as happens with NVMeOF is much bigger. Their work concludes that write operations get highly impacted by the increased latency compared to physically-attached NVMe, a situation that does not occur on NVMeOF. On the other hand, existing flash devices schedule requests from different NVMe hardware queues using simplistic round-robin arbitration. To guarantee SLAs, ReFlex has to use a software scheduler that implements rate limiting and priorities. At the time of the paper, the authors did not enable this feature on their scheduler, thus not being able to provide the same guarantees.

In [31], the authors make an evaluation of the impact on workloads of disaggregating NVMe over fabrics versus using it locally. It is concluded there is not a noticeable impact for the applications on using NVMeOF.

## 2.3 DISAGGREGATION OF ACCELERATORS

[79] proposes an architecture to disaggregate FPGA, claiming gains towards virtualization or containerization of FPGA. In [2] they continue their work by scaling it into a scaled data center with 64 disaggregated FPGA.

GPU disaggregation has been researched, and there are technologies such as rCUDA [68][67] enabling GPU disaggregation through Application Programming Interface (API) remoting. However, exposing only an API reduces the capabilities to those allowed by it. Moreover, many solutions using this technique are limited to specific API versions, making them obsolete in a few years or even months after new GPU generations appear. [17] proposes a different scheme to disaggregate resources, bypassing the limits of solely exposing an API. The proposal builds an entire SDI architecture. Falconwitch [23] is a proprietary technology achieving GPU disaggregation allowing for topology alteration supporting multiple hosts and a software-defined fabric. [69] makes a comparative

study of GPU disaggregation techniques [62][68][28] using virtualization on three generations of GPUs.

## 2.4 ORCHESTRATION

One of the key allocation problems on cloud computing is deciding on which physical machine to allocate a VM. This problem is commonly referred as a Virtual Machine Placement (VMP) problem. Those problems are typically NP-hard problems and heuristics are used to be solved. [15] presents a proposal on which the cloud encompasses a set of virtual machines with a given characteristics that need to be assigned into physical machines. Their proposal focuses on solving the scenario when there is a sudden and urgent need to assign VM to workloads. To solve this they view the VMP as a multi-objective function with two goal: (a) to reduce the amount of physical machines required to assign the VMs and (b) to reduce as much as possible the distance between the performance of a physical machine and the VM. That is, to assign the VM into a physical machine that meets their performance demands. They assume a VM may be assigned a sub-optimal physical machine because of the need of meeting urgent demands. To solve the NP-hard problem they apply a genetic algorithm as an heuristic. Similar works solving the problem of assigning resources to virtual machines can be found in [73][54][42][80][49][82]. Although those works may solve a similar problem that this thesis wants to solve all of them share in common of having the issue that a physical machine may have availability for some resources, while having some others occupied. Thus, the decisions are either on placing the machines either way under-performing, waiting or a mix of different strategies. This is precisely the key characteristic of resource disaggregation: if the resources are dissaggregated, there is no need to find a suitable physical machine, we just need to find suitable resources on the pool of disaggregated resources and then compose a machine out of them. In practical term, the pool of resources becomes a fat node with all the physical machines. Thus, reducing the complexity of VMP problems as it is no longer required that a single physical machine meets all the demands. Instead, a combination of them is enough (virtually, as those are no longer physical machines).

Orchestration must be redefined in the SDI environment to consider resources' distribution, granting the best possible for the composed nodes. In this respect, [48] presents a cluster partitioning approach for High-Perforamnce Computing (HPC) service optimization based on analyzing quality metrics and job characteristics. An algorithm is proposed to partition a cluster into different sub-clusters dynamically. Those sub-clusters target the different kinds of workloads characteristics that has been identified

on the datacenter. This algorithm is based on an analytical model that captures system performance in several scenarios, further based on a model of job characteristics created using various distributions from data obtained on an HPC cluster. Regarding cloud infrastructures, [76] proposed a scheduling algorithm that groups the available resources into clusters and then performs job allocation among the different clusters of resources. The method also relies on similar provisioning tasks to the same cluster, for which the user provides how many groups of tasks and resources need to be created after the cluster phase. Similarly [43] presents a resource allocation strategy to divide the system into pools of core nodes and accelerator nodes for data analytics workloads and then dynamically adjust the size of the pools to optimize utilization and cost.

On the other hand, [63] presents a scheduler proposal for SDI environments to help allocate resources on-demand, trying to maximize the number of resources being used and minimize node latency.

## 2.5 SMUFIN: A THROUGHPUT-ORIENTED GENOMICS WORKLOAD

Most currently available methods for detecting genomic variations rely on an initial step that involves aligning sequence reads to a reference genome generally using Burrows-Wheeler transform [45], which has an impact not only on performance, but also on the accuracy of results. First, tumoral reads that carry variation may be harder or impossible to align against a reference genome. Second, the use of references also leads to interference with millions of inherited (germline) variants that affect the actual identification of somatic changes, consequently decreasing the final reliability and applicability of the results. The initial alignment also has an impact on subsequent analysis since most methods are tuned to identify only a particular kind or size of mutation [56]. Alternative methods that don't rely on the initial alignment of sequenced reads against a reference genome have been developed. In particular, the application used in this work is based on SMUFIN [58], a reference-free approach based on a direct comparison between normal and tumoral samples from the same patient. The basic idea behind SMUFIN can be summarized in the following steps: (i) input two sets of nucleic acid reads, normal and tumoral; (ii) build frequency counters of substrings in the input reads; and (iii) compare branches to find imbalances, which are then extracted as candidate positions for variation.

Internally, SMUFIN consists of a set of checkpointable stages that are combined to build fully-fledged workloads (Figure 2.2). These stages can be shaped on computing platforms depending on different criteria, such as availability or cost-effectiveness, allowing executions to be adapted to its environment. Data can be split into one or

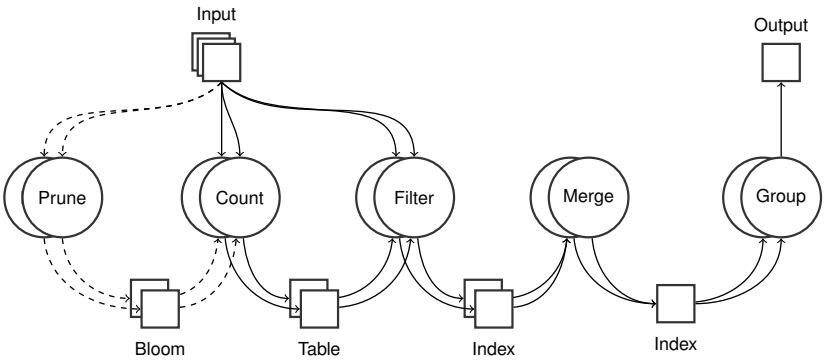



Figure 2.2: SMUFIN's variant calling architecture: overview of stages and its data flow[7]

more *partitions* and each one of these partitions can then be placed and distributed as needed: sequentially in a single machine, in parallel in multiple nodes, or even in different hardware depending on the characteristics of the stage. Data partitioning can be effectively used to adapt executions to a particular level of resources made available to SMUFIN, because it imposes a trade-off between computation and IO. This data partitioning can be achieved by going multiple times through the input data set that corresponds to each stage: *Prune*, *Count*, and *Filter*. In practice, systems with high-end capabilities will not require a high level of partitioning, and hence IO, is what ends up with scale-up solutions; on the opposite side of the spectrum, lower-end platforms are able to run the algorithm by partitioning data and duplicating IO, leading to scale-out solutions. The goal of each one of the stages is as follows:

- *Prune*: Discards sequences from the input by generating a bloom filter of k-mers that have been observed in the input more than once. Allows lowering memory requirements at the expense of additional computation and IO.
- *Count*: Builds a frequency table of normal and tumoral k-mers in the input sequences. More specifically, k-mer counters are used to detect imbalances when comparing two samples.
- *Filter*: Selects k-mers with imbalanced frequencies, which are candidates for variation, while also building indexes of sequences with such k-mers.
- *Merge*: Reads and combines multiple filter indexes from different partitions into single, unified indexes. Merging indexes only involves simple operations such as concatenation, OR on bitmaps, and appending.
- *Group*: Matches candidate sequences that belong to the same region. First, selecting reads that meet certain criteria, and then retrieving related reads by looking up those that contain the same imbalanced k-mers.

One of the main characteristics of the current version of SMUFIN [7] is its ability to use NVM as a memory extension. This can be exploited in two different ways. First,

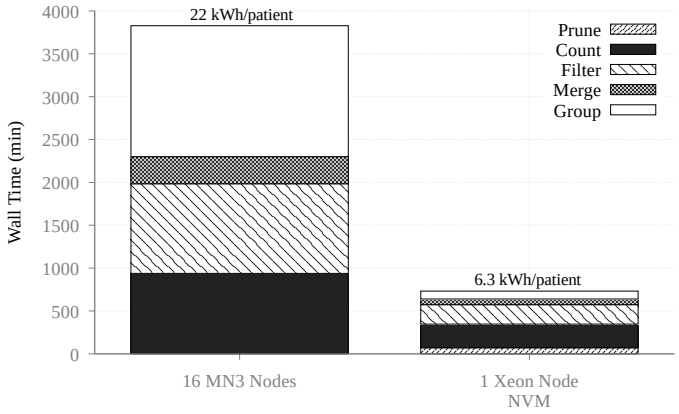



Figure 2.3: Aggregate CPU time of a SMUFIN execution running in 16 MareNostrum nodes and in 1 Xeon-based node with NVM. Power consumption per execution (one patient) shown for reference.[7]

using an NVM optimized Key-Value Store such as RocksDB, and second, using a custom optimized swapping mechanism to flush memory directly to the device. When such memory extensions are available, a maximum size for the data structures is set; once such size is reached, data is flushed to the memory extension while a new empty structure becomes available. Generally speaking, bigger sizes are recommended: they help avoid duplicate data, and also lead to higher performance, as writing big chunks to a Non-Volatile Memory allows exploiting internal parallelism typical of flash drives [14].

SMUFIN's performance greatly benefits from NVM, as shown in Figure 2.3[7], which compares an execution in 16 machines in a supercomputing facility (left) and a scale-up execution in a single node with NVM enabled (right). The latter leads to faster executions and lower power consumption. NVM can be leveraged in some way in most SMUFIN stages, and the experiments performed in this thesis are focused on *Merge* using the RocksDB-based implementation, which is one of the most IO intensive of the pipeline. However, other stages have similar characteristics and the same techniques can be used elsewhere.

SMUFIN's technique can be leveraged for personalized medicine, where patients DNA is processed using the SMFUIN method to identify potential variations in the genome that may lead to diseases. In personalized medicine, response time is critical: it is not reasonable to wait more than a certain amount of time, as in that time, the patient could develop a severe condition. Thus, in the case of SMUFIN, proper and timely access to NVM is critical, becoming another reason for selecting it as an element for the context of this thesis.

## 2.6 YOLO: YOU ONLY LOOK ONCE

Object detection algorithms are a topic of interest in many fields of image detection: security, observation, picture retrieval, vehicle identification, autonomous driving, etcetera. Proof of this are the many algorithms performing object detection that appeared during the last decades, such as [55] [81] [72] [46].

Object detection algorithms are computationally expensive, and this has been the main bottleneck in terms of performance: the tasks for which they are targeted require fast detection, however accessing to resources providing enough computational capabilities is not always that fast. With the appearance of co-processors such as GPU and FPGA, algorithms could be optimized and increase its performance [53] [24].

On top of these, a new algorithm was released, YOLO [70]. YOLO falls into the category of two-stage detectors. Those two stages are:

- Potential regions detection: it detects all regions of an image that can potentially contain an object.
- Image classification: out of the potential regions containing an object, it classifies the regions into actual objects.

The main drawback of two-stage detectors compared to single-stage detectors was the many inferences that had to be made in a single frame to identify all potential regions and classify its objects. However, that stage divisions granted higher accuracy on the detections. On the other hand, single-stage detectors did a similar initial procedure, but instead of using inference algorithms to find potential regions, they selected an arbitrary number of regions and then classified them into images. They achieved higher performance but less accuracy.

The key difference with YOLO is that it is a one-stage detector but performs the steps of a two-stage detector. Instead of selecting arbitrary boxes, YOLO network uses features from the entire image to predict them and assign them a score related to the probability of containing a real object. Those boxes are generated for all classes at once, reducing the performance overhead of previous one-stage algorithms. It then utilizes classifies those boxes into actual objects. YOLO has become a *de facto* standard for object detection. Its initial paper has over 300 citations.

In the context of this thesis, YOLO helps us demonstrate a real-world application suited for edge computing that benefits from co-processors acceleration. As we said earlier, the amount of specialized resources such as GPU that an edge node can hold is limited, especially on edge. For this reason, it is an optimal candidate to benefit from resource disaggregation. We can disaggregate the accelerators from the edge nodes

and place them together in a single rack, improving overall efficiency by allowing all nodes access to accelerators.

# 3

# DISAGGREGATING NON-VOLATILE MEMORY TOWARDS EFFICIENCY ON THROUGHPUT-ORIENTED WORKLOADS

This chapter describes the first contribution of this thesis. This contribution explores how disaggregating NVMe devices adds an extra layer of freedom. We explore how we can leverage resource sharing of NVMe resources on SMUFIN, a genomics throughput-oriented workload. Additionally, we enable resource composition, consisting of aggregating several resources into one bigger unit. These two characteristics increase workloads' performance under some circumstances explored in this contribution. Finally, we combine it with disaggregation through NVMeOF. We show how resource disaggregation allows sharing the resource across nodes, which helps avoid interferences on available CPU/memory.

The topics covered in this contribution are:

1. Characterization of SMUFIN workload under resource sharing and composition
2. Study how disaggregation eliminates interferences to improve efficiency
3. Propose a path to follow to leverage these properties on a data center

## 3.1 RESOURCE SHARING

Data centers have to deal with thousands of workloads demanding an heterogeneous amount of resources and requesting certain SLA. Resource managers are expected to handle resource allocation according to the workload estimated demands. Those workloads provide a wall time, a time interval during which the manager will allocate the resources to the workload. For the users, it is hard to predict how long their workloads' will run, and if they demand less time than the workload needs, they will lose all progress and will have to start all over again. Consequently, it is often the situation that the demanded amount of time exceeds the time during which the workload runs. This situation happens as otherwise, the workload could not

successfully finish and lose its resources. It is also frequent to demand more resources than they really need or that the usage of the required resource is only partial. All these many situations happening in the thousands of workloads translates into large periods upon which resources are allocated but not effectively being used. Moreover, while a resource is assigned to the workload, no other can use it regardless of the lack of usage. Hence, the data centers may potentially finish with a large percentage of locked and unused resources.

One solution the data centers can tackle is to share the resource across workloads. However, resources such as memory, FPGA, GPU, or other devices can be used in a fashion that does not allow for sharing without proper control mechanisms to avoid conflicts. Therefore, the workloads need to be aware of that situation and be adapted accordingly to enable resource sharing. This becomes a strong limitation for current data centers. On the other hand, even when control mechanisms are in place under resource sharing, individual performances are often degraded under resource sharing. Thus SLA may not be fulfilled. To know which workloads are good candidates to establish resource sharing or composition, we need specific knowledge about them. Thus, characterizing them under such conditions and building a performance model is critical to making orchestrators acknowledgeable. This contribution aims to provide this characterization to help build a model.

Many implementations to enable resource sharing and aggregation on NVMe devices have appeared in recent years. In this thesis, we use the implementation provided by Intel, Intel Rapid Storage Technology (Intel RST) [34]. The explored characteristics are:

WORKLOAD-UNAWARE RESOURCE SHARING. It allows partitioning NVMe and each one of these partitions can be exposed to workloads and computational nodes as an exclusive resource. This translates into workload-unaware resource sharing, which can lead to improved resource efficiency by maximizing resources' usage.

RESOURCE COMPOSITION. Certain resources can be aggregated and exposed as a single, physically attached resource. Instead of accessing individual units, accessing combined resources enables increased capacities that lead to improved performance. For instance, two NVMe disks with a bandwidth of 2GB/s each can be composed and exposed as a single one with twice as much capacity and bandwidth, providing 4GB/s.

## 3.2 RESOURCE DISAGGREGATION

While the technology allows sharing of resources across workloads, workloads must still compete for the remaining resources, such as CPU and memory. Consequently,

workloads may still need to wait to access the resource due to the memory or CPU they need is being used by someone else. Thus, the resource remains unused. However, this last limitation is not because of the inability to share certain resources but rather because the shareable resource is physically attached to a single machine. If this resource could be hot-plugged and de-plugged from/to any other host, conflicts on non-shareable resources can get resolved. Here we are talking about resource disaggregation. Instead of physically attaching the resources to a single machine, we have a pool of resources. Computing nodes will then get the resources on-demand from this pool. When a computing node finishes using the disaggregated resource, they de-attach it, and it becomes usable by other nodes.

NVMeOF is a technology that does precisely that. NVMeOF [84] is an emerging network protocol used to communicate nodes with NVMe devices over a networking fabric. The architecture of NVMeOF allows scaling to large numbers of devices. It supports a range of different network fabrics, usually through RDMA , to eliminate middle software layers and provide very low latency.

Disaggregating NVMe over the network with NVMeOF allows sharing or aggregating the resource across many nodes, which helps to scale up and improve the efficiency of workloads. In this contribution, we analyze SMUFIN[7], which was introduced in chapter 2.5.

## 3.3 CHARACTERIZING RESOURCE SHARING AND DISAGGREGATION ON SMUFIN

In a continuous need to deal with increasingly larger amounts of data, genomics workloads are quickly adapting, and NVM technologies have become widely used as a key component in the memory-storage hierarchy. This section explores how disaggregating NVM might have an impact on genomics workloads, and in particular SMUFIN.

### 3.3.1 *Experimental Environment*

The experiments are conducted in an environment as depicted in figure 3.1.

SMUFIN uses the NVMe drives as a memory extension over fabric to store temporary data structures required to accelerate the computation. As the drives are dual-controller, two NVMe devices – of half their physical size – are exposed by the system for each physical device. In order to expose a single NVMe consisting of its two controllers or unify several NVMe devices, Intel RST [34] is used. Intel RST composes a RAID0 of the

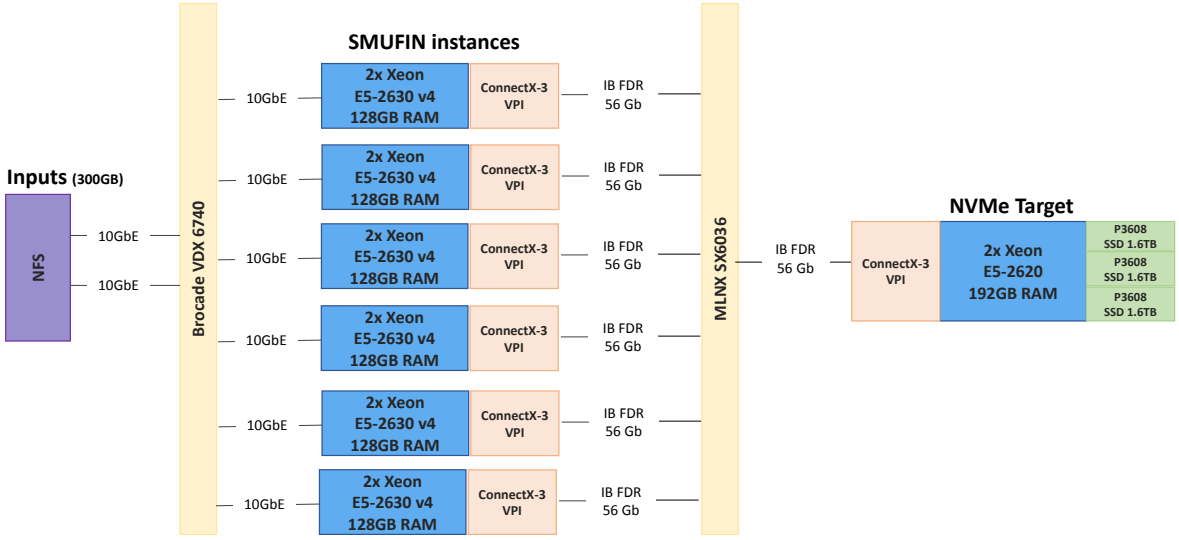



Figure 3.1: Experiments environment

controllers, which becomes exposed over fabric as a single NVMe card using NVMeOF. Mellanox OFED 4.0-2.0.0.1 drivers were used for the InfiniBand HCA adapters. The drivers included modules for NVMeOF, both the target and the client. Kernel 4.8.0-39 was used under Ubuntu server 16.10 operating system in all nodes.

SMUFIN on its merge stage was used. This stage is explained in section 2.5. The stage is indicated for our scenario as it is bandwidth-intensive on the NVMe device. In the following evaluations, each SMUFIN instance reads and processes a sample DNA input (+300GB) from an Network File System (NFS) shared storage, while the shared NVMe devices are used as memory extension for temporary data and final output. SMUFIN has been implemented to maximize sequential writes to the devices. This behavior has been verified by analyzing its access pattern. A block trace sample of requested blocks to the device was generated using Linux's *blktrace*[4], and the trace was then fed to the algorithm provided [18] to calculate the percentage of sequential write accesses. This method identified 88% of sequential writes after adapting the algorithm to consider accesses in which the final address matched the initial address of many immediately following requests, thus accounting for file appends. This behavior is optimal as its bandwidth-demanding and allows us to analyze the behavior of multiple SMUFIN instances over a single NVMe.

### 3.3.2 *Disaggregation benefits*

First we want to show how disaggregation can be leveraged to avoid interference of the host's resources (for instance, memory or CPU). For this, we run up to 5 instances

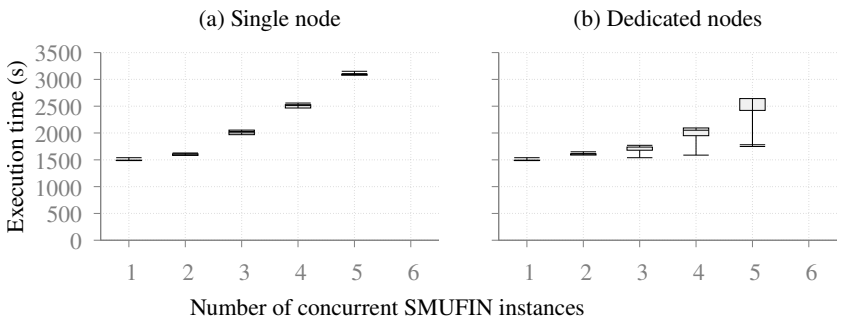



Figure 3.2: Boxplot of the execution time of dedicated nodes versus single node runs, running up to 5 SMUFIN instances using NVMeOF

concurrently on the same node. Each instance processes the same dataset, generating ≈150GB, with an average use of bandwidth of 477MB/s per SMUFIN instance. The NVMe device is capable of handling 2GB/s bandwidth under a sequential write pattern, as is the SMUFIN scenario. Then we repeat the experiment but each of the instances runs on a dedicated node. Each of those dedicated nodes shares a portion of the same NVMe, which is disaggregated via NVMeOF. The results are in figure 3.2. The figure shows how there is not a significant degradation when running a single or two instances in either case. Then we can show individually, in each case, workloads' performance starts degrading exponentially according to the level of concurrency. But we can see how the degradation becomes larger and grows quicker when sharing the same computing node than when each instance runs on a dedicated node. As a result, instances do not need to share memory and CPU, but only the NVMe device.

### 3.3.3 *Local resources interference*

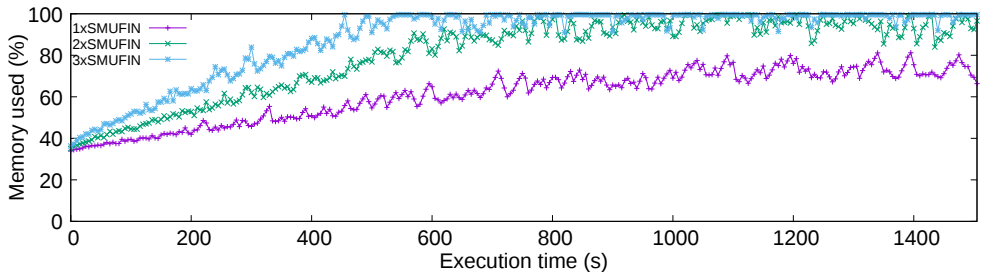



Figure 3.3: Memory usage on 1x,2x,3xSMUFIN scenarios using direct-attached NVMe on a period of 1500 seconds

.

To verify this latter hypothesis, we run up to three SMUFIN instances within the same node using a direct-attached NVMe device and monitor the memory usage. The results are shown in figure 3.3. The figure shows the memory usage when running up to three SMUFIN instances. The time interval analyzed is of 1500 seconds, which is the average time taken by a SMUFIN instance. Analyzing these results, the host's memory can handle all the intermediate data generated by SMUFIN for up to two instances. The NVMe becomes only used to output final data. However, the memory becomes a bottleneck with three instances, and intermediate data not fitting in memory gets flushed to the NVMe device more frequently. Is in this scenario when degradation is observed and performance comparison against NVMeOF is worse. Thus, confirming our hypothesis.

### 3.3.4 *Performance of NVMeOF*

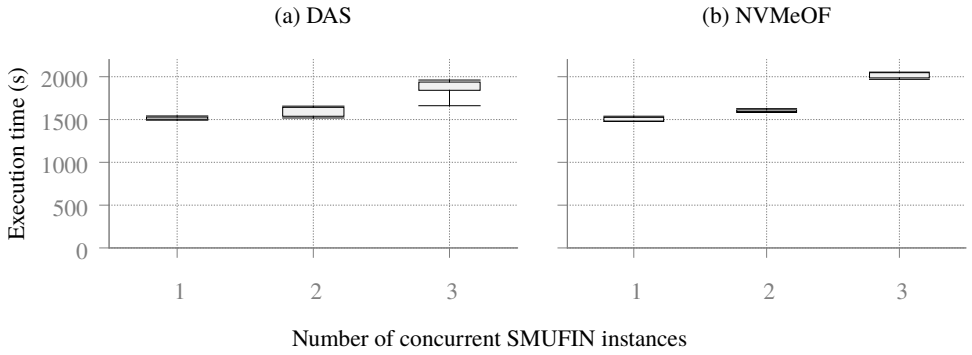



Figure 3.4: Boxplot of the execution time of Direct-Attached Storage (DAS) and NVMeOF when running 1x, 2x and 3x SMUFIN instances on the same node

The question mark would be whether NVMeOF can degrade SMUFIN performance. In such case, potential benefits of disaggregation could disappear. There is literature regards this question in [30]. They did not show any significant degradation when compared to *local* directly-attached storage. Nonetheless, we performed our experiments running up to 3 instances of SMUFIN in the same node: against a directly-attached NVMe device and against NVMeOF. Figure 3.4 shows the average execution time and deviation after repeating the executions six times. As it can be observed, when running one and two instances on local storage ( 3.4a) there is no performance degradation when disaggregating NVMe over fabrics ( 3.4b). However, when running three concurrent instances, there is a significant degradation of 6%, matching previous experiments. As already confirmed, it is due to memory-sharing. As observed, degradation grows exponentially from this point.

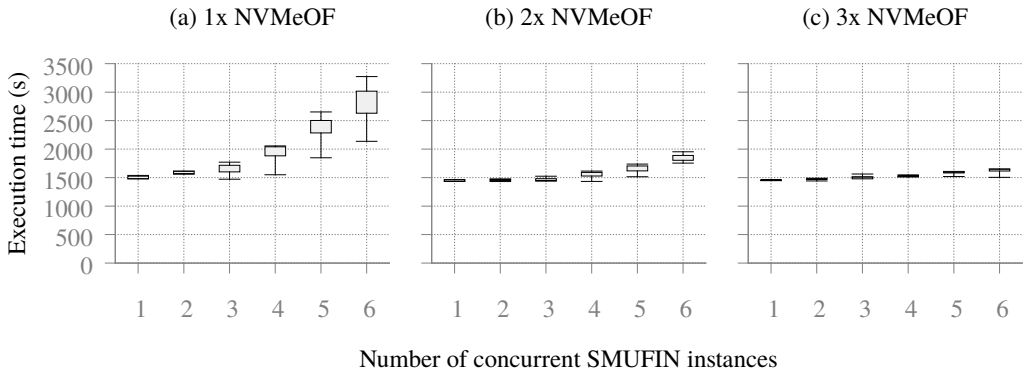


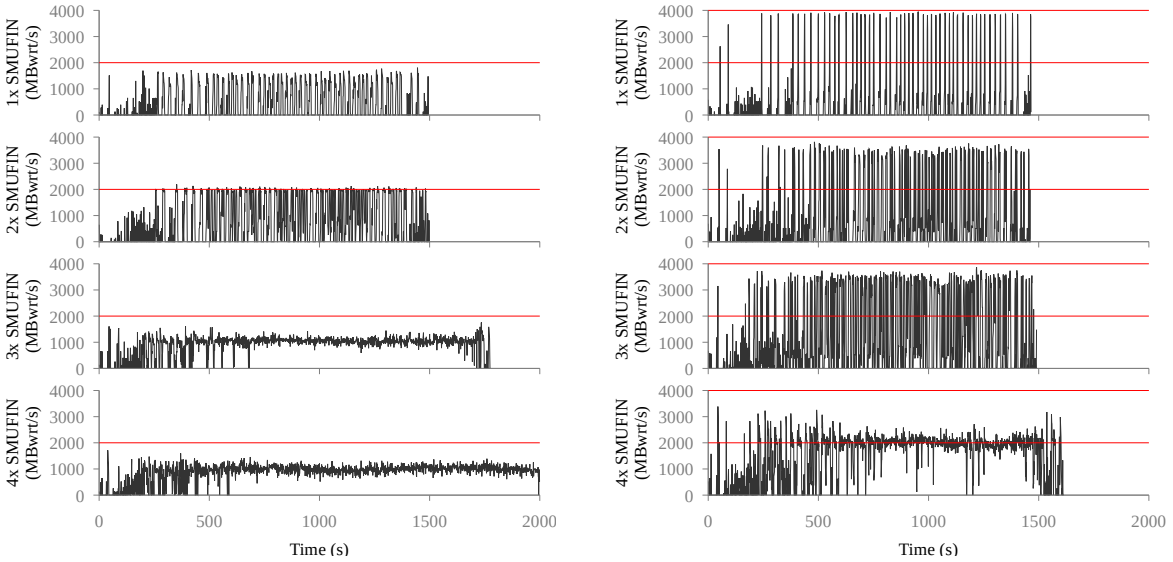

Figure 3.5: Boxplots showing how execution time evolves when running multiple SMUFIN instances: sharing a single device (1xNVMe) or sharing on composed nodes (2xNVMe, 3xNVMe)

## 3.4 RESOURCE COMPOSITION

When multiple workloads share resources and compete for its usage, their execution time compared to a dedicated execution in isolation degrades when a threshold is reached. As stated at the beginning, there is yet another benefit of Intel RST: resource composition. When composing NVMe altogether, their bandwidth and capacity are aggregated and exposed as a single fat device. This raises the question whether we can take advantage of it to move the degradation threshold further and be able to share a device with more workloads. We perform a new experiment where we run up to six concurrent instances, all of them using partitions from the same set of NVMe devices and running on separate nodes to avoid the detected interference.

Figure 3.5 depicts the boxplot of individual execution times under different number of resources composed, along with its quartiles, median, and standard deviation. In (a) only one NVMe is used. We have previously seen that running three instances separately against a single device does not degrade as significantly as running under the same node. However, performance degradation is still experienced when a certain resource-sharing threshold is reached due to the shared memory.

On the other hand, (b) and (c) shows the results when using a composition of two and three devices respectively. Under composition, profiling data shows that the Intel RST driver balances the bandwidth evenly through all composed devices. It is also observed that used bandwidth scales linearly with the number of devices. Hence under 2 and 3-compositions, 4GB/s and 6GB/s of sequential write speed can be reached respectively (each individual drive provides 2GB/s). Through composition, performance degradation can be mitigated. Compositions of two and three NVMe exposed as a single target to clients increases the bare-performance as a composition aggregates the total available bandwidth. In the 2-composition scenario 3.5b, up to 3




(a) 1x NVMe (b) 2x NVMe

Figure 3.6: Bandwidth measured from the NVMe pool server for 1x, 2x, 3x, and 4x instances of SMUFIN

sharing workloads obtain the same performance as if running alone in a single NVMe. The level of concurrency can be increased without introducing significant degradation using a composition of 3 NVMe devices (figure 3.5c), being able to have six sharing workloads with a similar performance as when running alone in a single device. Thus, workloads indeed benefit resource composition. However, in all scenarios, performance degradation still occurs on reaching a certain threshold, a threshold that is further away as more devices are used. Under 2-NVMe compositions, it is at four workloads, whereas on the 3-composition, the tendency is observed at six instances. This shows how resource composition can be leveraged to increase resource-sharing capacities.

### 3.4.1 *Bandwidth Bottleneck*

We observed performance degradation when a certain sharing ratio of resources is reached. Despite composition increases this threshold, degradation still occurs regardless of composition. The memory bottleneck was removed, as instances are running on dedicated nodes. The network bandwidth does not impose a bottleneck as per our experimentation environment; we are providing up to 56Gb/s, bigger than the maximum 6GB/s a composition of 3 NVMe devices can deliver. Therefore we must analyze the target NVMe devices' bandwidth to try to find the bottleneck.

Figures 3.6a and 3.6b show the NVMe bandwidth over time for experiments running up to four concurrent SMUFIN instances in the single-resource and the 2-composed resource configuration. The solid horizontal lines indicate the maximum bandwidth for sequential write that the resources can provide (2GB/s in single-resource configuration, 4GB/s for the composed scenario).

From the figures, two essential characteristics can be observer in both scenarios: (1) the bandwidth observed from the NVMe perspective is steadier; (2) the bandwidth that the NVMe device is capable of delivering is reduced as more concurrent instances are added. Running a single instance, the the full bandwidth of the combined NVMe can be used with bursts at a maximum 4 GB/s. However, as more concurrent executions are added, these bursts make use of less bandwidth until reaching saturation levels, decreasing significantly. Thus it becomes clear that degradation when using dedicated nodes happen from the NVMe becoming a bottleneck, and that composing resources increases available bandwidth and therefore allows for a bigger sharing ratio.

## 3.5 TOWARDS EFFICIENT ORCHESTRATION OF SHARED AND COMPOSED RESOURCES

Previous sections have shown how NVMe disaggregation provides new ways to use resources through resource sharing and composition. However, its behavior is not obvious a priori: heavy resource sharing may have a negative impact on performance, whereas composition may help increase sharing ratios without degradation. Therefore, deciding whether to compose a resource or share it among many workloads is not a a trivial decision. With the help of workload characterization, platform orchestrator will be able to make more informed and smarter decisions.

In Figure 3.7 we present different orchestration policies that could be managed with our data. The figure shows our cluster running five concurrent instances of SMUFIN, and three different resource allocation strategies for the instances: a) sharing a single device, b) sharing two NVMe devices, and c) one instance-dedicated device and the remaining four instances on a shared NVM device. This example was run under the same setup as in section 3.3. When the SMUFIN instances use two composed devices (b) it leads to faster executions times than using a single device (a). However, when using a dedicated device to run a single instance and a shared device to run the remaining four (c), the dedicated-device does not grant that instance an improved performance compared to a fully shared scenario using both devices (b). Intuitively it might be believed that just sharing all the resources under composition is the obvious winning strategy. However, this approach does not consider arriving workloads might

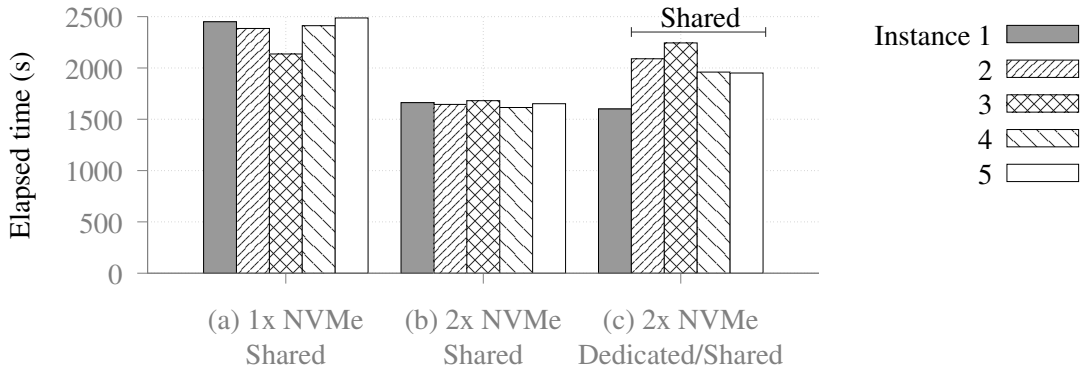



Figure 3.7: Execution time of 5 SMUFIN instances under different scenarios

have a time requirement for completion, and upon arrival of those workloads, if the resources are fully occupied serving others, the orchestrator will be unable to meet the requirement. Other concerns might be power consumption or total cost of ownership (as more resources, more expensive it becomes to run). Therefore, the strategy to follow must consider the trade-off between execution time and requirements of current and incoming workloads to maximize the granted quality of service, which in the case of genomics might be critical. The work on those policies becomes part of the second contribution of this thesis.

## 3.6 RELATED WORK

Genomics workloads and pipelines, in general, are a good fit for disaggregation, but previously to this work, application,ns haven't explored its large-scale exploitation. A number of different approaches to parallelize the whole genome analysis in HPC systems have been proposed in the literature [66], [39], and [47], but these tend simply adapt existing algorithms without considering or taking complete advantage of next-generation computing platforms.

Resource disaggregation is being increasingly studied in the literature. In [26], the authors examine the network requirements for disaggregating resources at rack- and data-center levels. Minimum requirements are measured in terms of network bandwidth and latency. Those requirements must be such so that a given set of applications doesn't experiment any performance degradation when disaggregation memory or other resources over the fabric. [41] implements NVMe disaggregation, but unlike the work presented in this chapter, the authors focus on a custom software layer to expose devices instead of using the NVMeOF standard.

## 3.7 CONCLUSIONS

This chapter has evaluated how resource sharing and composition benefits NVM-centric workloads in disaggregated data centers. This work takes SMUFIN, a real production workload in the field of computational genomics, leveraging remote NVMe devices as a memory extension. We have presented a comprehensive characterization of SMUFIN's resource consumption patterns. It is shown NVMe is utilized in a sequential write pattern. A performance comparison between directly-attached NVMe and NVMeOF is then presented and shown that as long the system's memory is capable of handling SMUFIN instances, there is no degradation. To increase concurrency disaggregating over fabrics allows sharing the same resource across multiple nodes running instances and the possibility of composition. Thus, we can handle more concurrent SMUFIN instances without individual degradation through disaggregation. On the other hand, reaching the resources' sharing ratio limit significantly degrades performance as the utilization of the available bandwidth diminishes, never reaching its maximum. Thus the NVMe becomes the bottleneck.

Finally, we have explained how the results of this characterization could be used to implement data-center scheduling policies to maximize efficiency in terms of Quality of Service (QoS). QoS could be understood in terms of execution time, so all workloads should be completing their executions within a certain requested time frame. The work on those policies is explored later in this thesis.

## 3.8 PUBLICATIONS

The contents of this contribution are summarized in the publication:
[9]Aaron Call, Jorda Polo, David Carrera, Francesc Guim, and Sujoy Sen. "Disaggregating Non-Volatile Memory for Throughput-Oriented Genomics Workloads." In: *Euro-Par 2018 International Workshops, Revised Selected Papers* (Turin Italy). Jan. 2019, pp. 613–625. ISBN: 978-3-030-10548-8. DOI: 10.1007/978- 3- 030- 10549-5_48.

# 4

# WORKLOAD-AWARE PLACEMENT FOR NVME-BASED DISAGGREGATED DATA CENTERS

In this chapter, the knowledge of chapter 3 is gathered and transformed into a model about the SMUFIN workload performance under disaggregation. With this knowledge, we simulate the environment of an actual data center with a mix of workloads arriving in it, among them SMUFIN workloads. An orchestrator is built, and a couple of policies are proposed. Those policies are workload-aware in the sense that it is aware of the workloads' performance under resource disaggregation. This contribution then shows the benefits of proper orchestration of disaggregated NVMe. Disaggregation brings the flexibility of assigning or re-assigning a resource to the nodes who need it. It allows meeting workloads SLA under tight conditions much better than workload-unaware policies.

The achievements of the contribution are:

- Two model-aware placement policies. Maximize composition and minimize fragmentation. The former leverages resources composition and sharing, while the latter intends to minimize the effects of fragmentation.
- A disaggregation-aware scheduling policy that decides which of the placement policies fits best under the current data center load.
- A comparative analysis of the benefits of resource disaggregation versus physically-attached resources.

## 4.1 INTRODUCTION

As seen in 3 resource disaggregation allows sharing resources without performance degradation up to a threshold. From that threshold, the degradation will rise exponentially. However, that knowledge is workload-specific. That is, not all workloads behave equally under sharing. We have also observed that some workloads benefit

from resource composition by increasing this sharing threshold further. In an actual data center however we will have a mix of workloads, some candidates to leverage resource disaggregation and some will not. Hence, deciding how to allocate resources under such a mix of workloads is a non-trivial task which result has significant impact on the overall data center performance. Therefore orchestrators of the disaggregated center need workload-aware placement policies to leverage that knowledge and make the best possible resource allocations to maximize the SLA fulfilled.

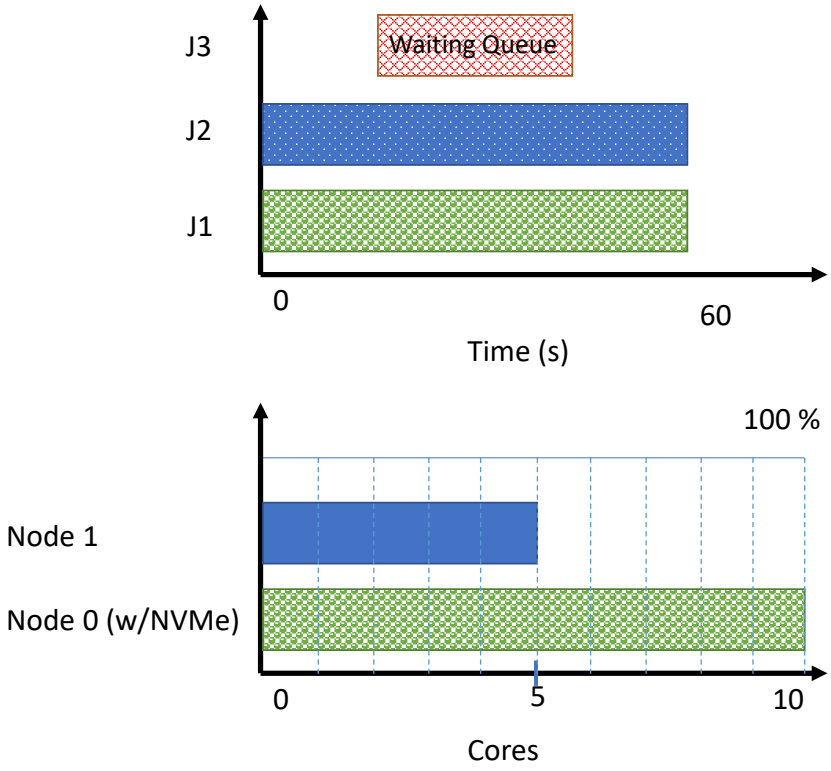
To study the impact of resource disaggregation on a data center, we focus on an architecture where several nodes can access a set of disaggregated resources. This chapter only considers nodes with a set of CPUs, which all workloads require, and NVMe as the main disaggregated resource.

To demonstrate the importance of disaggregation-aware workload placement policies, we will consider a simple yet illustrative example with two nodes and three workloads, as shown in Figure 4.1. Workload J1 requires 100% of the cores of one node (pictured green), J2 requires 50% of the cores (blue), and J3 requires one core and NVMe device (red). We consider two scenarios. In the first scenario, as shown in Figure 4.1a, only one node has physically-attached NVMe. When J1 and J2 arrive, they are placed in nodes 0 and 1, respectively. When J3 arrives, it only has enough cores to run in node 1. However node 1 does not have an NVMe, so J3 gets into the NVMe queue, waiting for J1 to complete and be able to run on node 0, where there is an NVMe available. In the second scenario, in Figure 4.1b, the NVMe is disaggregated and can be pooled. So either node can attach and share the resource on-demand. In this case, J3 can be placed in node 1 due it pools the disaggregated NVMe on-demand from the remote pool. Thus, disaggregation leads to more efficient use of available resources and quicker completion of jobs.

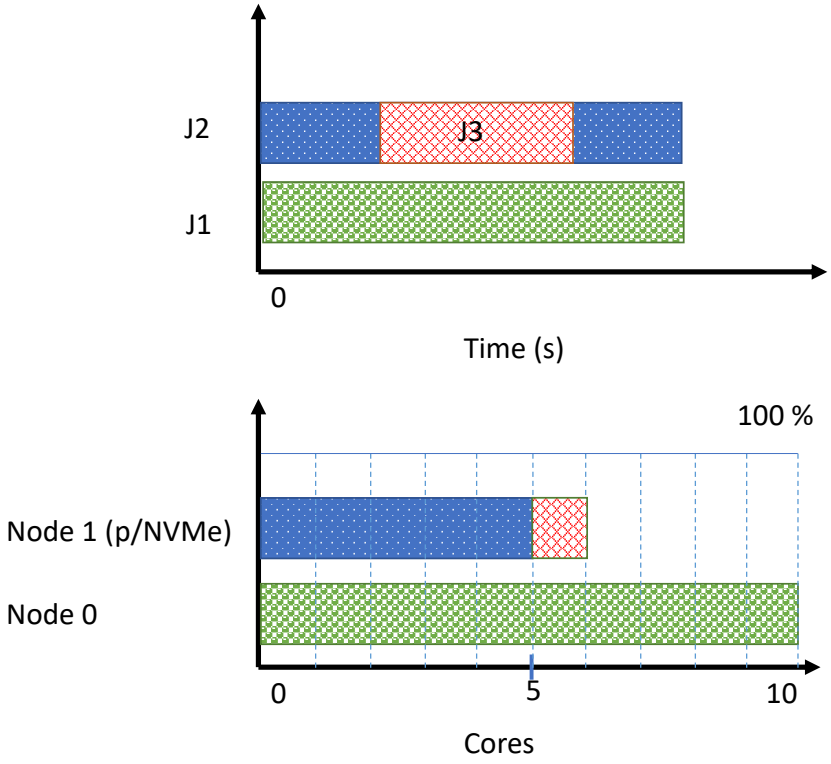
While this is a simple example of basic placement strategies, similar situations arise in many placement policies.

When the behavior of workloads under disaggregation is known, it is possible to develop policies to take maximum advantage of the resources.

In this contribution, we use the results of chapter 3 to build a performance model for the SMUFIN workload, in which assigning more resources than the strictly required leads to performance improvements. Using this model, we propose strategies to address the challenges of managing pools of disaggregated resources. This contribution answers questions such as when and where to attach/detach resources, how many workloads should be allowed to run using the same device, and whether or not compositions should be made.




(a) Physically attached NVMe in node 0. Job J3 (red) has to wait until node 0 is available.




(b) Disaggregated NVMe in node 0. Job J3 (red) is executed in node 1 with NVMe pooled and remotely attached.

Figure 4.1: Timeline showing the execution of three jobs in two nodes with physically-attached NVMe (top) and disaggregated NVMe (bottom)

## 4.2 DISAGGREGATION CHALLENGES

To study the impact of resource disaggregation on a data center, we focus on an architecture where several nodes can access a set of disaggregated resources. We only consider nodes with a set of CPUs, which all workloads require, and NVMe as the main disaggregated resource.

Figure 4.2 depicts the described architecture. We have a set of nodes, each of which has some CPUs (in the depicted example, 20), and a pool of NVMe resources with given bandwidth and capacity. The architecture allows splitting into many pools of resources. However, resources from different pools can't be composed into a single one. This chapter bases the infrastructure on NVMeOF.

Our infrastructure is different from traditional remote network storage. Traditional storage resource pooling usually involves either a shared filesystem over the network, or shared volumes made of storage devices accessed via iSCSI or similar protocols. The approach used in this thesis is closer to the latter, but still differs in a number of ways, including: 1) performance, in particular, lower latency leading to more use-cases based on using NVMe directly as a block device or memory extension; 2) dynamicity and frequency at which devices can be expected to be reconfigured, attached, and detached; and 3) flexibility of resource sharing and composition. All of these lead to significant changes in terms of resource management. In particular, the proposed disaggregated data center allows for full dynamic reconfiguration. It is expected to be attaching and detaching devices with high frequency, depending on the demand. This malleability is not expected on traditional storage systems, so current tools cannot properly manage this kind of infrastructure yet. Moreover, note that accessing the device as a block device enables HPC workloads to use this disaggregated environment. Traditional cloud systems are file-based, which limits application for many HPC workloads.

### 4.2.1 *Fragmentation*

In addition to the challenges of resource sharing and composition described in chapter 3, in this chapter we consider the challenge of fragmentation as well. Fragmentation occurs inherently to disaggregation, and it is often impossible to avoid. Which one out of the pool should be provided whenever a resource is requested? A resource already attached to some other node, a composition of resources, or a resource not currently pooled to any other node? Proposed policies will explore different alternatives. Depending on the situation, it might lead to *fragmentation*. Figure 4.1 shows this situation. In scenario 4.1a, when J1 and J2 are scheduled, node 0 has its

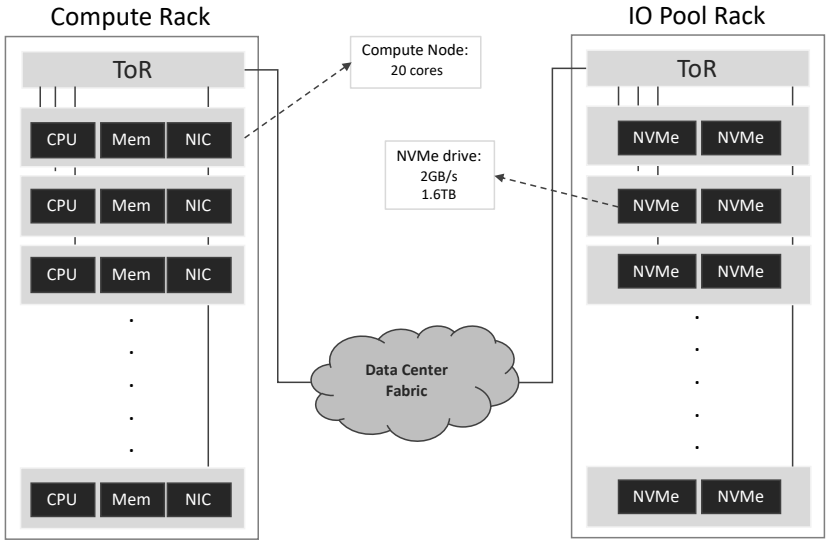



Figure 4.2: Overview of data center with a disaggregated IO architecture, with pooled NVMe devices over the fabric

cores fully utilized, while its NVMe resources are completely free. Thus, the system has unused areas (NVMe) that are at the same time inaccessible until J1 frees its cores. It has "split" half of the resources, fragmenting the system. Intermediate cases are relevant as well. Following the same situation, when J3 is finally scheduled, J3 utilizes only 10% of the cores. Imagine the usage of the NVMe it does is of 80%. It would be using most of the NVMe, however, at the same time leaving 90% of the available cores free. This situation implies that NVMe-dependent workloads will hardly be allocated in this node (in most scenarios they will require more than 20% of the device). Only compute-intensive workloads will likely use the node. In this situation, if compute-intensive workloads presence is low, the cores will be unused for long periods. Thus, a better scenario would have been that most of the cores were also used, or the NVMe was less utilized. However, this is not a situation the orchestrator can control, as it is frequent that workloads only require resources partially. Thus, fragmentation is of utmost importance when disaggregating; however, it is inherent and often can't be avoided but only minimized. Our proposed policies and following evaluation will explore the usefulness of minimizing it.

## 4.3 STRATEGIES FOR SDI ORCHESTRATION

Based on the described challenges, two placement policies are proposed: one exploiting resource sharing and composition, and a second one reducing system fragmentation. Finally, a disaggregation-aware scheduler is proposed to decide how to switch between

both policies. The policies described focus on a homogeneous set of resources (i.e, all NVMe have the same characteristics). However, a heterogeneous environment could be managed using heuristics whenever a group of unused resources meeting the requirements (NVMe capacity and bandwidth) is needed. Nevertheless, the impact of such modification in the policies is low, as the strategy doesn't change. Hence the focus is on homogeneous characteristics.

As stated previously, our policy design assumes knowledge of the workloads' impact under resource sharing and composition, this is the critical difference between our proposal and traditional data centers. Traditional data centers assume workloads are static, and they do not possibly benefit from having more resources than requested. In this chapter we consider it is possible to know about some workloads' behavior under resource sharing and composition. Thus, if a known workload may take advantage of it, our placement policies will be aware and act accordingly. With those models, it is possible to develop the right placement policies to use both properties. If a workload is unknown, the proposed policies will work by calculating its completion time based on the provided execution time.

### 4.3.1 *Maximize Composition Placement Policy*

Some workloads benefit from having more NVMe bandwidth than available. When this over-provisioning occurs, they might improve their performance. On the other hand, the sharing ratio of resources without performance degradation also increases with the higher availability of bandwidth and capacity. Thus, combining resources into large compositions helps the system's overall performance for this kind of workload. This behavior can be observed in our previous experiments [9]. Consequently, the idea behind the first proposed policy is, when unused resources must be found and allocated to workloads, to compose many resources into a single one, over-provisioning, as long as the workload will benefit from it. The policy is described in pseudo-code on algorithm 1.

Lines 2-9 iterate over compositions in use (resources already allocated and being used by other workloads). As long as the assignment meets the deadline and SLA (NVMe bandwidth, capacity, cores), the composition becomes a fitting candidate. To decide among candidates, the time to live of the composition if the placement would be made is calculated (line 5), and as a tie-breaker parameter, the fitness is computed as described in equation 4.1 (line 6). Let $bw_c$ be the bandwidth available in composition $c$, $bw_j$ be the bandwidth demand for job $j$, $sto_c$ be the storage capacity of $c$ and $sto_j$ be the storage capacity demand for job $j$. Then, $fitness$ represents how much bandwidth

**Algorithm 1** Maximize composition placement

```
1: procedure MAXIMIZE COMPOSITION PLACEMENT(job, deadline)
2:     parameters = {bandwidth, capacity, cores}
3:     for all c in compositionsInUse() do
4:         if meetCompletionSLA(c, job, deadline) then
5:             ttl = TTL(job) − TTL(c)
6:             f = fitness(c,job)
7:             insertSortedAscending(candidateList, ttl, f, c)
8:         end if
9:     end for
10:    if ¬empty(candidateList) then
11:        composition = first(candidateList)
12:        assignResources(job, composition, assignedCoresNode(composition))
13:    else
14:        request = calculateRequestForMinimalTime(job,parameters)
15:        composition = findFreeResources(request)
16:        coresNode = FirstFitFindCores(job)
17:        assignResources(job, composition,coresNode)
18:    end if
19: end procedure
20: procedure CALCULATEREQUESTFORMINIMALTIME(job, parameters)
21:    prevTime = baseTime(job)
22:    request = Set()
23:    for all p in parameters do
24:        for i = 0; i < maxJobParameter(job, p); i+ = 1 do
25:            curTime = workloadPerformanceParameter(job, p, i)
26:            if prevTime > curTime then
27:                prevTime = curTime
28:            else
29:                request.add(p,i)
30:                continue
31:            end if
32:        end for
33:    end for
       return request
34: end procedure
```

and capacity would be left in the composition after satisfying job *job* demands. Both parameters ($fitness$ and $composition$) are ordered in ascending order (line 7). The policy attempts to free resources quickly, and as a secondary priority to compress them as much as possible. If placing the workload would significantly increase the composition's lifetime, it rather creates a new composition.

$$fitness = (bw_c - bw_j) + (sto_c - sto_j) \tag{4.1}$$

Lines 10-12 allocate the best-fit candidate composition (already existing, if any). Otherwise, it is necessary to search among free resources (from the pool of NVMe) and create a new composition. Lines 14-17 show the latter. The priority is to make compositions with more resources than needed, that is, over-provisioning. However, over-provision is made as long as workloads' performance improves. If making a composition of three or four resources makes no difference, the smallest one is chosen. The achieved performance is estimated using the performance models. Thus, becoming a model-aware policy. The calculus for this is made on lines 20-33. Lines 24-30 check if over-provisioning by each parameter would improve workloads' performance with respect to smaller ones. The last parameter value is chosen once the composition size stops improving performance (line 29). Line 24 also limits some parameters. A model may indicate workloads will not improve their performance past a certain bandwidth or capacity of the resources. Hence the policy does not attempt to increase the parameter beyond it. As an example, as described, storage-based workloads will not achieve any performance upgrade past their base capacity and bandwidth. Thus, for this kind of workload, the parameters chosen will be the minimum workloads' request. Once the parameters' values are elected, the policy looks into the pool of free resources for the minimal set of resources fulfilling the requirements (line 15). This line's algorithm is not described as finding the optimal set in a homogeneous set of resources is trivial. Last, a node with enough available cores is found using a first-fit strategy (line 16). Finally, the set of free resources selected turns into a new composition and is attached to the node with enough cores previously found (line 17).

### 4.3.2 *Minimize Fragmentation Placement Policy*

This policy aims to minimize fragmentation, based on situations like shown in the introductory example in figure 4.1 and explained in section 4.2.1. To minimize fragmentation, we need to consider the allocation carefully. On which node the job will run in, and which NVMe resource will it use. If we are not careful enough, the remaining free

space resulting from the allocation might be such that any incoming workload can use it. This is what we previously defined as fragmentation. As fragmentation is inherent in the system, we develop this policy to try to minimize it, making better use of the available space which can lead to better performance of the system. For it, we require a metric that defines how much fragmentation an allocation introduces. In this regard we make two considerations: on the one hand the internal fragmentation resulting to allocating a job to a specific composition. If the allocation makes a composition fully utilized, this particular composition will not have internal fragmentation. On the other hand, how many free cores will remain in the node where the composition becomes attached to. Again, if the node we decide to attach the composition ends up having all its cores utilized, it will be a better placement than one leaving spare resources. With these considerations in mind, the ratio of fragmentation $\alpha$ between the NVMe bandwidth and capacity requested, plus the remaining cores after scheduling the workload, is estimated. This computation is made according to equation 4.2. Let $bw_c$ be the unallocated bandwidth available in composition $c$, $bw_j$ be the bandwidth demand for job $j$, $sto_c$ be the unallocated storage capacity for composition $c$, $sto_j$ be the storage capacity demand for job $j$, $core_n$ be the unallocated cores in node $n$ and $core_j$ be the core count demand for job $j$. Then $\alpha$ is defined as:

$$\alpha = \frac{1 - (\frac{bw_j}{bw_c} + \frac{sto_j}{sto_c})}{\frac{core_j}{core_n}} \tag{4.2}$$

The numerator calculates the amount of a composition utilization after the allocation of job $j$. This is subtracted by 1 to calculate the remaining spare space on composition $c$. The denominator computes the utilization of cores on the node if $j$ is run on node $n$. The reasoning behind setting this division is the following: given a certain fixed percentage of cores to use from the available cores in the node, the lower the attached resource's left-over, the lower the fragmentation. If cores usage is fixed, minimizing fragmentation is simply making the most use of the attached resource (we want low left-over). So the numerator computes this free space on the pooled resource. We do have two parameters on NVMe, bandwidth and capacity. As different workloads might want more bandwidth or capacity, we can't prioritize either of them. So we consider both equally important and add them up. For the denominator the reasoning is similar: given a fixed spare space on the pooled resource, the higher the cores used, the lower the fragmentation. Indeed, if resource's available space is fixed, the only way to minimize fragmentation is to make the most use of the cores. We can then conclude that the higher the value of alpha, the higher fragmentation of the system.

This equation serves for deciding among the already attached and in use resources. However, when there is a need to make a new attachment among free resources, a new formula is needed. The goal of this chapter is to orchestrate disaggregated resources, and is not possible to fragment (by definition) the pool of disaggregated resources. Thus, finding a set of free resources meeting a workload's requirements is trivial. However, the available cores are not disaggregated but physically attached to actual physical nodes. Thus, it is possible to introduce cores fragmentation when finding cores to execute workloads on. Thus, when deciding which node to attach our resources to, we attempt to maximize the utilization of the physically-attached cores. To achieve this goal, we compute cores' slack using equation 4.3. This equation is similar to the concept of resource's slack provided by [36]. The equation computes the percentage of free cores remaining in the node, thus becoming our slack. Let $core_n$ be the unallocated cores in the node, and $tcore_n$ the total cores in the node, $\beta$ computes the percentage of utilized cores (as we made the reverse). Thus, the lower $\beta$, more cores are available in the node. Therefore, the resource strategy will place workloads first on the most under-utilized nodes, progressively utilizing all of the nodes as workloads are placed. This proposal helps to minimize cores fragmentation for the scenario of a large amount of compute-intensive workloads present in the system. This situation belongs to the second consideration in 4.2.1.

$$\beta = 1 - \frac{core_n}{tcore_n} \tag{4.3}$$

Algorithm 2 shows the pseudo-code for the policy:

Lines 2-15 are very similar to maximize composition policy. However, in this case, the $\alpha$ parameter is used to determine the fitness of elements and reduce fragmentation. In case there are no candidate compositions, and free resources need to be used, as described first available resources meeting the requirements are found (line 13) following the same strategy as in maximizing composition. Finally, a node with available cores is found. This search is done according to the algorithm in lines 18-26, and it selects the node minimizing fragmentation of cores following equation 4.3 (lines 21-22).

### 4.3.3 *Disaggregation-aware Scheduling Policy*

This thesis focuses on the QoS of workloads in terms of deadlines, deciding between high-priority workloads and regular-priority workloads. For this reason, our proposed disaggregation-aware policy takes Earliest Deadline First (EDF) as a base scheduling

**Algorithm 2** Minimize fragmentation placement

```
1: procedure Min-Fragmentation Placement(job, deadline)
2:     parameters = bandwidth,capacity,cores
3:     for all c in compositionsInUse() do
4:         if meetCompletionSLA(c, job, deadline) then
5:             α = α(job,c)
6:             insertSortedAscendent(candidateList, α)
7:         end if
8:     end for
9:     if ¬empty(candidateList) then
10:        placement = first(candidateList)
11:        assignResources(job, placement)
12:    else
13:        composition = findFreeResources(job,parameters)
14:        coresNode = minFragCoresNode(cores)
15:        assignResources(job, composition, coresNode)
16:    end if
17: end procedure
18: procedure minFragCoresNode(cores)
19:    for all n in nodes do
20:        if freeCores(n) >= cores then
21:            β = β(n)
22:            insertSortedAscendent(candidateList, β)
23:        end if
24:    end for
25:    if ¬empty(candidateList) then
26:        return first(candidateList)
27:    else
28:        return false
29:    end if
30: end procedure
```

policy. This is one of the fundamental scheduling policies suitable for scenarios where deadlines are the priority, enabling differentiation between priority and non-priority workloads. We run a set of experiments choosing one or the another strategy throughout the simulation in the same scenarios as the following evaluation of this chapter. During those experiments we concluded that whilst minimizing fragmentation is a good strategy if there is high presence of non-IO bound workloads, it does not behave that well when those workloads are a majority. Likewise with maximize composition, it behaves well when IO-bound workloads are the most, however it worsens when this situation does not happen. For this reason, and as stated at the beginning of this chapter, our proposed policy does switch between both proposed placement policies dynamically. The conditions that must be meet to apply one or the other were obtained out of the aforementioned experiments. We define (NVMe) bandwidth and capacity load factors as the relationship between demanded and total available bandwidth and capacity, respectively. Let $LF_{bw}$ be the bandwidth load factor and $LF_{cap}$ the capacity load factor. Our scheduler applies *Maximize Composition* (Max. Comp.) or *Minimize Fragmentation* (Min. Frag.) according to function 4.4.

$$\text{Policy} \equiv \begin{cases} \text{Max. Comp.} & LF_{bw} \leq 50\% \text{ and } LF_{cap} \leq 50\% \\ \text{Max. Comp.} & LF_{bw} \geq 70\% \text{ and } LF_{cap} \leq 70\% \\ \text{Min. Frag.} & \text{Otherwise} \end{cases} \tag{4.4}$$

## 4.4 EVALUATION

In this section, our proposed policies and disaggregated environment's performance are compared against a basic one. First, we describe the environment (infrastructure and workloads) used to perform the evaluation. It follows with a definition of some metrics that will help to define our concept of performance. Finally, experiments are made using those metrics. The experiments are split into two. On the one hand, resource disaggregation versus physically-attached infrastructures, and on the other hand, a comparison of our proposed strategy versus a basic strategy simulating a traditional data-center. On the latter, further experiments are made to assess the different metrics.

### 4.4.1 *Methodology*

We built a simulator to evaluate the impact of the proposed strategies. The simulator emulates the architecture described in figure 4.2. The simulator only considers NVMe

resources for disaggregation. We chose to build our simulator as our requirements were rather simple, and we considered the effort to be smaller than adapting a previously existing simulator for computing architectures to our architecture and needs. Moreover, to the best of our knowledge, no simulators are emulating disaggregated environments. The existing simulators are generally complex and emulate generic infrastructures. Our simulator code can be found online in [8].

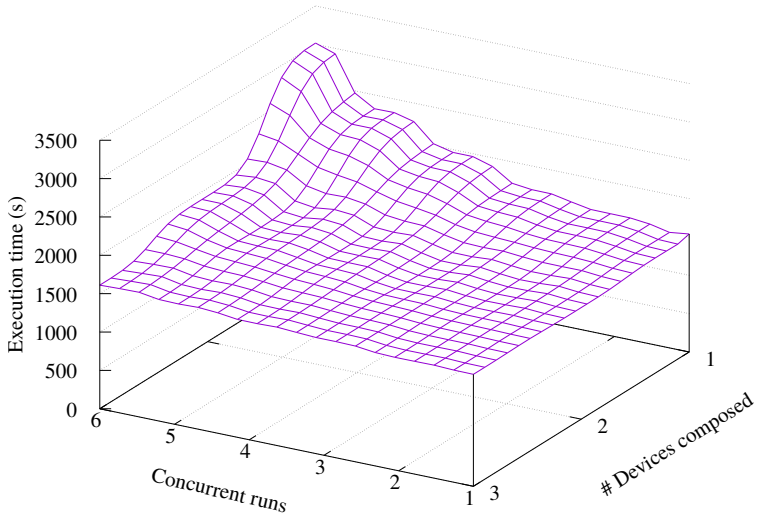



Figure 4.3: Execution times for a bandwidth-bound workload showing how execution time evolves when running multiple SMUFIN instances: sharing a single device (1xNVMe) or sharing on composed nodes (2xNVMe, 3xNVMe).

The simulated infrastructure comprises five nodes with 25 cores each, and a pool of 10 disaggregated NVMe devices, featuring a bandwidth of 2GB/s and 600GB of storage capacity. On the physically-attached infrastructure, only two of the five nodes have NVMe. Of those two, the first has attached six of them and the remaining four at the second.

To evaluate the impact of our strategies in a wider scenario, three types of workloads are established:

1. Bandwidth-bound: represents workloads that are sensitive to bandwidth, and so multiple concurrent workloads running in the same the device may have an impact on performance if bandwidth capacity is exceeded. We use our work on SMUFIN[58][7], to model the behavior of this workload and the results presented in [9], summarized in figure 4.3. The figure shows the execution times for real, SMUFIN runs on compositions of 1, 2, and 3 NVMe devices, with up to

Table 4.1: Workloads' requested resources

| Workload type | Base execution time | NVMe bandwidth | NVMe capacity | CPU cores |
|---|---|---|---|---|
| Bandwidth-bound | 1600s | 1800MB/s | 43GB | 6 |
| Capacity-bound | 800s | 160MB/s | 600GB | 6 |
| Compute-bound | 900s | N/A | N/A | 15 |

Table 4.2: Default simulation parameters on three scenarios

| | Bandwidth-bound (%) | Capacity-bound (%) | Compute-bound(%) | Non-priority deadline factor | deadline factor | High-priority workloads (%) | Number of nodes | Cores per node | NVMe bandwidth | NVMe capacity | Number NVMe on pool |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S1. High-bandwidth | 70% | 10% | 20% | 4 | 1.2 | 20% | 5 | 25 | 2 GB/s | 600GB | 10 |
| S2. High-capacity | 10% | 70% | 20% | | | | | | | | |
| S3. High-compute | 20% | 10% | 70% | | | | | | | | |

6 concurrent executions were sharing the same composition. This model shows how over-provisioning the workload may have some benefits both in terms of performance as well as workload's concurrence threshold increase without experiencing degradation. The model is fed to our placement policies.

2. Capacity-bound: represents workloads that have significant storage capacity requirements and relatively low bandwidth requirements. Unlike bandwidth-bound workloads, these do not perceive any performance degradation from sharing the device with other workloads, as long as the demanded capacity does not exceed the available capacity. A real-world example of such would be TPCx-IoT [20], a TPC benchmark that attempts to emulate an Edge Computing scenario. It provides the mentioned characteristics: it uses NVMe mostly for storage and performing random reads on the data. However, we do not use a model about its performance in the current evaluation for simplicity reasons.

3. Compute-bound workload: we emulate the situation where we have CPU-consuming workloads that do not require NVMe usage. We introduce this synthetic workload to emulate situations where workloads not using the NVMe might prevent other workloads from using them in non-disaggregated scenarios. It could be any compute-bound real-world workload, for example, mathematical computations of weather forecasting.

We do not possess a performance model for capacity-bound and compute-bound workloads. Thus, its performance does not degrade or improves modifying sharing or composition ratio, as long as the capacity and bandwidth requirements are met. However, notice that if we possessed it, placement policies would use it. Sharing and composition only impact bandwidth-bound workloads, for which, as mentioned, the SMUFIN model is used to estimate the performance in such situations. This scenario becomes the main target to evaluate in this chapter.

The considered workloads utilize NVMe (bandwidth and capacity) and cores. Workloads specify their base performance (assuming their SLA is met). Table 5.1 describes the resources used by each kind of workload.

To consider a wide spectrum of situations, three different distributions (from now on scenarios) of the workloads' are taken. Table 4.2 presents the distribution settings for each scenario. It also summarizes the default settings used. The scenarios are as follows: a high-bandwidth scenario (S1) with 70% bandwidth-bound, 10% capacity-bound, and 20% compute-bound workloads. A high-capacity scenario (S2) with 70% capacity-bound, 10% bandwidth-bound, and 20% compute-bound workloads. Finally, a high-compute scenario (S3) with 70% compute-bound, 20% bandwidth-bound, and 10% capacity-bound workloads.

The simulator generates the arrivals for the requested workloads within a requested timeframe. In our evaluations, we simulate 1500 workloads are arriving in a period of 3 days. The inter-arrival time of the workloads follows a Poisson process. Poisson is chosen as it introduces periods when the inter-arrivals are much shorter (emulating rush hours) and periods with larger inter-arrivals (regular hours).

Moreover, across all the workloads, a randomly selected 20% of them are considered high-priority and are assigned a tight deadline, whereas the rest have a relaxed deadline. Let $e_w$ be the base execution time of the workload, $\delta$ the deadline factor, and $a_w$ the arrival time of the workload, then the deadline is computed according to equation 4.5:

$$deadline_w = e_w\delta + a_w \tag{4.5}$$

The deadline factor $\delta$ defines the tightness of them. A relaxed deadline factor is 4 (400% the base execution time $e_w$). Later an evaluation is presented, modifying the factor for high-priority deadlines. The default high-priority deadline factor is 1.2 (120% of the base execution time $e_w$).

A Poisson distribution is commonly assumed for allocating resources into cloud data centers. Examples of this can be found in [40] where they solve the problem of assigning VMs into physical machines. Other examples are [57][16]. Other papers assume a 2-modulated markov process [77]. There is literature attempting to better model the arrival of workloads into the cloud [35]. During the work described in this chapter, we also modeled the 2-modulated Markov process as well as random and uniform distributions. All of them shared the pattern that the distribution increased or decreased the load factor quicker or slower for the same parameters. However, the results' conclusions were very similar to those we show in this section. Therefore, although certainly, the absolute numbers vary depending on the distribution, the findings do not. For this reason, and because the primary goal of this thesis was to assess whether disaggregation helps or not in the data center, we decided to take the most common distribution to date.

### 4.4.2 *Load factor*

A key element of analysis is the load factor of the system. An ideal load factor is calculated to evaluate the impact of this distribution of workloads. The load factor is described in this thesis as the ratio between resources requested and available. As we have three different kinds of resources (bandwidth, capacity, and cores), three load factors can be calculated. In this chapter, we put particular emphasis on the CPU load factor (cores). Let $c_s$ be the cores available in the system, $c_j$ the cores requested by a job, and $J$ the set of jobs currently active (arrived but not completed) on the system the CPU load factor is calculated on equation 4.6 as the quotient between $c_s$ and the sum $c_j$ of all jobs $J$ running the system.

$$LF_{cpu} = \frac{c_s}{\sum_{j \in J} c_j} \tag{4.6}$$

It is only necessary to replace cores with bandwidth or capacity to calculate bandwidth and capacity load factors. Notice, however, that the results of the equation are subject to the scheduling and placement policy. Depending on the policy, specific jobs will complete later or earlier. Thus, the set $J$ of jobs running on the system will vary, and so will the load factor. This makes it hard to compare the load factor between

strategies. For this reason, in order to calculate the load factor of the system, we use an artificial scheduler assuming an ideal infrastructure where all resources are aggregated into a single fat node. Thus, no fragmentation is possible. In this scheduler, whenever a workload arrives, if resources are available, it is run and completes after its base execution time. Otherwise, it must wait until some workload frees enough resources. With this formulation, we can calculate a load factor independent of the policy. Therefore given the distribution of workloads and a set of resources, our strategies will evaluate the same load factor.

As stated, to evaluate our strategies, we take the CPU load factor as our target load factor. Depending on the target, the performance of the strategies will vary. An over-saturated system with a very high load factor implies that it will be impossible to fit all jobs as soon as they arrive. Therefore deadlines will be inevitably missed regardless of the strategy. On the other hand, a very relaxed system with a low load factor implies that even the dumbest strategy will achieve good performance, as there are plenty of resources available. However, deciding which load factor is appropriate to evaluate is a challenging problem. For this reason, the three scenarios of workloads' distributions are evaluated for different targets. Notice, however, that for every distribution, to achieve the same load factor, the inter-arrivals time of workloads needs to be different ($\lambda$ parameter in our Poisson distribution), due that every topology of workloads has different resource requirements. Moreover, all metrics utilized are only measured once an ideal CPU load factor has reached 0.7, and the measurement window stops as soon as the latest workload arrives.

### 4.4.3 *Impact of disaggregation*

This evaluation compares our three scenarios on the following experiments:

1. Load factor and physically-attached versus disaggregated environment: evaluates how system saturation impacts the strategies. Its behavior is compared between physically-attached NVMe and disaggregated NVMe.
2. High-priority deadlines factor: evaluates the impact of pressure due to high-priority workloads.

As a comparison base, a First Fit policy is implemented with an EDF scheduler. As its name indicates, this policy performs the first resource allocation possible for a workload. Moreover, it assumes a traditional data center where over-provisioning is not possible, thus is not using our model, assuming any workload can share a resource neither benefits from resource composition. It is chosen as a comparison baseline due it does not consider any state of the system, as long as the allocation fulfills the

workloads' requirements. Thus, it allows us to understand the impact of considering the infrastructure's characteristics over not doing so. Notice our over-provisioning policy performs as well as the first fit. However, as it tries to over-provision, it allocates extra devices. Those additional devices also follow a first fit method. Therefore, we are comparing First Fit against "First Fit"-aware; aware of how workloads behave under disaggregation.

This chapter assesses the impact of disaggregation by comparing the infrastructure against an infrastructure with physically-attached resources. On the latter, only two of the five nodes have NVMe attached. The first with four and six on the second. The other three nodes do not have any NVMe attached. In tables 4.3, 4.4 and 4.5 we present the results for our strategies in both a disaggregated and a physically attached (PA) infrastructures. All the results are shown for different target CPU load factors in order to show a wide spectrum of situations. The metrics definitions shown in the tables are as follows:

- Target CPU load factor: is the average CPU load factor on an ideal infrastructure with no fragmentation.
- Observed load factor: real load factor in the system under the given strategy.
- Missed deadlines: percentage of workloads missing its deadline.
- Missed high-priority deadlines: percentage of high-priority workloads missing its deadline. The absolute value cannot be higher than workloads missing its deadline.
- NVMe usage: in percentage, average of NVMe usage across simulation.
- Avg. Waiting time: average waiting time of workloads. The waiting time is defined as the time elapsed from arrival until resources are allocated.
- Avg. Compositions' size: average number of NVMe composed together across the simulation.
- Avg. resource sharing: average number of workloads sharing an NVMe (or composition of NVMe) across the simulation.

Figure 4.4 summarizes the results' for missed deadlines of the tables, comparing Disaggregation-Aware strategy versus First Fit. The y-axes show the strategies' missed deadlines for the different load factors (x-axes) of the three scenarios shown in the tables. On 4.4a disaggregated infrastructure is shown whereas 4.4b shows the physically-attached one. Observing the disaggregated infrastructure, Disaggregation-Aware strategy performs better on high-bandwidth scenarios until the system becomes highly saturated. However, it performs roughly equal in the two other scenarios (slightly

Table 4.3: high-bandwidth scenario: 70%IO workloads

| Target CPU load factor | Policy | Obs. resources' utilization | | | % Missed deadlines | % Missed high-prio | NVMe usage(%) | Waiting time (avg.) | Compositions size (avg.) | Sharing ratio (avg.) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | CPU | BW | Cap | | | | | | |
| 0.9 | First Fit | 0.84 | 0.84 | 0.14 | 96.71% | 19.52% | 99.81% | 45916s | 1.00 | 1.00 |
| | PA First Fit | 0.74 | 0.67 | 0.11 | 98.05% | 20.19% | 79.75% | 63108s | 1.00 | 1.00 |
| | Disagg-Aware | 0.84 | 0.94 | 0.16 | 92.42% | 17.51% | 99.91% | 38428s | 1.02 | 1.26 |
| | PA Disagg-Aware | 0.74 | 0.66 | 0.12 | 97.18% | 19.58% | 78.77% | 57668s | 1.02 | 1.06 |
| 0.8 | First Fit | 0.72 | 0.84 | 0.14 | 89.13% | 18.71% | 99.91% | 18088s | 1.00 | 1.00 |
| | PA First Fit | 0.65 | 0.67 | 0.11 | 75.99% | 19.79% | 79.75% | 34302s | 1.00 | 1.00 |
| | Disagg-Aware | 0.87 | 1.08 | 0.18 | 4.70% | 0.40% | 92.51% | 569s | 1.24 | 1.96 |
| | PA Disagg-Aware | 0.65 | 0.67 | 0.11 | 75.05% | 19.25% | 77.98% | 29150s | 1.04 | 1.12 |
| 0.7 | First Fit | 0.69 | 0.84 | 0.14 | 47.55% | 11.80% | 99.72% | 3888s | 1.00 | 1.00 |
| | PA First Fit | 0.60 | 0.67 | 0.11 | 72.43% | 18.85% | 79.97% | 20616s | 1.00 | 1.00 |
| | Disagg-Aware | 0.70 | 0.85 | 0.14 | 0.47% | 0.47% | 76.07% | 29s | 1.36 | 2.06 |
| | PA Disagg-Aware | 0.60 | 0.67 | 0.11 | 71.43% | 18.31% | 76.88% | 15528s | 1.04 | 1.18 |
| 0.6 | First Fit | 0.60 | 0.74 | 0.12 | 0.07% | 0.07% | 87.92% | 53s | 1.00 | 1.00 |
| | PA First Fit | 0.56 | 0.67 | 0.11 | 64.52% | 17.00% | 79.90% | 7986s | 1.00 | 1.00 |
| | Disagg-Aware | 0.58 | 0.70 | 0.12 | 0.00% | 0.00% | 60.79% | 3s | 1.51 | 2.27 |
| | PA Disagg-Aware | 0.56 | 0.67 | 0.11 | 50.81% | 13.37% | 70.06% | 3712s | 1.15 | 1.57 |
| 0.5 | First Fit | 0.50 | 0.62 | 0.10 | 0.00% | 0.00% | 73.16% | 1s | 1.00 | 1.00 |
| | PA First Fit | 0.50 | 0.62 | 0.10 | 0.54% | 0.54% | 73.16% | 124s | 1.00 | 1.00 |
| | Disagg-Aware | 0.48 | 0.58 | 0.10 | 0.00% | 0.00% | 51.34% | 0s | 1.58 | 2.28 |
| | PA Disagg-Aware | 0.48 | 0.59 | 0.10 | 0.07% | 0.07% | 44.78% | 74s | 1.58 | 2.71 |

PA: Physically attached.

Table 4.4: High-capacity scenario: 70% IoT Workloads

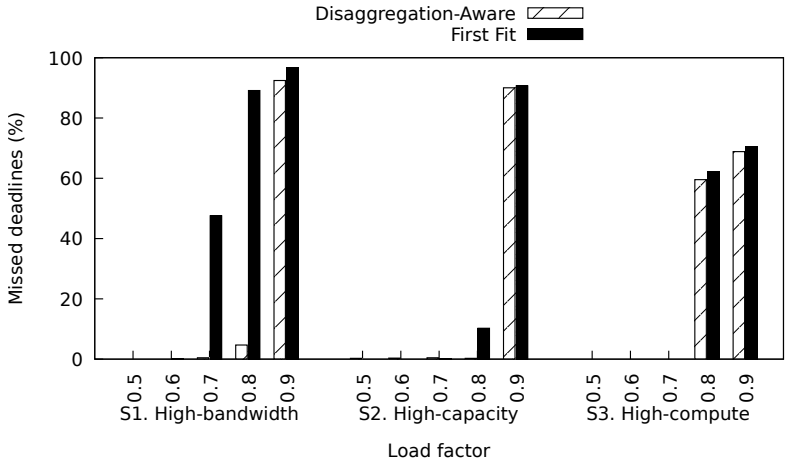
| Target CPU load factor | Policy | Obs. resources' utilization | | | % Missed deadlines | % Missed high-prio | NVMe usage(%) | Waiting time (avg.) | Compositions size (avg.) | Sharing ratio (avg.) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | CPU | BW | Cap | | | | | | |
| 0.9 | First Fit | 0.84 | 0.29 | 0.76 | 90.68% | 18.51% | 99.86% | 11396s | 1.00 | 1.00 |
| | PA First Fit | 0.74 | 0.23 | 0.61 | 95.51% | 19.85% | 79.83% | 21947s | 1.00 | 1.00 |
| | Disagg-Aware | 0.84 | 0.28 | 0.78 | 90.01% | 18.44% | 99.95% | 10614s | 1.00 | 1.00 |
| | PA Disagg-Aware | 0.74 | 0.22 | 0.62 | 94.97% | 19.72% | 79.87% | 21018s | 1.00 | 1.00 |
| 0.8 | First Fit | 0.79 | 0.28 | 0.76 | 10.26% | 5.16% | 98.56% | 945s | 1.00 | 1.00 |
| | PA First Fit | 0.70 | 0.23 | 0.61 | 72.10% | 18.98% | 79.91% | 10391s | 1.00 | 1.00 |
| | Disagg-Aware | 0.79 | 0.27 | 0.76 | 0.27% | 0.27% | 90.86% | 154s | 1.00 | 1.00 |
| | PA Disagg-Aware | 0.70 | 0.22 | 0.62 | 72.37% | 19.18% | 79.78% | 9635s | 1.00 | 1.00 |
| 0.7 | First Fit | 0.69 | 0.24 | 0.66 | 0.07% | 0.07% | 85.73% | 65s | 1.00 | 1.00 |
| | PA First Fit | 0.66 | 0.23 | 0.61 | 63.45% | 16.90% | 79.87% | 3965s | 1.00 | 1.00 |
| | Disagg-Aware | 0.69 | 0.24 | 0.66 | 0.47% | 0.47% | 79.97% | 29s | 1.00 | 1.00 |
| | PA Disagg-Aware | 0.66 | 0.22 | 0.62 | 60.63% | 15.96% | 79.29% | 3215s | 1.00 | 1.00 |
| 0.6 | First Fit | 0.58 | 0.20 | 0.56 | 0.00% | 0.00% | 71.93% | 8s | 1.00 | 1.00 |
| | PA First Fit | 0.58 | 0.20 | 0.56 | 1.27% | 1.27% | 71.93% | 118s | 1.00 | 1.00 |
| | Disagg-Aware | 0.58 | 0.20 | 0.56 | 0.34% | 0.34% | 68.92% | 8s | 1.00 | 1.00 |
| | PA Disagg-Aware | 0.58 | 0.20 | 0.56 | 0.74% | 0.74% | 69.50% | 109s | 1.00 | 1.00 |
| 0.5 | First Fit | 0.52 | 0.18 | 0.50 | 0.00% | 0.00% | 64.16% | 3s | 1.00 | 1.00 |
| | PA First Fit | 0.52 | 0.18 | 0.50 | 0.54% | 0.54% | 64.16% | 31s | 1.00 | 1.00 |
| | Disagg-Aware | 0.52 | 0.17 | 0.50 | 0.27% | 0.27% | 63.22% | 3s | 1.00 | 1.00 |
| | PA Disagg-Aware | 0.52 | 0.17 | 0.50 | 0.80% | 0.80% | 64.15% | 44s | 1.00 | 1.00 |

PA: Physically attached.

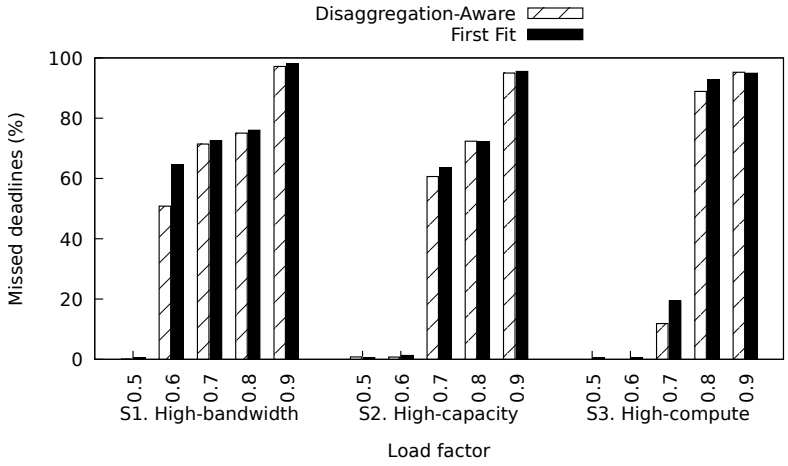
Table 4.5: High-compute scenario: 70% CPU intensive

| Target CPU load factor | Policy | Obs. resources' utilization | | | % Missed deadlines | % Missed high-prio | NVMe usage(%) | Waiting time (avg.) | Compositions size (avg.) | Sharing ratio (avg.) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | CPU | BW | Cap | | | | | | |
| 0.9 | First Fit | 0.79 | 0.30 | 0.10 | 70.46% | 4.69% | 40.74% | 13968s | 1.00 | 1.00 |
| | PA First Fit | 0.70 | 0.25 | 0.07 | 94.91% | 18.29% | 32.38% | 27096s | 1.00 | 1.00 |
| | Disagg-Aware | 0.78 | 0.28 | 0.10 | 68.85% | 1.88% | 63.51% | 14437s | 1.35 | 1.03 |
| | PA Disagg-Aware | 0.71 | 0.28 | 0.10 | 95.24% | 18.96% | 38.52% | 24523s | 1.04 | 1.04 |
| 0.8 | First Fit | 0.77 | 0.26 | 0.09 | 62.22% | 1.61% | 34.82% | 4983s | 1.00 | 0.98 |
| | PA First Fit | 0.70 | 0.24 | 0.08 | 92.83% | 18.49% | 31.53% | 14116s | 1.00 | 1.00 |
| | Disagg-Aware | 0.76 | 0.24 | 0.09 | 59.54% | 0.74% | 56.89% | 4774s | 1.44 | 0.98 |
| | PA Disagg-Aware | 0.70 | 0.23 | 0.08 | 88.88% | 16.14% | 33.41% | 12734s | 1.10 | 1.03 |
| 0.7 | First Fit | 0.69 | 0.22 | 0.07 | 0.00% | 0.00% | 29.30% | 186s | 1.00 | 0.97 |
| | PA First Fit | 0.68 | 0.22 | 0.07 | 19.36% | 6.77% | 29.26% | 1581s | 1.00 | 0.98 |
| | Disagg-Aware | 0.68 | 0.20 | 0.07 | 0.00% | 0.00% | 40.67% | 169s | 1.50 | 1.13 |
| | PA Disagg-Aware | 0.67 | 0.20 | 0.07 | 11.86% | 3.21% | 33.21% | 1409s | 1.41 | 1.22 |
| 0.6 | First Fit | 0.57 | 0.18 | 0.06 | 0.00% | 0.00% | 24.38% | 20s | 1.00 | 0.94 |
| | PA First Fit | 0.57 | 0.18 | 0.06 | 0.47% | 0.47% | 24.38% | 34s | 1.00 | 0.94 |
| | Disagg-Aware | 0.56 | 0.17 | 0.06 | 0.00% | 0.00% | 30.36% | 19s | 1.53 | 1.19 |
| | PA Disagg-Aware | 0.56 | 0.17 | 0.06 | 0.00% | 0.00% | 27.82% | 26s | 1.53 | 1.26 |
| 0.5 | First Fit | 0.50 | 0.16 | 0.06 | 0.00% | 0.00% | 21.46% | 2s | 1.00 | 0.92 |
| | PA First Fit | 0.50 | 0.16 | 0.06 | 0.50% | 0.50% | 21.46% | 8s | 1.00 | 0.92 |
| | Disagg-Aware | 0.50 | 0.14 | 0.05 | 0.00% | 0.00% | 26.61% | 1s | 1.53 | 1.16 |
| | PA Disagg-Aware | 0.50 | 0.15 | 0.05 | 0.00% | 0.00% | 24.69% | 4s | 1.54 | 1.25 |

PA: Physically attached.

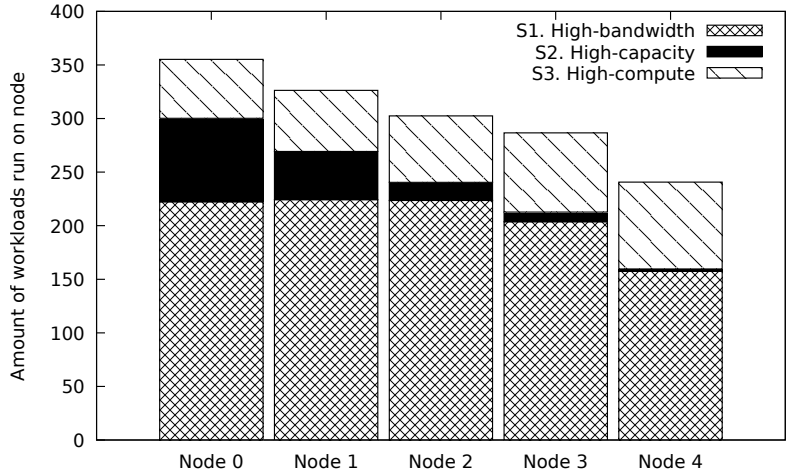



(a) Disaggregated layout

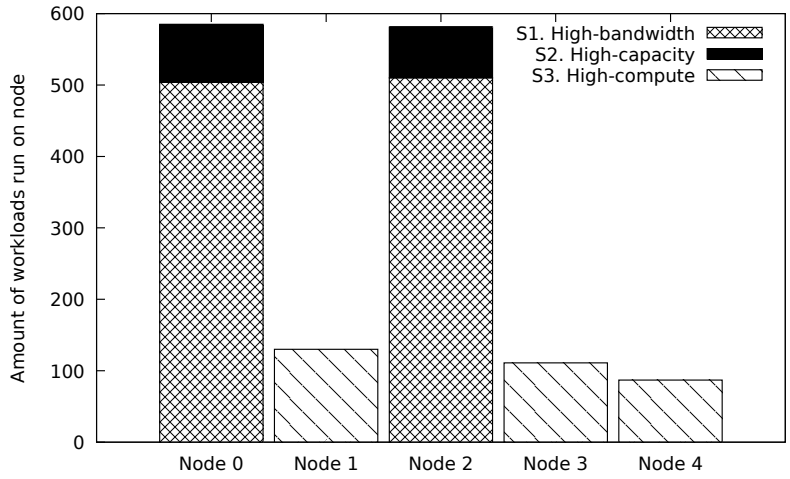



(b) Physically-attached

Figure 4.4: Performance of the strategies on different load factors for the three scenarios. Disaggregated and physically-attached infrastructures.




(a) Disaggregated layout




(b) Physically attached

Figure 4.5: Distribution of workloads-type run on each node of the infrastructure during a simulation. Disaggregated and physically attached infrastructures. Nodes 0 and 2 have physically-attached NVMe on (b).

worse in some situations, but the absolute numbers show this difference is neglectable). This is due in the high-bandwidth scenario, most of the workloads (bandwidth-bound) take advantage of sharing and composition. Our performance model enables the knowledge that composing resources may benefit bandwidth-bound workloads. Thus, a model-aware strategy being able to make compositions and raise sharing ratios outperforms traditional data-centers. In the other two scenarios, however, most workloads do not benefit from that. Thus, the Disaggregation-Aware strategy slightly helps mitigate fragmentation, but it does not show a significant impact versus not doing so. Special emphasis should be made on that having a model for the other two kinds of workloads could enable composition on those scenarios and enhance our performance. Compared to having a physically attached infrastructure, it is clear that all scenarios and strategies perform much worse on the mid to high saturation levels. On low saturation levels, the performance is close to optimal regardless of the infrastructure or scenario, as in such relaxed systems, it does not matter how bad the decisions are, as there is plenty of room for error.

In figure 4.5 we present the amount of workloads run on each of the compute nodes by its kind, comparing the disaggregated infrastructure (figure 4.5a) and the physically attached (4.5b) one. It can be observed how disaggregation allows balancing all the workloads, regardless of their need to use NVMe across all the nodes. Simultaneously, the physically attached is more restrictive and forces all the workloads requiring NVMe to be allocated on the only two nodes with NVMe availability, thus limiting scheduling flexibility and therefore performing worse. In these figures, a high-bandwidth scenario is shown with a target CPU load factor of 0.7. However, the remaining scenarios and load factors have the same behavior.

Notice that low load factors are not represented due to such situations. The saturation of the system at any point reaches 70%. Thus we can never start the measurement window on which we compute the metrics.

On the other hand, as previously stated, our scheduling policy is designed to shift between placement policies according to observed capacity and bandwidth utilization. To show this behavior, we depict in figure 4.6 the number of workloads' resources placed with each policy on each load factor and scenario. It can be observed how in the high-bandwidth and high-compute scenarios, almost only the "maximize composition" policy is used. This is due bandwidth requested is either very high (high-bandwidth) or both bandwidth and capacity requests are really low (high-compute), provoking the use of only one of the policies. However, in the high-capacity scenario, as the target CPU load factor increases, the amount of request for capacity increases, provoking an escalated shift to using only minimize fragmentation policy. This shows the expected

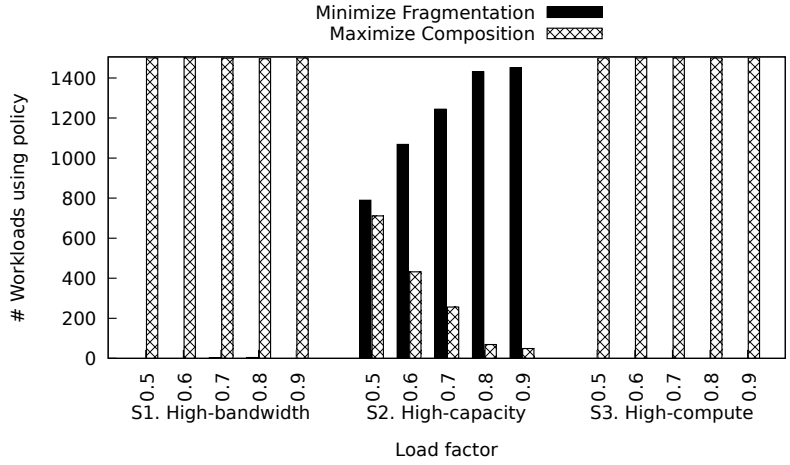



Figure 4.6: Amount of workloads' resources allocated by each placement policy, load factor, and scenario.

behavior, as in such situations, minimizing fragmentation either slightly improves results (0.8 target CPU load factor) or does not worsen performance.

### 4.4.4 *Scheduling with tight deadlines*

An alternative to analyzing the system's performance in tight situations is to set a specific load factor and add pressure by changing the deadline factor of high-priority workloads. Previously we established a high-priority workload has a deadline factor of 1.2 times its base execution time to complete. The lowest the deadline factor is, the sooner workloads need to be completed and the more pressure the system has. This experiment explores how setting this parameter impacts performance while maintaining the target CPU load factor in 0.7. 0.7 is chosen as we consider the optimal scenario a not relaxed but not saturated system.

Figure 4.7 presents the high-priority deadlines missed (percentage) as the deadline factor changes. From left to right, the lowest factor the hardest to fulfill the deadlines are. The figure shows Disaggregation-Aware performs better in the high-bandwidth scenario and has a similar performance in the high-capacity scenario. However, the high-compute scenario outperforms First Fit when the high-priority coefficient is very tight (1.0). Thus, Disaggregation-Aware performs correctly in the target scenario (having high-bandwidth model-enabled workloads) and does not worsen a general scenario with a low presence of such workloads.

Figure 4.8 depicts the average amount of workloads sharing the same resource. This figure allows us to observe how the sharing ratio increases when Disaggregation-

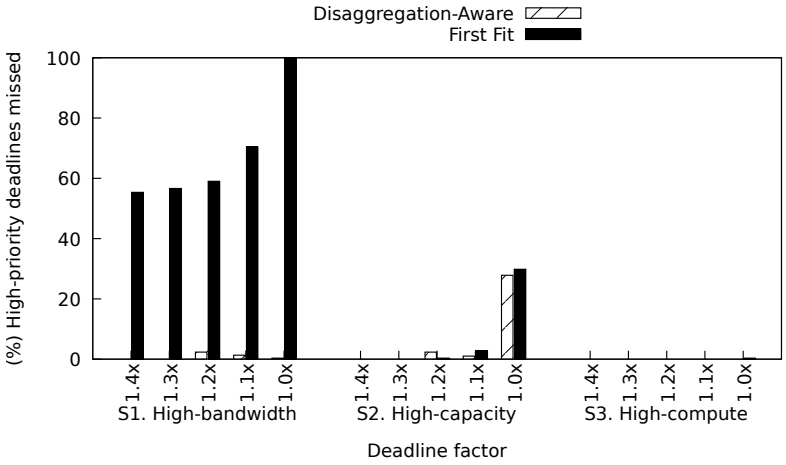



Figure 4.7: High-priority workloads' deadline factor impact on high-priority deadlines missed. Results are categorized per each scenario and deadline factor.

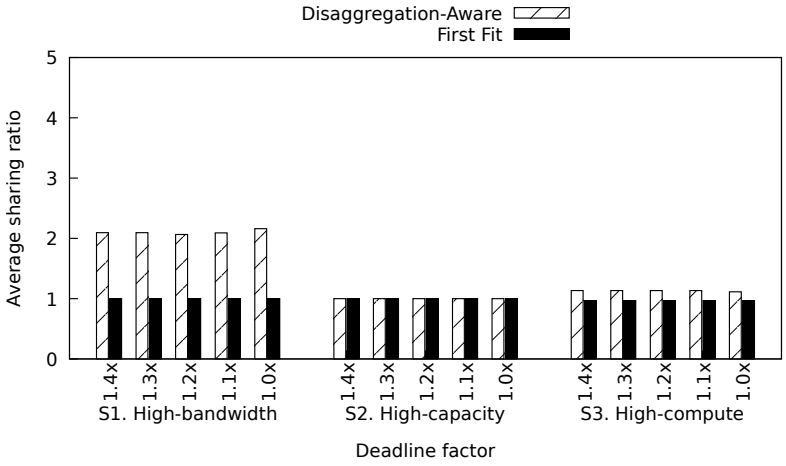



Figure 4.8: High-priority workloads' deadline factor impact on workloads' sharing ratio. Results are categorized per each scenario and deadline factor.

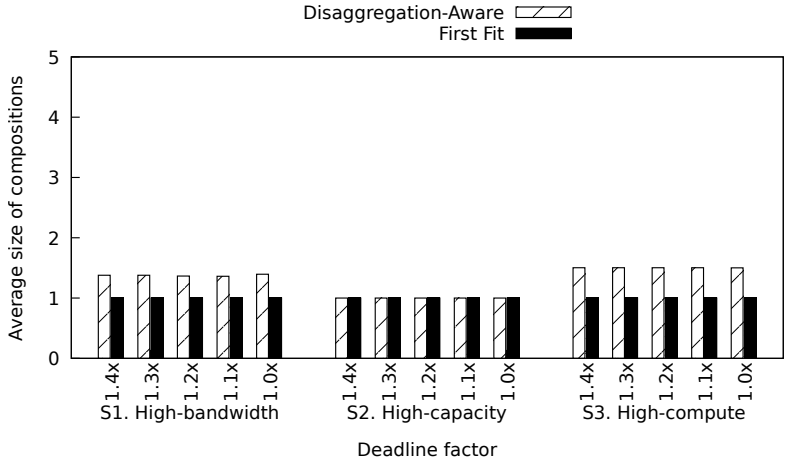



Figure 4.9: High-priority workloads' deadline factor impact on average composition size. Results are categorized per each scenario and deadline factor.

Aware performs better (high-bandwidth scenario). While the high-capacity scenario, with equal performance, does not benefit from it. Thus, not being able to exploit the advantages of resource disaggregation. The sharing ratio is increased thanks to a greater presence of bandwidth-bound, modeled workloads. However, the high-compute scenario also benefits from this, due there is a lower amount of workloads using NVMe (neither capacity-intensive NVMe nor bandwidth-bound). The presence of such workloads allows to exploit composition for the arriving bandwidth-bound workloads, as using more NVMe than needed will not prevent incoming capacity-intensive workloads from running due to lack of NVMe, as there are not so many workloads requiring it. This can be verified through figure 4.9. It depicts the average composition size on the different deadline factors for high-priority workloads. It is indeed observed how Disaggregation-Aware makes greater compositions on the high-bandwidth scenario and does some amount of them in the high-compute scenario. In both scenarios, performance is better compared to first fit. Therefore enabling a model on workloads allowing to share resource's that otherwise would not be possible enhances system's performance.

### 4.4.5 *Resources availability*

In the described experiments, the simulated systems had 10 NVMe devices. To understand if the availability of resources impacts performance, we run experiments diminishing the number of devices. Results are displayed on figure 4.10. On the x-axes the number of devices on the simulated system. The y-axes represent the percentage

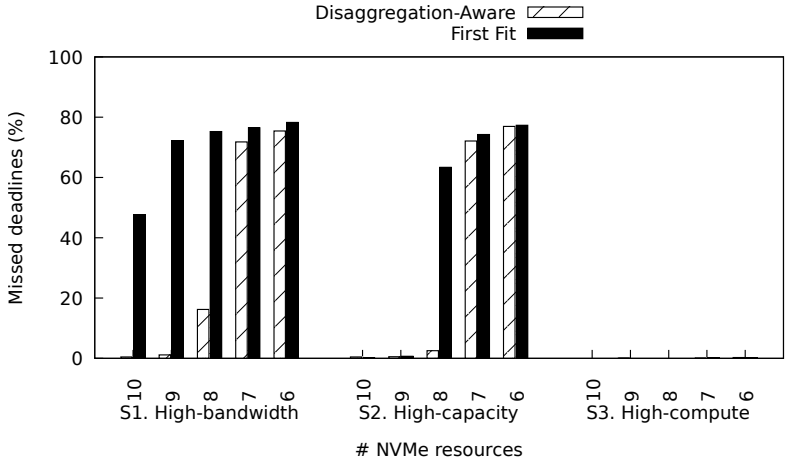



Figure 4.10: Performance of strategies for different nunmber of NVMe devices on the three different scenarios. Results based on a target CPU load factor of 0.7.

of missed deadlines. The results shown are for a fixed target CPU load factor of 0.7. The three scenarios are displayed. It can be observed how, for the high-bandwidth scenario, where workloads are mostly sensitive to bandwidth, there is a significant difference between both policies in many cases. Having more resources available implies more options to make larger compositions, increasing provided bandwidth and thus sensitive workloads' performance. As the amount of available resources diminishes, this difference diminishes as well. As more constrained the system is on resources, while maintaining the demand, implies less flexibility to make compositions (which uses more resources for the same workload), thus the less our policy benefits the system. Scenarios up to one single device are omitted for simplicity, due the tendency to behave equally is already observable.

### 4.4.6 *Mixing bandwidth-bound modeled workloads*

As a final experiment, we add a second modeled workload into the high-bandwidth scenario. This workload is the synthetic benchmark fio [21]. This benchmark is intended to check for performance failures on storage devices, thus generating high and steady bandwidth loads. Due to the synthetic nature of the workload, we observed an almost perfectly linear nature after running the same set of experiments we run for SMUFIN. That is, let $t$ be the execution time of the workload on a single NVMe device; when composing two devices together, the execution time becomes $t/2$. Despite its synthetic nature, it demonstrates how bandwidth-intensive workloads may benefit our proposed policy, even when mixed. For this purpose, we repeated the experiments shown in

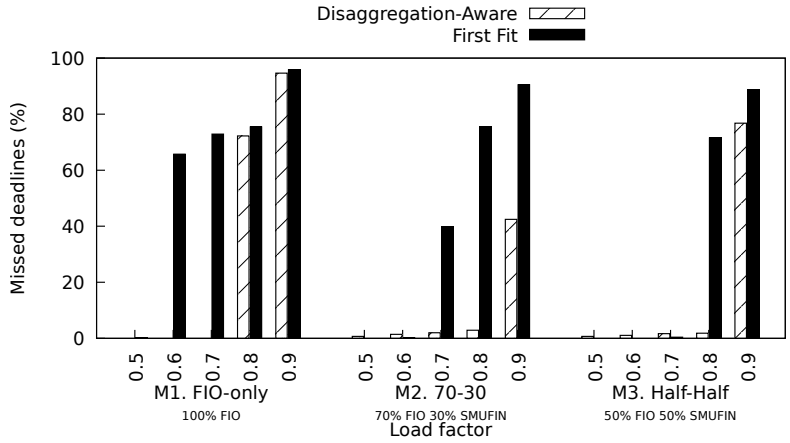



Figure 4.11: Performance of the strategies on different load factors for three FIO-SMUFIN workloads' mixes. Disaggregated infrastructure.

figure 4.4 for the high-bandwidth scenario S1. However, instead of consisting of a 70% only SMUFIN workloads, we take three mixes on this 70% high-bandwidth workloads:

- M1. FIO-only: 100% FIO, no SMUFIN.
- M2. 70-30: 70% FIO, 30% SMUFIN.
- M3. Half-Half: 50% FIO, 50% SMUFIN.

Notice the percentages are relative to the 70% total bandwidth-bound workloads. The remaining, as in previous experiments, are composed of capacity and compute-bound workloads. Figure 4.11 shows the missed deadlines on different load factors for each of the described mixes. It can be noticed how our policy outperforms first-fit, particularly on saturated systems. It can also be observed how, in some combinations, on relaxed systems, there is no gain due the overall performance is nearly optimal, as almost no deadlines are missed on either policy.

## 4.5 RELATED WORK

The The idea of SDI has been increasingly studied in the literature over the past few years. Chapter 2.1 mentions a few of that related work. However, due to the recent nature of these infrastructures, there aren't yet many efforts regarding orchestrating them. Nonetheless, we can find literature on scheduling efforts that can be similar to the work presented in this chapter.

[36] presents a scheduler strategy trying to optimize resource usage by allocating opportunistic containers on under-utilized resources. This strategy is in the context of

Big Data workloads and particularly on the YARN resource manager. The chapter lets YARN schedule containers by its default policy (fair or capacity scheduler) and then backfills the opportunistic containers in the unused space. In this chapter, they call allocated resources to a job, not using them as fragmentation. However, our concept is different. As explained in section 4.2.1, our fragmentation arises when allocating the requested resources leaves a left-over on cores or nvme capacity/bandwidth. It is not because the job is requested but does not make use of it, but because the available resources were exceeding the job's demand. Although this left-over is free to use by any other job, it potentially might not be possible to use due to its size, not fitting most jobs arriving into the system. This chapter uses their concept of resources' slack to calculate how much utilized the cores are within a node. However, their proposal is to calculate the resources requested and allocated for a given job and not use it, whereas we use it to calculate the number of free cores remaining on the computing node. This concept is used when there is a need to allocate free resources due to existing left-overs' (fragmented free resources) cannot be used.

[64] a power-scheduling system is presented. The proposal tries to solve the problem of jobs waiting in the queue due to their requested power is currently unavailable. For this, it proposes backfilling scheduling where jobs are allocated less power than required, even though this translates into performance degradation, to maximize the power-wise utilization of the cluster's resources as well as throughput.

[51] presents a modification of a scheduler to manage resources on virtual machines. The aim is to ensure all jobs have a fair share of the system's resources in a shared environment. To achieve this goal, containers are enabled in a larger parent container. Those smaller containers are allocated resources from the parent container, on which jobs will be run. Thus the share of resources, as many jobs will run within the same parent. To ensure a fair share of parents' resources across the containers, they introduce metrics that help the scheduler make decisions. All containers are assigned a minimum share of the resources, granting more resources if they become available up to a given limit. Different from our work, in this chapter, access to resources is prioritized over performance, degrading jobs if required to ensure a fair share. On the other hand, differently from our proposal, a single resource (core/nvme device) is not shared simultaneously by many jobs, but a parent container has a set of them and splits it fairly across jobs.

[27] presents Dominant Resource Fairness (DRF) allocation algorithm. Their algorithm intends to improve resource sharing by assuming workloads usually lie about the resource usage they request. This provokes an underutilization of resources. DRF ensures fair resource usage across users. As opposed to our proposed policies, they do

not consider SLA but focus solely on increasing resource usage and fairness. This is a similar work than [52] where they apply an algorithm to ensure sharing of resources in a cluster of virtual machines. The main difference between those works is that the first is focusing on users rather than in a data center level, while the second is focusing on virtual machines sharing on a data center.

Finally, [63] presents a scheduling proposal for in order to help allocate resources on-demand, trying to maximize the number of resources being used and minimize node latency. To the best of our knowledge, there is no literature about strategies for disaggregation on similar terms as this chapter.

## 4.6 CONCLUSIONS

This chapter shows that providing a performance model for workloads under disaggregation allows for better placement strategies. Such models enable the designing of placement strategies aware of the workloads' requirements to fully leverage the disaggregation of resources in data centers.

We have proposed two model-aware placement policies that benefit from disaggregation. One policy enables resource composition and resource sharing, whereas the second one deals with system fragmentation. Minimizing such fragmentation helps not to degrade performance in the scenarios where our modeled workloads are marginally represented. All of these are relevant to help meet workload requirements in data centers. As observed through simulation, resource composition increases resource sharing up to 2x. We have shown how these advantages are critically enabled by resource disaggregation. It has compared results to a physically-attached infrastructure, where performance is significantly slower. When using the first fit policy, results show that a disaggregated system can reduce missed deadlines up to 49% when compared to a physically-attached system. Enabling disaggregation helps to meet all deadlines on mid-relaxed systems with trivial first fit strategy, while our proposed strategy can deal with more saturated systems as well. On the other hand, when workload awareness is enabled in a disaggregated system, our proposed policy reduces missed deadlines up to 100% (no deadlines missed) under load factors of 0.7 or less. When the system is more saturated (load factor of 0.8), this reduction is of 18.96%, still becoming a significant improvement. These results show that not being able to dynamically allocate resources into the compute-nodes limits orchestration flexibility leading to performance degradation, demonstrating the advantages of resource disaggregation.

## 4.7 PUBLICATIONS

The contents of this contribution are summarized in the publication:
[10]Aaron Call, Jordà Polo, and David Carrera. "Workload-Aware Placement Strategies to Leverage Disaggregated Resources in the data center." In: *IEEE Systems Journal 16.1 (2022)*, pp. 1697–1708. DOI: 10.1109/JSYST.2021.3090306.

# 5

# ACCELERATORS PARTITIONING AND ORCHESTRATION FOR HETEROGENEOUS DISAGGREGATED DATA CENTERS

In this chapter, we describe this thesis's third and final contribution. In our previous chapters, we have exposed how resource disaggregation allows us to share a device across workloads without sharing computing resources, bringing us the opportunity to share without experiencing performance degradation. However, since workloads are unaware that the device is shared, we had to handle potential conflicts such as two workloads overwriting the same areas of a block device, among others. Previously we did so by leveraging the knowledge that the phase of SMUFIN we used utilizes a filesystem rather than the device as a block device. So we could partition our resources and indicate to each instance where to write. However, ideally, this should be done without needing to inform any workload. Moreover, this wouldn't be possible in other SMUFIN phases where the NVMe is used as a block device. This chapter introduces GPU partitioning, where we partition the GPU memory and expose to the nodes only a portion of that memory as if it was an entire GPU, eliminating any potential conflicts and the need to make the workloads aware of it. We call this portions Virtual GPU (vGPU). If the vGPUs are also disaggregated, the data center's flexibility increases dramatically. In this chapter, we add GPU into the mix. First, we show how disaggregating vGPU achieves the same result as with two unsplittable disaggregated GPUs. On the other hand, adding new resources adds a new layer of complexity. Such resources are more expensive to acquire and handle than compute nodes, and thus their availability is more limited. When there is a pool of heterogeneous resources which are limited in quantity, and the demand is high, deciding where to run the workloads requesting them becomes crucial. In this contribution, we explore how minimizing fragmentation has a significant impact when dealing with heterogeneous resources.

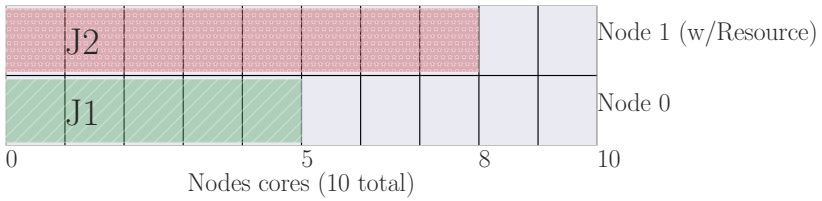
## 5.1 GPU PARTITIONING

In the previous two contributions, we have explored how disaggregation increases data center flexibility by allowing to de-attach and re-attach devices across nodes. However, sharing the resource simultaneously across nodes, it was something not possible without a way to solve potential memory conflicts, as the entire device was exposed. To do so, workloads had to be aware of the sharing and then be modified accordingly to enable such an option. However, doing so defies one of the main advantages of disaggregation is to share resources in a workload-unaware fashion to increase flexibility as well as being able to expand those techniques to generic data centers.

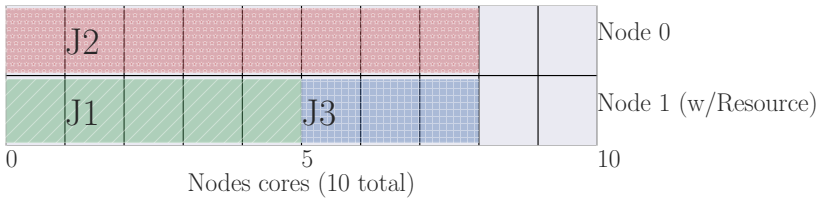
GPU partitioning allows us to solve this conflict. In this chapter, we use the implementation for partitioning NVIDIA vGPU [59]. vGPU allows dealing with those conflicts, enabling sharing the same GPU across many virtual machines in a non-conflicting way. vGPU is a software layer that splits a GPU memory and exposes each partition as a single smaller GPU that is then attached to a virtual machine. The software is placed on top of a hypervisor and top of the NVIDIA virtualization software. The main difference between existing mechanisms of sharing GPUs or other devices is that the workloads or a scheduler had to manage potential conflicts when sharing the same memory. With this technology, we bypass this problem as the software splits the memory, and it is not possible to have such conflicts. Therefore workloads perceive the partitions as an exclusive GPU and can use them as they would use a physical GPU exclusively assigned to them.

In this chapter, we disaggregate partitioned GPU so we achieve disaggregation of vGPU instances. GPU disaggregation has been researched, and there are technologies such as rCUDA [68][67] enabling GPU disaggregation through API remoting. [17] proposes a different scheme to disaggregate resources, bypassing the limits of solely exposing an API. Falconwitch [23] is a proprietary technology achieving GPU disaggregation allowing for topology alteration supporting multiple hosts and a software-defined fabric.

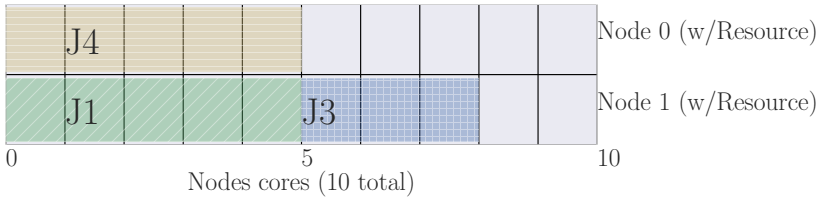
Given GPU can now be split into smaller pieces and shared across nodes, we are eliminating the constraint we previously encountered. This raises, even more, our resource management flexibility. Figure 5.1 tries to exemplify this situation. On scenario 5.1a a resource (namely GPU, NVMe, or other) is physically-attached to node 0. Jobs J1 (green) and J2 (red) get allocated a number of CPU cores, leaving node 0 with only 2 free cores. Then, J3 arrives (blue), demanding 4 cores and resource usage. However, the cores are only available on node 1, but the resource is attached to node 0. Thus it




(a) Physically attached resource in node 0. Job J3 (blue) has to wait until J2(red) finishes to have some cores available.




(b) Disaggregated resource. Job J3 (blue) is executed in node 1 with the resource remotely attached. J4 (yellow) has to wait until either J3 finishes, so resource becomes free.




(c) Disaggregated and shared resource. Job J4 (yellow) is executed in node 0 with the resource remotely attached and shared across nodes (with blue J3)

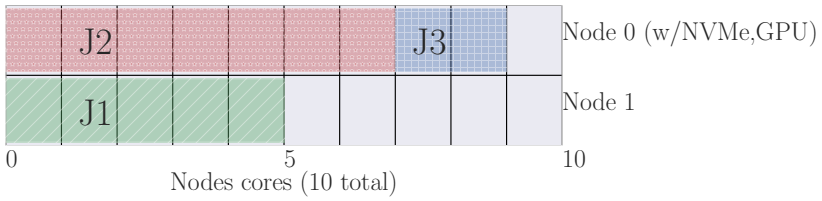
Figure 5.1: Timeline showing the execution of three jobs in two nodes with a physically-attached resource (top), disaggregated resource (middle) and disaggregated and shared resources (bottom)

can't be allocated resources until J2 frees cores so J3 can run and access the resource. Disaggregation solves this issue by allowing the resource to be remotely attached to node 1, with free cores available, so J3 can be allocated there and run (case 5.1b). In this scenario, we are assuming that neither J1 nor J2 access the resource. This scenario has the restriction that the resource will remain attached to node 1 and can't be re-attached until J3 completes its task. Let's imagine while J3 is running, J2 completes, and then J4 (yellow) demanding 5 cores arrives. J4 does not have cores available on node 1. However, node 0 has free cores. Given node 0 does not have the resource attached, it can't be allocated there until J3 finishes and the resource can be re-attached. This is the limitation we have had so far with disaggregation. We can overcome the restriction if a resource can be shared simultaneously across nodes. As we stated, this was not previously possible due resources are accessed as a block device, thus sharing it across nodes simultaneously would generate memory conflicts between workloads, as workloads aren't aware other jobs are using it (the node sees resources as exclusive resources). vGPU aids in solving the issue as it partitions memory to avoid such conflicts. Then, this allows attaching the resource into node 0 and J4 can run there (case 5.1c). This example shows how the resources may be utilized as best as possible thanks to the possibility of disaggregation combined with resource-sharing across nodes.

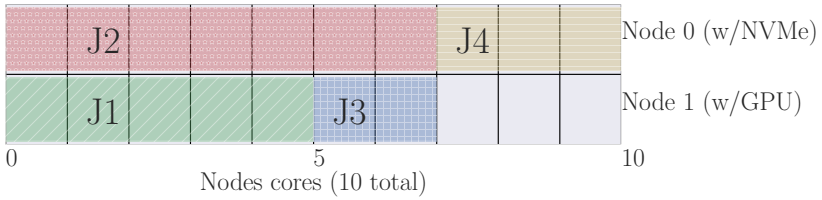
## 5.2 HETEROGENEOUS RESOURCES

In this chapter, we are dealing with a heterogeneous set of resources, as besides NVMe we are now adding GPU. Both resources are limited in quantity and very demanded by a number of workloads in recent years. Additionally, as we introduced earlier, GPU provides an extra level of flexibility, as we can split them and share the splits into multiple nodes simultaneously. While with NVMe we can attach and re-attach to nodes as we please, we can't attach them to two nodes simultaneously, as doing so requires the workload to be aware that the NVMe is being shared to avoid potential conflicts. Thus, we face the situation where deciding how to place workloads using GPU can become critical.

This is exemplified in figure 5.2. In the example, we have a situation with four jobs. Jobs J2 (red) and J4 (yellow) require an NVMe, and use 7 and 3 CPU cores, respectively. Job J3 requires access to a GPU and utilizes 2 cores, while J1 (green) solely uses 5 CPU cores. When the third job, J3 (blue) arrives, it can fit either in node 0 or 1 (figure 5.2a). If we allocate it in the first candidate, node 0, as shown in the figure, then we only leave 1 core available on the node while at the same time, the two resources that are




(a) NVMe is attached to node 0. Job J4 (yellow) can't run because there aren't enough cores.




(b) Disaggregated resource. Job J3 (blue) is executed in node 1 with the resource remotely attached. J4 (yellow) has to wait until either J3 finishes, so resource becomes free.

Figure 5.2: Timeline showing the execution of four jobs in two nodes using NVMe and GPU. Showing the impact of workload allocation under heterogeneous resources.

limited in amount become attached there. This provokes the situation that when job J4 arrives requiring an NVMe, despite having enough cores to run on node 1, it can't be allocated there because the NVMe is attached on node 0. Notice this situation would not occur if job J4 required a GPU instead of NVMe. Given that the GPU is a resource more flexible and we can split it, we could attach it to both nodes simultaneously, given there was enough memory for both jobs. However, as NVMe is less flexible, it cannot run. Thus, a better placement would have been the one on figure 5.2b where all jobs can be run.

In the disaggregated-aware policy we presented in chapter 4, we considered system fragmentation which focused on the unused space within a resource. Given that we only had one kind of resource, our policy considered the balance between remaining space within an attached resource and the remaining cores in the computing node. If we did the same in this scenario of heterogeneous resources, in 5.2a J3 would have been allocated to node 0 as well because most of the cores of a node will be used, leaving the minimum spare space possible on the nodes. Notice J3 utilizes GPU. Regardless of whether it is a virtual GPU or not, the assigned one will be fully utilized by J3 as per resource characteristics. Thus the key difference between our previous work and this one is the fact that we have different resources, forcing us to consider fragmentation attending to the different characteristics of the resources.

## 5.3 RESOURCE-AWARE POLICY

**Algorithm 3** GPU-Partitioning Algorithm

```
1: procedure PARTITION GPU(job, deadline)
2:     gpu = findUnusedGPU()
3:     if ¬gpu then # (GPU not available)
4:         return false
5:     else
6:         bwPartition = getBandwidth(gpu) / bandwidth
7:         memoryPartition = getMemory(gpu) / memory
8:         numPartitions = min (bwPartitions,memoryPartitions)
9:         for p in numPartitions do
10:            addNewvGPU(gpu)
11:        end for
12:    end if
13: end procedure
```

The first algorithm we introduce is how to partition a GPU into vGPU units. Algorithm 3 shows the pseudo-code. GPU parameters are bandwidth and memory. We follow the process to make as many splits as possible as long as the job fits in them. That is, we take advantage of the GPU slack the workload would produce if assigned the full GPU in exclusive. If a job is requesting 4GB/s bandwidth and we have a GPU with 16GB/s bandwidth, the slack would be 12GB/s. Instead, we can split the GPU into 4 vGPU units (line 7). Each of the units would then be fully utilized if a workload arrived. We must also consider the memory (line 6), for which the same technique applies. Then we select the minimum of both parameters (line 8) and split the GPU accordingly (lines 9-10).

Now we can split GPUs into smaller portions, but we must decide how to allocate resources for workloads using GPUs. This is shown in the pseudo-code of algorithm **??**. The process is first to find if there is any GPU with an available vGPU. That is, a vGPU is not being used. Line 2 invokes this algorithm, defined in lines 21-30. The process to find an available vGPU is to check on the list of GPUs that have a vGPU (lines 22-23). Out of that vGPUs, we check if there are some not in use and for which allocating a new workload would meet the workload's deadline (line 24). In this chapter we use YOLO[70] as an accelerated workload. We experimented with its behavior when different instances are running concurrently on the same partitioned GPU. The results are depicted in figure 5.3. In the evaluation section, the environment details are described. We use this information to decide whether allocating the new workload meets the deadline. If allocating the workload on a partition degrades its performance compared to using another GPU, we choose to run on the other GPU. To do so, we identify how many workloads are already running across all the vGPUs of

**Algorithm 4** GPU-Aware Placement

```
1: procedure GPUAwarePlacement(job, deadline)
2:     vgpu = availvGPU(deadline)
3:     if vgpu then
4:         candidatesList = {}
5:         for all n in nodes() do
6:             if freeCores(n) ≥ cores(job) then
7:                 insert(n,vgpu,candidatesList)
8:             end if
9:         end for
10:        if ¬empty(candidateList) then
11:            candidate = ResAwareCoresAlloc(job, candidateList)
12:            assignvGPU(candidate.n,candidate.vgpu)
13:            assignCores(candidate.n,job)
14:        end if
15:    else if PartitionGPU(job, deadline) then
16:        ResAwarePlacmnet(job,deadline)
17:    else
18:        return false
19:    end if
20: end procedure
21: procedure availvGPU(deadline)
22:    for g in gpus() do
23:        for v in vgpus(g) do
24:            if ¬inUse(v) and newWorkloadTime(v) + step ≤ deadline then
25:                return v
26:            end if
27:        end for
28:    end for
29:    return false
30: end procedure
31: procedure ResAwareCoresAlloc(job, candidateNodes)
32:    candidateList = {}
33:    for all c in candidateNodes do
34:        remCores = freeCores(c.n)-cores
35:        f = remCores /β
36:        insertSortedDesceding(candidateList, f, c)
37:    end for
38:    return first(candidateList)
39: end procedure
```

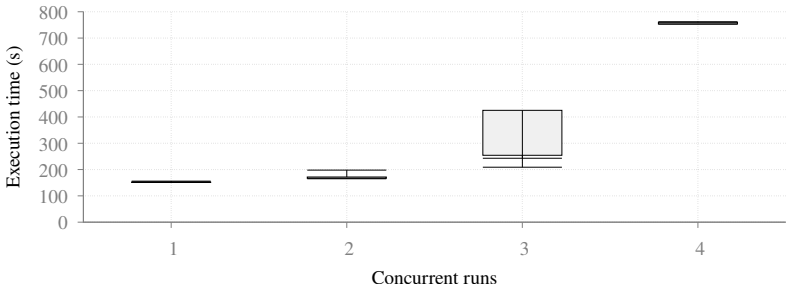



Figure 5.3: YOLO Execution times while using vGPU with a 4-way splitted GPU. 1 to 4 concurrent runs, each on its own vGPU. Dots indicate the median time across executions.

the GPU. Then we apply the model to estimate performance if we run it on that vGPU to determine if we are meeting the deadline.

If there is a candidate vGPU, we look for all available nodes if there is one with enough cores, and if so, we add it to our candidates' list (lines 5-7). If some nodes are candidates, it is time to decide which of them is chosen (line 11). This is described in the procedure on lines 31-43. We calculate a fitness parameter for each candidate, f (lines 33-36), equation 5.1. Let $c_n$ be the remaining cores of node $n$, and $\beta$ the minimum amount of cores a workload using an NVMe device requests, the fraction between both provides the fitness parameter. We calculate the slack to allocate workloads using NVMe after allocating job $j$ on the node. This way, we prioritize the node on which the less flexible resource has more chances to allocate workloads. Since a GPU can be easily partitioned is not often limited by available cores. Once a candidate is chosen, all that remains is to assign the vGPU to the elected node and allocate the job to the node's cores (lines 12 and 13, respectively). If there was not a vGPU found, we attempt to partition an existing GPU with the algorithm 3 we just described (line 15). If found, we iterate again, knowing we will now find an available vGPU (line 16). If not, we cannot allocate a GPU workload due to unavailable resources.

$$fitness = \frac{c_n}{\beta} \tag{5.1}$$

To allocate NVMe workloads, we attempt to over-provision as long as we make a performance gain, as we did on chapter 4. When a workload arrives, we use the first fit to locate an available NVMe. Then, as long as its performance improves, we keep looking for more NVMe available and over-provision the workload. Regarding cores-only workloads, we allocate the workload on the computational node that induces less inter-resource fragmentation using equation 5.1 we just explained. Thus, in this chapter, we are incorporating GPU into the mix and modifying our previous policy on

| Workload type | Base execution time | NVMe bandwidth | NVMe capacity | GPU Bandwidth | GPU memory | CPU cores |
|---|---|---|---|---|---|---|
| Bandwidth-bound | 1600s | 1800MB/s | 43GB | N/A | N/A | 6 |
| Capacity-bound | 800s | 160MB/s | 600GB | N/A | N/A | 6 |
| Compute-bound | 900s | N/A | N/A | N/A | N/A | 15 |
| GPU-bound | 152s | N/A | N/A | 1000MB/s | 2GB | 2 |

Table 5.1: Workloads' requested resources

| Non-priority deadline factor | deadline factor | High-priority workloads (%) | Number of nodes | Cores per node | NVMe bandwidth | NVMe capacity | Number NVMe on pool | GPU Memory | GPU Bandwidth | Number of GPU in pool |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 1.2 | 20% | 5 | 25 | 2 GB/s | 600GB | 10 | 16GB | 4GB/s | 1 |

Table 5.2: Default simulation parameters

minimizing fragmentation to minimize inter-resource fragmentation. It is the latter that allows us to take into consideration a heterogeneous set of resources.

## 5.4 EVALUATION

In this section, we evaluate the performance of splitting GPU. Our environment for this setup consists of a NVIDIA Tesla T4 GPU [60] and a workstation with an AMD Ryzen 7 5800X 3.8GHz CPU and 16GB DDR4 memory. We did not possess a way to disaggregate GPUs in our lab at the moment of this work. We lacked network and computing resources, as well as the technology providing such disaggregation. However, GPU disaggregation has been widely researched. There are technologies such as rCUDA [68][67] enabling GPU disaggregation through API remoting. In those works, they show a methodology upon which CUDA API is exposed over the network on client nodes so that the nodes do not require the GPU physically attached. The drawback of those methods is that we are limited to what CUDA allows, but it does not expose the device on its own. For vGPU technology to work, however, it required to use the device as a whole and not just give instructions to it. On the other hand, resource

| | Bandwidth-bound (%) | Capacity-bound (%) | GPU -only (%) | Compute-bound(%) |
|---|---|---|---|---|
| S1. NVMe-intensive | 50% | 20% | 10% | 20% |
| S2. GPU -intensive | 10% | 0% | 80% | 10% |
| S3. Mixed-resources | 20% | 20% | 40% | 10% |

Table 5.3: Workload distributions for the three scenarios

disaggregation, as presented in this thesis, provides exactly that, exposing a device or a piece of it as if it was physically attached to the client's machine, and thus the client would use it as a device in itself. rCUDA, however, just grants the ability to process instructions on a remote device. [17] proposes a different scheme to disaggregate resources, bypassing the limits of solely exposing an API. They propose an entire architecture to disaggregate resources, and not only GPU. The proposal builds an SDI data center. Falconwitch [23] is a proprietary technology achieving GPU disaggregation allowing for topology alteration supporting multiple hosts and a software-defined fabric. This is a real-world implementation of disaggregating devices, as presented in this work. This technology, combined with vGPU technology presented in this chapter, would allow us to not only disaggregate GPU but also to split them and expose them as single devices to the client nodes. Thus, providing all the benefits of such flexibility as presented in this chapter. This related work proves that disaggregating GPU is not just feasible but is already out there. Therefore we assume we do disaggregate them, and via GPU partitioning, we achieve disaggregation of vGPU instances.

This work and evaluation is a continuation of our previous work [10]. Thus we use the same simulator [8]. This time we incorporate GPU resources and vGPU partitioning in our simulator and add the policies we explained in the previous section. Consequently, our simulator now supports workloads requesting NVMe and accelerated workloads using GPU and allows us to partition GPUs into vGPUs. Then the vGPUs can be assigned to each of the nodes. Due to the limitations of the GPU partitioning we used on the natural environment, we had to run GPU-based workloads on a VM. In this situation, we assume workloads requesting a GPU run on its VM with its vGPU instance while the rest of the workloads run on the host itself.

We incorporate YOLO [70] as an accelerated workload for the evaluation. Thus, we have four kinds of workloads:

1. Bandwidth-bound: represents workloads that are sensitive to bandwidth, so multiple concurrent workloads running in the same device may impact performance if bandwidth capacity is exceeded. We use our work on SMUFIN[58][7], to model the behavior of this workload and the results presented in [9]. This model is then fed to the placement policies to assess how to allocate resources.
2. Capacity-bound: represents workloads with significant storage capacity and relatively low bandwidth requirements. Unlike bandwidth-bound workloads, these do not perceive any performance degradation from sharing the device with other workloads, as long as the demanded capacity does not exceed the available capacity. A real-world example of such would be TPCx-IoT [20], a TPC benchmark that attempts to emulate an Edge Computing scenario. It provides the mentioned characteristics: it uses NVMe mostly for storage and performing random reads on the data.
3. Compute-bound workload: we emulate the situation where we have CPU-consuming workloads that do not require NVMe usage. We introduce this synthetic workload to emulate situations where workloads not using the NVMe might prevent other workloads from using them in non-disaggregated scenarios. It could be any compute-bound real-world workload, for example, mathematical computations of weather forecasting.
4. GPU-bound workload: represents workloads accelerated through GPU. In this paper we focus on the case of YOLO[70] . YOLO is an object-detection approach. It frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network directly predicts bounding boxes and class probabilities from full images in one evaluation. YOLO has become a widely recognized object-detection workload with over 300 citations.

Table 5.1 describes our workloads characteristics.

Similarly to what we had done in our previous work, we established three scenarios defining the type of workloads that arrive at the data center:

- **Mixed workloads**: represents the situation when there is a roughly an equivalent mix of workloads requesting NVMe and the use of GPU. It contains a small portion of compute-intensive workloads as well.
- **GPU-intensive**: represents the situation when most of the workloads are GPU-accelerated while keeping a small amount of NVMe-dependent and compute-intensive workloads.

– **NVMe-intensive**: we emulate the situation when most of the workloads are NVMe-intensive, mostly bandwidth-intensive, and some capacity-based. There is a small portion of compute-intensive workloads as well as GPU-only.

The distributions of workloads in each of these scenarios are described in table 5.3 while the default simulations parameters are summarized in table 5.2. The idea behind the distributions and scenarios is to try to cover all possible cases, those that can potentially benefit more from partitioning (gpu-intensive) and those that can't and intermediate scenarios. Optimally one would like to cover all possibilities, but a simple calculation shows there are a hundred million possible combinations reducing our options to simulate everything. However, we consider the distributions shown allows us to infer the intermediate ranges. With the scenarios defined, we explain the evaluation results using our simulator.

### 5.4.1 *Performance of vGPU vs physical GPU*

First, we evaluate whether running YOLO on a VM or the host poses a significant performance difference when running on a CPU. And then, we repeat the experiment by running it on GPU, comparing a physical GPU on the host versus splitting the GPU into a vGPU, and then running the workload inside a VM with its vGPU. The results are shown in figure 5.4. We can observe how when running under CPU. The VM introduces a significant performance degradation of almost 2x worse. However, when running on GPUs, not only is there not performance degradation, but even it slightly improves results. The explanation for this is unclear. We detected during the experiments that GPU temperature was too high on occasions to the limit it shut down due to a fail-safe mechanism. When that did occur, there was a significant performance degradation on the GPU. We enabled a few ventilation improvements to avoid that situation, although the temperature still eventually increased. A potential explanation could be better utilization of the GPU when physically attached, which provoked a temperature rise due to insufficient ventilation of our workstation. The GPU is designed to be placed within a rack, and therefore it does not have its own ventilators. We put it on a workstation and added external ventilators to it. However, the performance of such a ventilation system can't be expected to be the same as when inside a rack. We did a couple of additional experiments when the temperature was low and observed only slightly worse performance than with vGPU. Thus, we assume this might be the cause. Even though it will require more in-deep analysis to explain the situation, which is also hard to do due to the use of a proprietary implementation of vGPU.

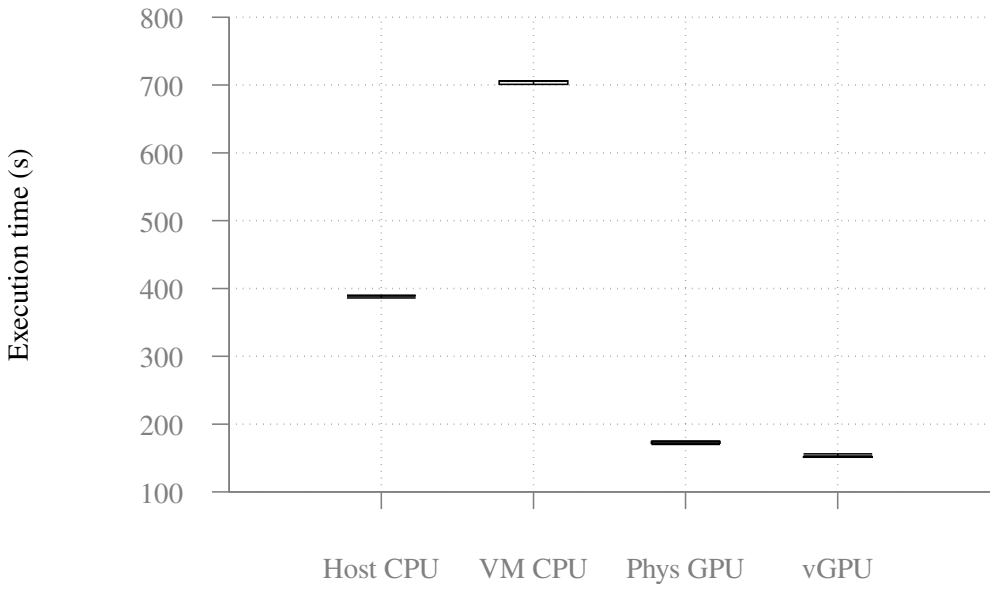
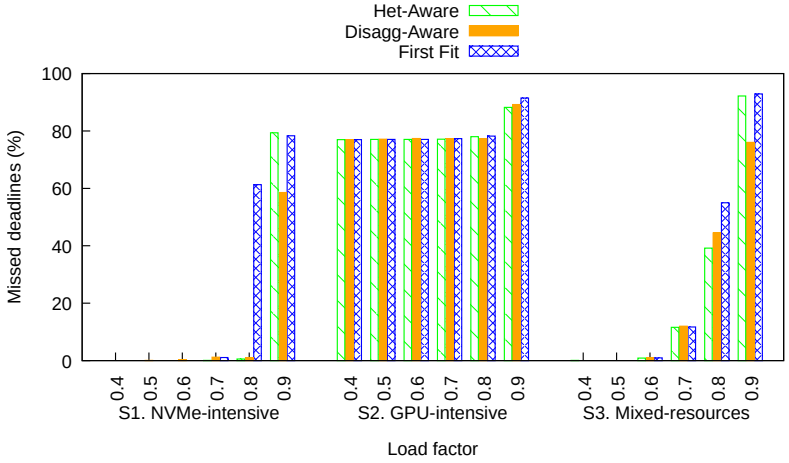



Figure 5.4: YOLO runs on different setups: on host and VM CPU-only, and using the physical GPU both on host and on a VM. The figure depicts the quartiles and standard deviation out of 6 runs. The points indicate the medians.

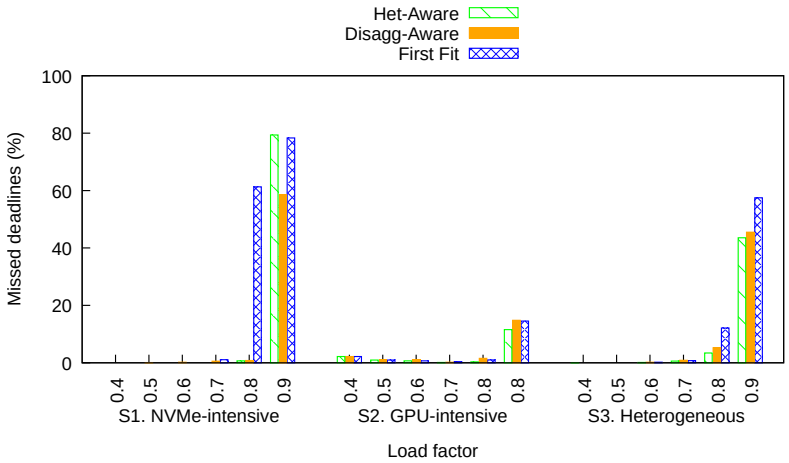
On the other hand, it is interesting to observe how acceleration using GPU improves the performance of the workload slightly more than 2x when bare-metal, and about 3.5x when running vGPU. Therefore we conclude that our dataset YOLO setup is appropriate for analyzing the impact of a data center with accelerated workloads, as it indeed benefits from acceleration.

About the splits of vGPU, the NVIDIA T4 only allows up to 4 splits. Each GPU has its characteristics and allowances for splits. The possible configurations were either not to split at all, a half-split, or 4-splitted. The election of such splits impacts the amount of memory each vGPU posses. In our case, we observed that the memory requirements for the dataset we chose on YOLO were not too high, allowing us to explore a good enough scenario.

The already introduced figure 5.3 shows the performance of YOLO under different amounts of concurrent runs. In each of the cases, the GPU is divided into 4. It is clear the performance degradation scales exponentially from 3 runs. However, the degradation between 3 concurrent runs and 2 is not excessive, and thus it still grants possibilities where it may be a good decision to share it with 3 simultaneous workloads to deliver a better overall SLA. However, this situation will hardly happen when sharing it with 4 instances; in that case, the performance degradation is about 3.5x worse.




(a) Without GPU splits




(b) With GPU splits

Figure 5.5: Deadlines missed according to policy and scenario

### 5.4.2 *Impact of splitting GPU*

We have explained how the main difference in this contribution regards disaggregated resources is that we can split a disaggregated GPU and share its splits across many nodes simultaneously. It is important to make particular emphasis on the fact that those splits and posterior sharing are made in a workload unaware fashion. So the workloads are not aware of it, and neither needs any modification to leverage it. It is done by the orchestrator alone. To evaluate the impact of having the ability to split GPU into smaller pieces, we compare an scenario with one unsplittable GPU (figure 5.5a and another scenario with an splittable GPU (figure 5.5b).

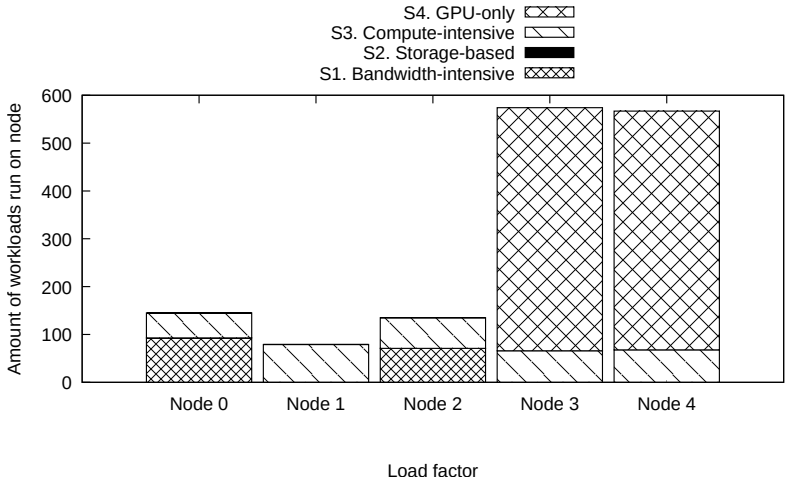
It can be observed with the same amount of physical GPU. Many fewer deadlines are missed in all scenarios when GPU can be split. This is due we can use all available nodes simultaneously for the workloads requiring a GPU. Otherwise, nodes must wait to complete before switching the GPU to another one if needed. It is as well clear that when most workloads require the use of GPU, and we have only one that at the same time is not splittable, no policy can deal with it. There is an evident lack of resources that can't be shared.

To go deeper into this, we represent the workloads' distribution (by workload kind) across nodes during the execution in figure 5.6. We explore all three scenarios: physically-attached resources (both GPU and NVMe), disaggregated resources but not splittable GPU, and splitting GPUs.

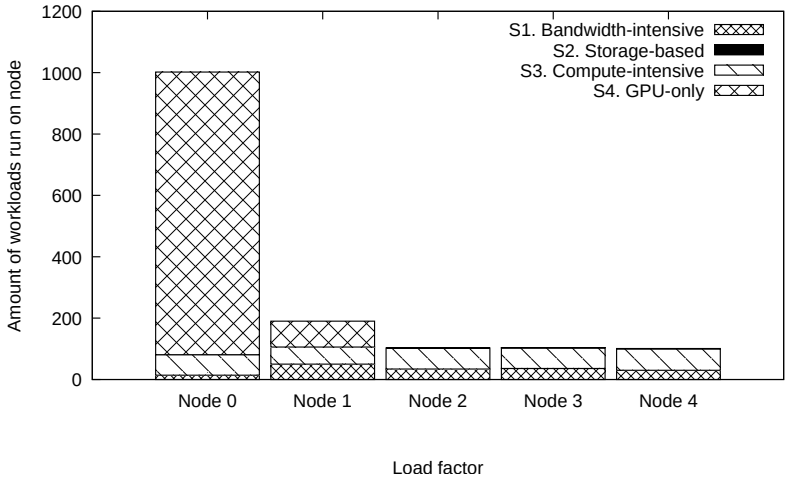
We can appreciate how there is a precise distribution of GPU workloads when GPUs can be split (5.6c), and this situation barely happens when GPUs cannot be split (5.6b). Notice in the latter case that only 2 nodes were running on GPU would appear. However, there is a single GPU on the system. Thus, the orchestrator decides to run most workloads on a single node and leave the remaining nodes for the other workloads. The situation is even worse when resources are physically attached (5.6a). In that case, the workload is forcefully assigned by type to specific nodes, as only a few nodes possess the requested resources.

### 5.4.3 *Heterogeneous disaggregated resources*

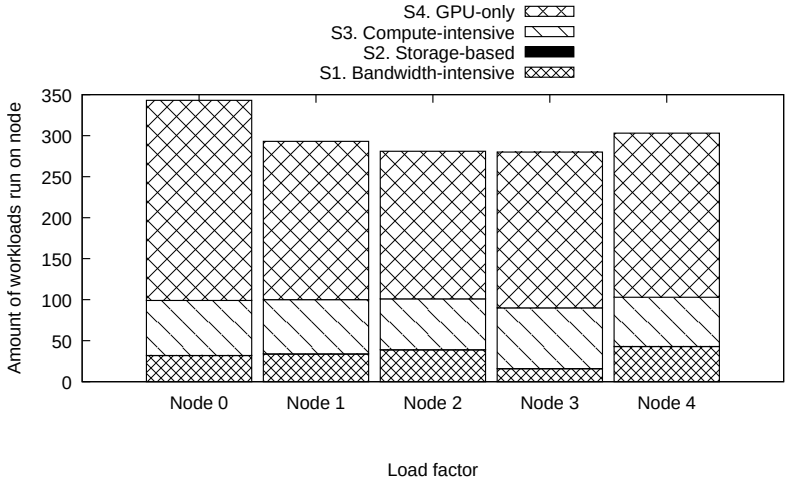
Finally, we evaluate the impact of our policy when dealing with a heterogeneous set of resources. As we mentioned earlier, on edge computing environments edge nodes are typically small. Thus, it is not often that they have much resources in it. Benefiting from disaggregation one could place some bigger inermediary nodes with the resources. However it is still hard for data centers to acquire specialized resources such as GPU or NVMe, among other resources we could consider. Thus, it is fair to assume the amount of such resources will be limited compared to the amount of computational nodes. For this reason, we developed a policy that considers that and tries to allocate workloads' cores in the optimal nodes to maximize the possibility of using the available resources. To evaluate our policy, we must as well adapt our simulation scenario. So far, we have had plenty of NVMe available (10) to deal with the demands. The load factor on NVMe and GPU was never too high and, in any case, higher than the CPU load factor. However, in a regional cloud data center or edge node will not always be so many resources. The target of this policy is to deal with scenarios where the availability of disaggregated specialized resources is limited and yet be able to deal with the SLA




(a) Physically attached scenario




(b) Without splitting GPU




(c) Splitting GPU

Figure 5.6: Workloads distribution on physically-attached, unsplittable GPU and splitting GPU scenarios

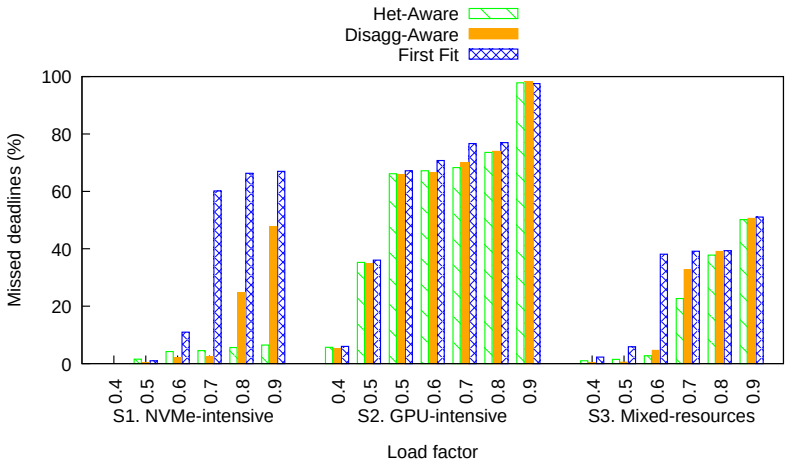



Figure 5.7: Deadlines missed according to policy on heterogeneous scenario. No GPU partitioning. NVMe load factor.

required by workloads. Our scenario contains a pool of 3 NVMe nodes and 1 GPU. As per load factor, we consider the maximum load factor between NVMe-bandwidth load factor and NVMe-capacity load factor. The capacity and bandwidth load factor are the proportion between the requested bandwidth (or capacity) and the available bandwidth. The availability is calculated assuming a fat node, that is, an aggregation of all the nodes and resources into a single one. Then, each time a workload arrives, it is assumed it runs as long as the fat node has resources available and runs for a period of time equal to running in isolation under ideal conditions. When a workload finishes, it frees the fat node resources. Figure 5.7 shows the deadlines missed per policy on our three scenarios. Using this the layout of limited resources and the NVMe load factor. On the other hand, the scenario we design does not allow GPU partitioning. GPU partitioning is so flexible that it essentially equals removing one resource. When splitting GPU, we can allocate a portion of it to all compute nodes and then handle the workloads allocating accordingly to NVMe devices. Therefore, in practice is like we only have to handle NVMe resources. However, we want to prove what happens when dealing with heterogeneous resources. For this reason, we assume GPUs cannot be split, but they are disaggregated.

We can appreciate how in the case of NVMe-intensive workloads are the majority. Our policy outperforms all others. Even on the tightest of the scenarios (0.9 load factor). Thus, properly allocating computational nodes when dealing with a heterogeneous and limited set of resources is of utmost importance. On the other hand, all policies perform equally badly in a GPU-intensive scenario. This is due we only have a single GPU that cannot be split and a significant amount of GPU-demanding workloads. This causes an under-provisioning of the data center. Thus nothing can be done to deal

with this situation. If more GPUs are added or partitioning is enabled, disaggregation-aware policy and heterogeneous-aware would perform roughly equal. This is due to GPU-intensive workloads being handled in the same fashion, and there is a dominant presence of such workloads. On the other hand, in a mixed scenario, tight situations (0.8 and 0.9 load factors) perform roughly equal. However, it improves significantly on an average scenario of 0.7 and 0.6 load factor. Although in this mixed scenario, it could be expected to perform better, we are in the same situation as in a GPU-intensive scenario. We have many GPU workloads and only a single GPU for them. Thus, our policy cannot do much to deal with this situation. However, if the load factor is not excessively high (0.7) it can deal with it relatively better than other policies.

### 5.4.4 *Overhead of GPU disaggregation*

As mentioned in this chapter, we assume GPU is disaggregated over the network. We stated previous work showing how this is feasible, and it is currently implemented and put in practice. However, due to our lack of resources, we could not make experiments with the GPU disaggregated. Instead, they were physically attached. One could say, however, that disaggregation might introduce some penalties in performance. In our previous chapter, we have seen that disaggregating NVMe over fabrics does not introduce any significant penalty. Performance differences compared to physically attached are under 1%. However, we have implemented an overhead factor to degrade the performance of GPU workloads in our simulator. Then we re-run the experiments on the heterogeneous scenario to see the impact of the overhead on the results. Since we have many scenarios and load factors for each, showing different overhead factors for them all would result in a rather large and hard-to-read figure. So we have picked a load factor of 0.8. As we appreciated in this scenario, our heterogeneous-aware policy performs better than disaggregated-aware. In figure 5.8 we show the experiments with different overhead factors for this load factor and the comparison among the three policies for our three scenarios.

It is clear that not even when the overhead reaches 5%, the results show almost no difference. For the GPU-intensive scenario, there is a tiny difference in higher overheads. However almost unnoticeable in the figure. Thus, our previous conclusions remain valid regardless of this overhead. Moreover, notice disaggregation of NVMe did not result in such big impacts. The Infiniband network has three orders of magnitude latency below PCIe v3 (the one used for our workstation and GPU). Thus it does not seem reasonable to assume the penalty for disaggregating GPU is that big.

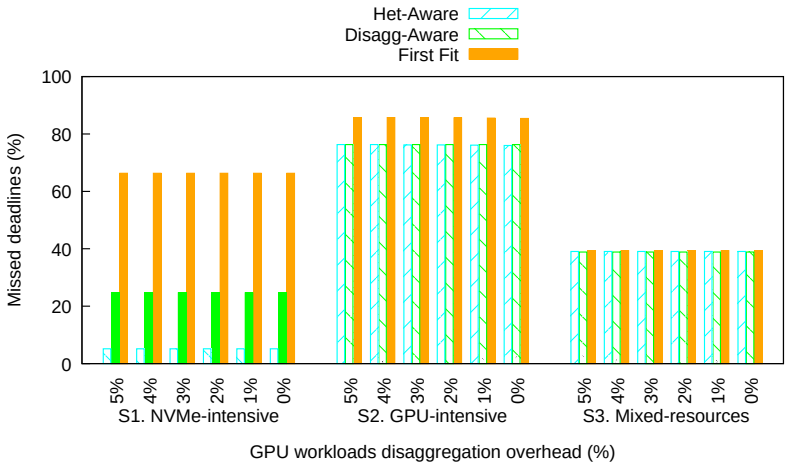



Figure 5.8: Deadlines missed according to policy on heterogeneous scenario, on different ovehreads for a NVMe-badnwdith load factor of 0.8. No GPU partitioning.

## 5.5 RELATED WORK

The work in [3] presents a scheduler proposal to allocate workloads using multi-GPU awareness of the GPUs topology. Given a cluster with compute nodes possessing many GPUs, deciding which GPUs will be allocated to a workload have a significant performance impact when the allocated GPUs are not interconnected with each other with a high-bandwidth connectivity or far from their respective CPUs. The scheduler proposal attempts to make GPUs allocation aware of it to maximize SLA provided to workloads. This work is complementary to ours. Our work provides flexibility in the terms that GPU can be reallocated to nodes where cores are available so the workload can run, regardless of the physical location of the GPU. On the other hand, our work allows us to share the same GPU across different workloads, especially relevant for cases when a workload does not require the whole GPU. The presented work attempts to maximize the performance of workloads requiring many GPUs to run. Thus, the problems solved are different but complementary, as in the cloud-edge datacenter workloads requiring multiple GPUs to run will also happen. Combined together with the possibility of disaggregating mutiple GPUs to different nodes to be accessed by the same workload might be a very interesting research. It is left as future work to explore this possibility.

Following this work, the authors presented [12]. They present an orchestrator for disaggregated GPUs. One key difference in their work is that they enable GPU disaggregation via rCUDA, which, as we said, does not expose the GPU but it only allows a remote client to run the instructions on it. Thus, they had the limitation that they could not share the device across workloads. They found the issue to be the lack of separate

address spaces on pre-Volta architectures. In Volta architectures, they still faced the issue that a single exception triggered by any client would terminate any other client sharing it. In our work, this situation is solved through the usage of vGPU technology which allows us to share portions of GPU as independent entities and avoid these complications. Combined with a proper disaggregation such as Falconwitch [23], we can enable disaggregated and shared GPUs. Another critical difference in their work is that they attempt to solve the issue of the added network latency due to disaggregation using rCUDA. On the other hand, the proposal is based on this latency, reducing it through a flow-network model and achieving good QoS thanks to the added flexibility disaggregation provides. Thus, we find this work complementary, as extra latency and reducing it are very relevant. Combining it to where the cores are available to run the computational part of the workloads can be critical to achieving an even better overall datacenter QoS.

[44] presents a cloud heterogeneous scheduler involving accelerators as co-processors. In their work they attempt to decide on which kind of Amazon EC2 instance to place workloads when those instances may or may not have accelerators. There are workloads that finish significantly earlier when using an accelerator and thus may be worth using a more expensive instance rather than a regular one without accelerator. For this they adapt a cost-function based scheduler considering this characteristic. As oppositoin to our work, resources are physically attached and available, and it is a matter of deciding the cheaper placemennt. Our work, however, considers the heterogeneity of resources form the point of view on where to place the accelerators with respect to other disaggregated resources in such a way that maximizes the flexibility of the datacenter.

Other works relating to heterogeneous resources often place their focus on the heterogeneous set of networks on cloud-edge infrastructures. Examples of this can be found in [50], where they analyze the heterogeneous set of wireless networks that can be found in edge and IoT contexts. They attempt to minimize the interferences induced by the different characteristics of the networks that will connect an edge device to its resources. For this, they modify existing swap matching algorithms to adapt to these characteristics. Finally, they evaluate their proposal on a simulated scenario achieving higher efficiency. [25] proposes a modified bat algorithm to maximize throughput on heterogeneous networks. Other than the fact that the paper focuses on network communications instead of resources, there is also a key difference in that we focus on meeting deadlines rather than maximizing throughput. [83] proposes a solution to the fact that when different service providers share the same backhaul network, a heterogeneous network appears. In that situation, the heterogeneous networks induced a mismatch in QoS. A downstream resource management solution is proposed, which

assigns a specific wavelength to each service provider. Then a QoS mapping algorithm is proposed to solve the QoS mismatch problem. The scenario is different from the one proposed as its focus is on network communications rather than on computational resources. [50][29] are works solving issues in the same directions.

On the other hand, [65] presents a work-in-progress method to apply game theory algorithms to allocating heterogeneous resources. Although the work's idea is to consider heterogeneous resources as described in their paper, there are two main differences. On the one hand, in the experiments, they only consider CPU cores with different frequencies (i.e., performance). On the other hand, they assume workloads can run on any of the resources. Thus, the workloads are computing for the whole set of resources, but their behavior will be different in performance or power consumption. Moreover, they consider the resources to be physically attached to the machine. Thus, they decide the allocation on a node once the workload arrives. The main differences with our work are, therefore: (a) workloads request a specific type of resources and can't be run on other resources, (b) resources are disaggregated. Therefore they do not be to be all placed on the same computational node.

Finally, [78] presents a modification of a multi-purpose and known scheduler, DRF, and adapts it to deal with heterogeneous resources. Then they implement it on a YARN scheduler. They modify DRF in order to ensure there is no resource starvation when dealing with heterogeneous resources (FPGA, GPU, etc.), a common situation under DRF. The main differences with our work are on the one hand, that we are dealing with fragmentation that occurred because of resource disaggregation. In their work, this situation can't happen as they consider resources physically attached. On the other hand, DRF and the modified proposed DRF goal is to maximize throughput, but they do not consider deadlines. In our context, however, deadlines are essential as we deal with real-time workloads. We understand real-time in this context as they need to be completed within a specific time. For instance, in genomics, workloads used for personalized medicine would not be acceptable to have the results completed after a certain time, as the patient could develop a severe condition.

## 5.6 CONCLUSIONS

In this chapter we presented an orchestration and resource provisioning set of policies for heterogeneous multi-resource computing environments, leveraging disaggregation for software-defined infrastructures. Such policies attempt to disaggregate resources by partitioning them and assigning them to computing nodes, allowing sharing of

resources that otherwise would be exclusively assigned to a node until they are released.

Through this study and evaluation, we have seen that vGPU implementation grants an extra level of flexibility. This added flexibility allows achieving the same result with half the resources. With 1 GPU using vGPU splits, we can achieve the same result as two unsplittable physical GPUs. Moreover, we analyze the impact of a policy considering heterogeneous resources when the specialized resources (NVMe, GPU) are limited. In such circumstances, our policy has proven to reduce the missed deadlines up to 86% compared to our previous policy based on only disaggregated NVMe resources and up to 90% compared to a trivial First Fit policy.

It is left as future work to add other co-processors such as FPGA or other devices with higher complexity. On the other hand, in the context of edge-cloud, data from edge devices must be moved from the edge to the cloud for complex computations. In those scenarios, it is left as future work to handle the distance between data location and the cloud to manage QoS properly. In the disaggregated scenario, QoS could also be managed by tuning network parameters. In such an environment, some edge devices could have better SLA when accessing remote resources than others.

## 5.7 PUBLICATIONS

The contents of this contribution are summarized in the following publication:
*[Submitted]*. Aaron Call, Josep Lluís Berral, and David Carrera. "Orchestration of Disaggregated Accelerators in Heterogeneous Resource Environments". In: IEEE Transactions on Network and Service Management.

# 6

# CONCLUSIONS AND FUTURE WORK

## 6.1 SUMMARY OF RESULTS

In this thesis, we have shown how the data centers paradigm is shifting from traditional physically attached resources to a disaggregated model. This shift translates into a more efficient resource usage, either via requiring fewer resources to achieve the same result or by increasing the number of workloads which SLA can be fulfilled with the same resources under tight situations. This thesis demonstrates the different components needed to manage the disaggregated data center, from workload characterization to resource allocation policies leveraging that knowledge. Finally, it incorporates disaggregation of accelerators into the mix to demonstrate how handling a heterogeneous set of resources can be critical to achieving good results. There is literature, although limited, on each of the topics individually, but there is no literature covering the whole picture, from workload characterization to effective resource management.

### 6.1.1 *Disaggregating Non-Volatile Memories Towards Efficiency on Throughput-Oriented Workloads*

In the first contribution of this thesis we present how disaggregation of NVMe can allow to share a resource with more workloads than previously possible. We characterize SMUFIN, a genomics workload, as an example case of a throughput-oriented workload. Through our analysis we discover the reason behind this behavior is the elimination of memory sharing, which caused a performance degradation when multiple workloads were running on the same machine. Consequently, resource sharing ratio increases. Moreover, we aggregate several resources into a single unit, with bigger bandwidth and capacity, resulting into a performance increase on the workloads, as well as allowing to share with even more resources.

### 6.1.2 *Workload-Aware Placement for NVMe-Based Disaggregated Data Centers*

The second contribution expands the area of analysis to the whole data center. This contribution aims to properly orchestrate disaggregated resources into the data center. First, we expand the analysis area from genomics workloads to a broader area, including edge and HPC workloads. Then, we show how leveraging the knowledge of workloads' behavior under disaggregation is critical to properly managing the data center. The knowledge gathered on the first contribution is summarized into a performance model that is then fed into a resource allocation policy. This policy attempts to allocate the resources so that workloads' benefiting from resource sharing and composition can do so by over-provisioning resources if the model predicts a performance improvement. Additionally, we introduce the concept of system fragmentation, which indicates that attaching a resource to a particular computational node can have a potential benefit or prejudice the overall data center performance. A second policy minimizing such fragmentation is designed leveraging this knoweldge. Finally, the contribution analyses the behavior of both policies to introduce our proposed disaggregation-aware policy to handle resources in a disaggregated environment appropriately. Our results show a performance improvement when disaggregating resources of up to 49% fewer deadlines missed. When comparing our workload-aware policy to a traditional policy, both on the disaggregated scenario, we can show a reduction of up to 100% deadlines missed in relaxed scenarios (less than 0.7 load factor). Thus, in this contribution, we improve the efficiency of the data center by improving the overall QoS of workloads.

### 6.1.3 *Accelerators Partitioning and Orchestration for Heterogeneous Disaggregated Resources*

The third and final contribution adds accelerators into the mix. While on the first two contributions, we were dealing only with NVMe devices, now we are incorporating GPU. On the one hand, this contribution adds accelerated workloads into the mix, and on the other adds a heterogeneous set of resources. Moreover, when disaggregated GPU and NVMe do not act in the same way, each has its characteristics. On the one hand, this contribution shows how dealing with this mix appropriately improves the performance of the data center. On the other hand, we show the specific characteristics of GPU. GPUs provide an extra level of flexibility through partitioning. We can partition GPU a memory so that two workloads sharing it will not conflict. Moreover, if we disaggregate a partition, we can share the same GPU across all nodes. Thus, it is as if we had a fat node in terms of CPU cores and memory, the resources that were not

disaggregated. They continue not to be disaggregated, but now, when a workload is partially using a GPU, the left area of it can be used by another workload even if there are no cores available on that specific node, as we can attach it simultaneously into another node with available cores. We still may be in the situation where no cores are available, but it is much less likely with this extra level of flexibility. This contribution shows how the efficiency of the data center may be improved by achieving the same results with half the resources. That is, with a single GPU partitioned and disaggregated, we can achieve the same results as with a single GPU disaggregated but unpartitioned. Finally, this contribution shows a new concept of fragmentation that occurs when we have a heterogeneous set of resources. It shows how when a workload utilizing one of such resources has many compute nodes candidates to be attached to, deciding on which one to do so is critical and may prevent further fragmentation. The contribution shows how preventing it improves the performance of the data center compared with our disaggregation-aware policy and a traditional first fit policy.

### 6.1.4 *Orchestrating disaggregate datacenters*

The thesis contributions allow cloud-edge datacenters to incorporate disaggregated resources. Such disaggregation increases datacenter flexibility and thus allows an overall QoS. Moreover, new workloads have been introduced with policies adapted to those. Having workload-aware performance models on disaggregated datacenter helps better meet workloads' requirements in terms of resources and further enhances the provided QoS. On the other hand, being able to have policies for a heterogeneous set of workloads expands the possibilities of this thesis' approach and facilitates the incorporation of this work in real datacenters. In this thesis, we have shown that without such a workload-aware approach, disaggregation would still provide benefits, but it would not be able to increase the QoS as much as when there is knowledge of them. On the other hand, a policy was proposed for such cases when resources are limited. Especially relevant for those less available resources as they are expensive or not so commonly requested. That last policy allows for balancing the datacenter even in such circumstances properly. Thus, the third contributions together allow leveraging disaggregation on datacenters focused on cloud-edge environments.

## 6.2 FUTURE WORK

In this thesis we have explored characteristics of GPU and NVMe, however there are other resources such as FPGA. It is left as future work to incorporate those resources. Through

[6] we have kwnoeledge that FPGA can be leverage to reduce power consumption while running SMUFIN, despite increasing its execution time. This thesis policies have always been considering workloads' deadlines as a parameter of efficiency. However, another metric could be power consumption. In a context of climate change and awareness of environmental impact, reducing energy consumption is critical. Thus sacrificing deadlines to achieve less power consumption would be an interesting metric to consider. On the other hand, accelerated workloads can often run utilizing CPU only as well. Thus, considering whether to run on the CPU is worth it despite the suffered degradation or not is another element left for future work.

The first contirbution of this thesis has shown that some times when workloads run together on a single machine there are interferences. In particular we have seen this regarding SMUFIN and memory consumption. It introduced performance degradation however the workloads could run nonetheless. Our performance models have been so far evaluating their performance when no such interferences where in place. However we could incoporate a performance model to measure interferences among an heterogeneous set of workloads when sharing memory, cpu and other non-disaggregated elements. This would allow us to improve our allocation strategies by being able to decide to over-provision those resources as well.

## 6.3 PUBLICATIONS

This thesis publications are as follows:

- Aaron Call, Jorda Polo, David Carrera, Francesc Guim, and Sujoy Sen. "Disaggregating Non-Volatile Memory for Throughput-Oriented Genomics Workloads." In: *Euro-Par 2018 International Workshops, Revised Selected Papers* (Turin Italy). Jan. 2019, pp. 613–625. ISBN: 978-3-030-10548-8. DOI: 10.1007/978- 3- 030- 10549-5_48.[9]
- Aaron Call, Jordà Polo, and David Carrera. "Workload-Aware Placement Strategies to Leverage Disaggregated Resources in the data center." In: *IEEE Systems Journal 16.1 (2022)*, pp. 1697–1708. DOI: 10.1109/JSYST.2021.3090306. [10]
- *[Submitted]. Orchestration of Disaggregated Accelerators in Heterogeneous Resource Environments*. Aaron Call, Josep Lluís Berral, and David Carrera. In: IEEE Transactions on Network and Service Management.

# BIBLIOGRAPHY


[1] Moustafa Abdelbaky, Javier Diaz-Montes, and Manish Parashar. "Towards Distributed Software-Defined Environments." In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2017, pp. 703–706. DOI: 10.1109/CCGRID.2017.30.

[2] Francois Abel, Jagath Weerasinghe, Christoph Hagleitner, Beat Weiss, and Stephan Paredes. "An FPGA Platform for Hyperscalers." In: *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*. 2017, pp. 29–32. DOI: 10.1109/HOTI.2017.13.

[3] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. "Topology-Aware GPU Scheduling for Learning Workloads in Cloud Environments." In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: Association for Computing Machinery, 2017. ISBN: 9781450351140. DOI: 10.1145/3126908.3126933. URL: https://doi.org/10.1145/3126908.3126933.

[4] Jens Axboe, Allan D. Brunelle, and Nathan Scott. *blktrace*. Version 1.2.0-5. May 2022. URL: https://git.kernel.org/pub/scm/linux/kernel/git/axboe/blktrace.git/about/.

[5] H. Bannazadeh, A. Tizghadam, and A. Leon-Garcia. "Smart city platforms on multitier software-defined infrastructure cloud computing." In: *ISC2 '16: Proceedings of the 2016 Second International Smart Cities Conference*. Trento, Italy, 2016. ISBN: 978-1-5090-1846-8. DOI: 10.1109/ISC2.2016.7580770.

[6] Nicola Cadenelli, Zoran Jacksić, Jordà Polo, and David Carrera. "Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads." In: *Future Generation Computer Systems* 94 (2019), pp. 148–159. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.11.028. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18314183.

[7] Nicola Cadenelli, Jordà Polo, and David Carrera. "Accelerating K-mer Frequency Counting with GPU and Non-Volatile Memory." In: *Proceedings of the 19th IEEE International Conference on High Performance Computing and Communications (HPCC)* (Bangkok, Thailand). IEEE Computer Society. 2017. eprint: arXiv:1712.03254.

[8] Aaron Call. *Simulator for disaggregated data-center environments*. Version 1.0. Dec. 2019. URL: https://github.com/MortI2C/simulator.

[9] Aaron Call, Jorda Polo, David Carrera, Francesc Guim, and Sujoy Sen. "Disaggregating Non-Volatile Memory for Throughput-Oriented Genomics Workloads." In: *Euro-Par 2018 International Workshops, Revised Selected Papers* (Turin Italy). Jan. 2019, pp. 613–625. ISBN: 978-3-030-10548-8. DOI: 10.1007/978-3-030-10549-5_48.

[10] Aaron Call, Jordà Polo, and David Carrera. "Workload-Aware Placement Strategies to Leverage Disaggregated Resources in the Datacenter." In: *IEEE Systems Journal* 16.1 (2022), pp. 1697–1708. DOI: 10.1109/JSYST.2021.3090306.

[11] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. "An Overview on Edge Computing Research." In: *IEEE Access* 8 (2020), pp. 85714–85728. DOI: 10.1109/ACCESS.2020.2991734.

[12] M. Carneiro Do, J. Polo, D. Carrera, N. González, C. Yang, A. Morari, B. D'Amora, A. Youssef, and M. Steinder. "DRMaestro: orchestrating disaggregated resources on virtualized data-centers." In: *Journal of cloud computing* 10.article 22 (Mar. 2021), pp. 1–20. DOI: 10.1186/s13677-021-00238-6. URL: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-021-00238-6.

[13] Marcus Carvalho, Walfredo Cirne, Francisco Brasileiro, and John Wilkes. "Long-Term SLOs for Reclaimed Cloud Computing Resources." In: *Proceedings of the ACM Symposium on Cloud Computing*. SOCC '14. Seattle, WA, USA: Association for Computing Machinery, 2014, pp. 1–13. ISBN: 9781450332521. DOI: 10.1145/2670979.2670999. URL: https://doi.org/10.1145/2670979.2670999.

[14] F. Chen, R. Lee, and X. Zhang. "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing." In: *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. Feb. 2011, pp. 266–277. DOI: 10.1109/HPCA.2011.5749735.

[15] Jing Chen. "A Cloud Resource Allocation Method Supporting Sudden and Urgent Demands." In: *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*. 2018, pp. 66–70. DOI: 10.1109/CBD.2018.00021.

[16] Shuang Chen, Shay GalOn, Christina Delimitrou, Srilatha Manne, and José F. Martínez. "Workload characterization of interactive cloud services on big and small server platforms." In: *2017 IEEE International Symposium on Workload Characterization (IISWC)*. 2017, pp. 125–134. DOI: 10.1109/IISWC.2017.8167770.

[17] I-Hsin Chung, Bulent Abali, and Paul Crumley. "Towards a Composable Computer System." In: *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*. HPC Asia 2018. Chiyoda, Tokyo, Japan: Association for Computing Machinery, 2018, pp. 137–147. ISBN: 9781450353724. DOI: 10.1145/3149457.3149466. URL: https://doi.org/10.1145/3149457.3149466.

[18] B. Ciciani, D. Didona, P. Di Sanzo, R. Palmieri, S. Peluso, F. Quaglia, and P. Romano. "Automated Workload Characterization in Cloud-based Transactional Data Grids." In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum* (Shanghai, China). 2012, pp. 1525–1533. DOI: 10.1109/IPDPSW.2012.192.

[19] ExpEther Consoritum. *ExpEther*. http://www.expether.org/. 2019.

[20] Transaction Processing Performance Council. *TPCx-BB*. http://www.tpc.org/tpcx-bb/default.asp. 2017.

[21] FIO. *FIO*. https://fio.readthedocs.io/en/latest/. 2020.

[22] Inc. Facebook. *Facebook Disaggregated Rack*. http://goo.gl/6h2Ut. 2016.

[23] Falconwitch. *Falconwitch*. https://www.h3platform.com/product. 2022.

[24] Hongxiang Fan, Shuanglong Liu, Martin Ferianc, Ho-Cheung Ng, Zhiqiang Que, Shen Liu, Xinyu Niu, and Wayne Luk. "A Real-Time Object Detection Accelerator with Compressed SSDLite on FPGA." In: *2018 International Conference on Field-Programmable Technology (FPT)*. 2018, pp. 14–21. DOI: 10.1109/FPT.2018.00014.

[25] Min Feng, Li Guomin, and Gong Wenrong. "Heterogeneous Network Resource Allocation Optimization Based on Improved Bat Algorithm." In: *2018 International Conference on Sensor Networks and Signal Processing (SNSP)*. 2018, pp. 55–59. DOI: 10.1109/SNSP.2018.00020.

[26] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. "Network Requirements for Resource Disaggregation." In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. Berkely, CA, USA: USENIX Association, 2016. ISBN: 978-1-931971-33-1.

[27] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types." In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Boston, MA: USENIX Association, 2011, pp. 323–336.

[28] Giulio Giunta, Raffaele Montella, Giuseppe Agrillo, and Giuseppe Coviello. "A GPGPU Transparent Virtualization Component for High Performance Computing Clouds." In: *Euro-Par 2010 - Parallel Processing*. Ed. by Pasqua D'Ambra, Mario Guarracino, and Domenico Talia. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 379–391. ISBN: 978-3-642-15277-1.

[29] Fan Gu. "Research on the Resource Allocation Control in Heterogeneous Networks." In: *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*. 2019, pp. 225–228. DOI: 10.1109/ICCNEA.2019.00051.

[30] Zvika Guz, Harry (Huan) Li, Anahita Shayesteh, and Vijay Balakrishnan. "NVMe-Over-Fabrics Performance Characterization and the Path to Low-overhead Flash Disaggregation." In: *Proceedings of the 10th ACM International Systems and Storage Conference*. SYSTOR '17. Haifa, Israel: ACM, 2017, 16:1–16:9. ISBN: 978-1-4503-5035-8. DOI: 10.1145/3078468.3078483.

[31] Zvika Guz, Harry (Huan) Li, Anahita Shayesteh, and Vijay Balakrishnan. "NVMe-over-fabrics Performance Characterization and the Path to Low-overhead Flash Disaggregation." In: *Proceedings of the 10th ACM International Systems and Storage Conference* (Haifa, Israel). SYSTOR '17. New York, NY, USA: ACM, 2017, 16:1–16:9. ISBN: 978-1-4503-5035-8. DOI: 10.1145/3078468.3078483.

[32] Inc. Hewlett-Packard. *HP The Machine*. http://www.hpl.hp.com/research/systems-research/themachine. 2016.

[33] Intel. *Intel Rack Scale Design*. Tech. rep. 332937-004. Intel Corporation, 2016, p. 21. URL: http://www.intel.com/content/dam/www/public/us/en/documents/guides/platform-hardware-design-guide.pdf.

[34] Intel. *Intel Rapid Storage Technology*. http://www.intel.com/content/www/us/en/support/technologies/intel-rapid-storage-technology-intel-rst.html. 2017.

[35] Da-Cheng Juan, Lei Li, Huan-Kai Peng, Diana Marculescu, and Christos Faloutsos. "Beyond Poisson: Modeling Inter-Arrival Time of Requests in a Datacenter." In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Vincent S. Tseng, Tu Bao Ho, Zhi-Hua Zhou, Arbee L. P. Chen, and Hung-Yu Kao. Cham: Springer International Publishing, 2014, pp. 198–209. ISBN: 978-3-319-06605-9.

[36] K. Kambatla, V. Yarlagadda, I. Goiri, and A. Grama. "UBIS: Utilization-Aware Cluster Scheduling." In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2018, pp. 358–367. DOI: 10.1109/IPDPS.2018.00045.

[37] G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black. "Software defined infrastructures." In: *IBM Journal of Research and Development* 58 (2014). ISSN: 0018-8646. DOI: 10.1147/JRD.2014.2298133.

[38] K. Katrinis et al. "Rack-scale disaggregated cloud data centers: The dReDBox project vision." In: *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. Dresden, Germany: IEEE Computer Society, 2016, pp. 690–695.

[39] Amit Kawalia, Susanne Motameny, Stephan Wonczak, Holger Thiele, Lech Nieroda, Kamel Jabbari, Stefan Borowski, Vishal Sinha, Wilfried Gunia, Ulrich Lang, et al. "Leveraging the power of high performance computing for next generation sequencing data analysis: tricks and twists from a high throughput exome workflow." In: *PloS one* 10.5 (2015), e0126321.

[40] Hadi Khani and Hamed Khanmirza. "Randomized routing of virtual machines in IaaS data centers." In: *PeerJ Computer Science* 5 (Sept. 2019), e211. DOI: 10.7717/peerj-cs.211.

[41] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. "ReFlex: Remote Flash & Local Flash." In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '17. 2017. ISBN: 978-1-4503-4465-4. DOI: 10.1145/3037697.3037732.

[42] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. "Modeling Virtualized Applications Using Machine Learning Techniques." In: *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*. VEE '12. London, England, UK: Association for Computing Machinery, 2012, pp. 3–14. ISBN: 9781450311762. DOI: 10.1145/2151024.2151028. URL: https://doi.org/10.1145/2151024.2151028.

[43] Gunho Lee, Byung-Gon Chun, and H. Katz. "Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud." In: *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'11. Portland, OR: USENIX Association, 2011, p. 4.

[44] Gunho Lee and Randy H. Katz. "Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud." In: *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 11)*. Portland, OR: USENIX Association, June 2011. URL: https://www.usenix.org/conference/hotcloud11/heterogeneity-aware-resource-allocation-and-scheduling-cloud.

[45] Heng Li and Richard Durbin. "Fast and accurate short read alignment with Burrows–Wheeler transform." In: *Bioinformatics* 25.14 (2009), pp. 1754–1760.

[46] Huiping Li, D. Doermann, and O. Kia. "Automatic text detection and tracking in digital video." In: *IEEE Transactions on Image Processing* 9.1 (2000), pp. 147–156. DOI: 10.1109/83.817607.

[47] Ruiqiang Li, Yingrui Li, Xiaodong Fang, Huanming Yang, Jian Wang, Karsten Kristiansen, and Jun Wang. "SNP detection for massively parallel whole-genome resequencing." In: *Genome research* 19.6 (2009), pp. 1124–1132.

[48] Xiaorong Li, Terence Hung, and Sharad Singhal. "A user-centric dynamic cluster partitioning approach for HPC service optimization." In: *2009 IEEE 28th International Performance Computing and Communications Conference*. 2009, pp. 121–128. DOI: 10.1109/PCCC.2009.5403803.

[49] Ching-Huang Lin, Chien-Tung Lu, Ying-Hsien Chen, and Jung-Shian Li. "Resource allocation in cloud virtual machines based on empirical service traces." In: *International Journal of Communication Systems* 27 (Dec. 2014). DOI: 10.1002/dac.2607.

[50] Gang Liu, Hangsheng Zhao, and Dali Li. "Resource allocation in heterogeneous networks: A modified many-to-one swap matching." In: *2017 IEEE 17th International Conference on Communication Technology (ICCT)*. 2017, pp. 508–512. DOI: 10.1109/ICCT.2017.8359688.

[51] Y. Liu, N. Bobroff, L. Fong, S. Seelam, and J. Delgado. "New Metrics for Scheduling Jobs on Cluster of Virtual Machines." In: *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. 2011, pp. 1001–1008. DOI: 10.1109/IPDPS.2011.245.

[52] Yanbin Liu, Norman Bobroff, Liana Fong, Seetharami Seelam, and Javier Delgado. "New Metrics for Scheduling Jobs on Cluster of Virtual Machines." In: *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. 2011, pp. 1001–1008. DOI: 10.1109/IPDPS.2011.245.

[53] P. Lv, W. Liu, and J. Li. "A FPGA-based accelerator implementaion for YOLOv2 object detection using Winograd algorithm." In: *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*. Los Alamitos, CA, USA: IEEE Computer Society, 2020, pp. 1894–1898. DOI: 10.1109/ICMCCE51767.2020.00415. URL: https://doi.ieeecomputersociety.org/10.1109/ICMCCE51767.2020.00415.

[54] Jacob Machina and Angela Sodan. "Predicting cache needs and cache sensitivity for applications in cloud computing on CMP servers with configurable caches." In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. 2009, pp. 1–8. DOI: 10.1109/IPDPS.2009.5161233.

[55] V.Y. Mariano and R. Kasturi. "Locating uniform-colored text in video frames." In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. Vol. 4. 2000, 539–542 vol.4. DOI: 10.1109/ICPR.2000.902976.

[56] Paul Medvedev, Monica Stanciu, and Michael Brudno. "Computational methods for discovering structural variation with next-generation sequencing." In: *Nature methods* 6 (2009), S13–S20.

[57] David Meisner and Thomas Wenisch. "Stochastic queuing simulation for data center workloads." In: (Jan. 2010).

[58] Valentí Moncunill, Santi Gonzalez, Sílvia Beà, Lise O Andrieux, Itziar Salaverria, Cristina Royo, Laura Martinez, Montserrat Puiggròs, Maia Segura-Wang, Adrian M Stütz, et al. "Comprehensive characterization of complex structural variations in cancer by directly comparing genome sequence reads." In: *Nature biotechnology* 32.11 (2014), pp. 1106–1112. DOI: 10.1038/nbt.3027.

[59] NVIDIA. *NVIDIA Virtual GPU Positioning. Selecting the Right GPU for Your Virtualized Workload*. Tech. rep. NVIDIA Corporation, 2021.

[60] NVIDIA. *Telsa T4*. https://www.nvidia.com/en-us/data-center/tesla-t4/. 2022.

[61] NVMexpress. *NVM express*. http://www.nvmexpress.com. 2017.

[62] Minoru Oikawa, Atsushi Kawai, Kentaro Nomura, Kenji Yasuoka, Kazuyuki Yoshikawa, and Tetsu Narumi. "DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment." In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 2012, pp. 1207–1214. DOI: 10.1109/SC.Companion.2012.146.

[63] A. D. Papaioannou, R. Nejabati, and D. Simeonidou. "The Benefits of a Disaggregated Data Centre: A Resource Allocation Approach." In: *Proceedings of the 35th IEEE Global Communications Conference (GLOBECOM)*. 2016. DOI: 10.1109/GLOCOM.2016.7842314.

[64] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. "Practical Resource Management in Power-Constrained, High Performance Computing." In: *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '15. Portland, Oregon, USA: Association for Computing Machinery, 2015, pp. 121–132. ISBN: 9781450335508. DOI: 10.1145/2749246.2749262. URL: https://doi.org/10.1145/2749246.2749262.

[65] Lara Premi, Federico Reghenzani, Giuseppe Massari, and William Fornaciari. "A Game Theory Approach to Heterogeneous Resource Management: Work-in-Progress." In: *2020 International Conference on Embedded Software (EMSOFT)*. 2020, pp. 25–27. DOI: 10.1109/EMSOFT51651.2020.9244046.

[66] Megan J. Puckelwartz, Lorenzo L. Pesce, Viswateja Nelakuditi, Lisa Dellefave-Castillo, Jessica R. Golbus, Sharlene M. Day, Thomas P. Cappola, Gerald W. Dorn II, Ian T. Foster, and Elizabeth M. McNally. "Supercomputing for the parallelization of whole genome analysis." In: *Bioinformatics* 30.11 (2014), p. 1508. DOI: 10.1093/bioinformatics/btu071.

[67] Carlos Reaño and Federico Silla. "A Performance Comparison of CUDA Remote GPU Virtualization Frameworks." In: *2015 IEEE International Conference on Cluster Computing*. 2015, pp. 488–489. DOI: 10.1109/CLUSTER.2015.76.

[68] Carlos Reaño, Federico Silla, Gilad Shainer, and Scot Schultz. "Local and Remote GPUs Perform Similar with EDR 100G InfiniBand." In: *Proceedings of the Industrial Track of the 16th International Middleware Conference*. Middleware Industry '15. Vancouver, BC, Canada: Association for Computing Machinery, 2015. ISBN: 9781450337274. DOI: 10.1145/2830013.2830015. URL: https://doi.org/10.1145/2830013.2830015.

[69] Carlos Reaño and Federico Silla. "A Comparative Performance Analysis of Remote GPU Virtualization over Three Generations of GPUs." In: *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*. 2017, pp. 121–128. DOI: 10.1109/ICPPW.2017.29.

[70] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640.

[71] Frank Schmuck and Roger Haskin. "GPFS: A Shared-Disk File System for Large Computing Clusters." In: *Proceedings of the 1st USENIX Conference on File and Storage Technologies*. FAST '02. Monterey, CA: USENIX Association, 2002, pp. 2–15. URL: http://dl.acm.org/citation.cfm?id=1083323.1083349.

[72] H. Schneiderman and T. Kanade. "A statistical method for 3D object detection applied to faces and cars." In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*. Vol. 1. 2000, 746–751 vol.1. DOI: 10.1109/CVPR.2000.855895.

[73] Thamarai Selvi Somasundaram, Balachandar R. Amarnath, R. Kumar, P. Balakrishnan, K. Rajendar, R. Rajiv, G. Kannan, G. Rajesh Britto, E. Mahendran, and B. Madusudhanan. "CARE Resource Broker: A framework for scheduling

and supporting virtual resource management." In: *Future Generation Computer Systems* 26.3 (2010), pp. 337–347. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2009.10.005. URL: https://www.sciencedirect.com/science/article/pii/S0167739X09001551.

[74] Sivashankar and S. Ramasamy. "Design and implementation of non-volatile memory express." In: *2014 International Conference on Recent Trends in Information Technology*. Chennai, India, 2014. ISBN: 9781479978687. DOI: 10.1109/ICRTIT.2014.6996190.

[75] Konstantinos Tokas, Ioannis Patronas, Christos Spatharakis, Dionysios Reisis, Paraskevas Bakopoulos, and Hercules Avramopoulos. "Slotted TDMA and optically switched network for disaggregated datacenters." In: *2017 19th International Conference on Transparent Optical Networks (ICTON)*. 2017, pp. 1–5. DOI: 10.1109/ICTON.2017.8024739.

[76] Mihaela-Andreea Vasile, Florin Pop, Radu-Ioan Tutueanu, and Valentin Cristea. "HySARC2: Hybrid Scheduling Algorithm Based on Resource Clustering in Cloud Environments." In: *Algorithms and Architectures for Parallel Processing*. Ed. by Joanna Kołodziej, Beniamino Di Martino, Domenico Talia, and Kaiqi Xiong. Cham: Springer International Publishing, 2013, pp. 416–425. ISBN: 978-3-319-03859-9.

[77] Bin Wang, Xiaolin Chang, and Jiqiang Liu. "Modeling Heterogeneous Virtual Machines on IaaS Data Centers." In: *IEEE Communications Letters* 19.4 (2015), pp. 537–540. DOI: 10.1109/LCOMM.2015.2403832.

[78] Wenzhu Wang, Yusong Tan, Qingbo Wu, and Yaoxue Zhang. "Multiple resources scheduling for diverse workloads in heterogeneous datacenter." In: *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*. Vol. 01. 2015, pp. 563–567. DOI: 10.1109/ICCSNT.2015.7490810.

[79] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf. "Disaggregated FPGAs: Network Performance Comparison against Bare-Metal Servers, Virtual Machines and Linux Containers." In: *Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science*. 2016. DOI: 10.1109/CloudCom.2016.0018.

[80] Jonathan Wildstrom, Peter Stone, Emmett Witchel, and Michael Dahlin. "Machine Learning for On-Line Hardware Reconfiguration." In: Jan. 2007, pp. 1113–1118.

[81] Ming-Hsuan Yang, D.J. Kriegman, and N. Ahuja. "Detecting faces in images: a survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.1 (2002), pp. 34–58. DOI: 10.1109/34.982883.

[82] Sharrukh Zaman and Daniel Grosu. "A Combinatorial Auction-Based Mechanism for Dynamic VM Provisioning and Allocation in Clouds." In: *IEEE Transactions on Cloud Computing* 1.2 (2013), pp. 129–141. DOI: 10.1109/TCC.2013.9.

[83] Hong Zhang, Chuang Huang, Jing Zhou, and Liao Chen. "QoS-Aware Virtualization Resource Management Mechanism in 5G Backhaul Heterogeneous Networks." In: *IEEE Access* 8 (2020), pp. 19479–19489. DOI: 10.1109/ACCESS.2020.2967101.

[84] NVM express. *NVMe over Fabrics overview*. Tech. rep. NVM express, 2017, p. 7. URL: http://www.nvmexpress.org/wp-content/uploads/nvme_over_fabrics.pdf.

# DECLARATION

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

*Barcelona, May 2022*

Aaron Call Barreiro