Workload-Aware Placement Strategies to Leverage Disaggregated Resources in the Datacenter

Aaron Call*†, Jordà Polo*, David Carrera*, *Member, IEEE,*
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya - BarcelonaTECH, Barcelona, Spain
E-mail: {aaron.call,jorda.polo,david.carrera}@bsc.es

***Abstract*—Disaggregation of resources is a data center strategy that aims to decouple the physical location of resources from the place where they are accessed, as opposed to physically attached devices connected to the PCIe bus. By attaching and detaching resources through a fast interconnection network, it is possible to increase the flexibility to manage data center infrastructures while keeping the performance of the pooled and disaggregated devices. This paper introduces workload scheduling and placement policies for environments with disaggregated memories. These policies are driven by accurate pre-built performance degradation models. We focus on the use of Non-Volatile Memory to store data and/or to provide memory extensions. Following a software-defined approach, persistent memories are combined to provide higher capacity and/or bandwidth devices, or used by multiple workloads to increase the number of running workloads. Different combinations of workloads and associated soft-deadlines are used to evaluate the placement policies using a system simulator. Using a first fit policy, results show a disaggregated system can reduce missed deadlines up to 49% when compared to a physically-attached system. When our proposed policy with workload-awareness is enabled in a disaggregated system, missed deadlines can be reduced up to 100% (no deadlines missed).**

***Index Terms*—IO pooling, Model-Aware, Software Defined Infrastructrures, Workload Placement, Resource Disaggregation, Composability, Scheduling, NVMe, Orchestration**

I. Introduction

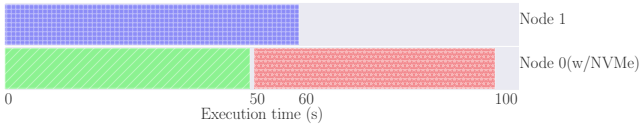
Traditional datacenter architectures comprise sets of mostly homogeneous compute platforms (also referred to as computing nodes or servers). Resources statically compose these platforms: compute, memory, storage, network fabric, and accelerators. They also access remote resources over the network, such as blob storage or network file systems. In a typical batch processing context, workloads are associated with a completion time goal or soft-deadline. They are managed by a scheduling subsystem that defines many queues or server pools used to place workloads admitted for execution.

Generally, the scheduling queues or pools provide access to a limited amount of pre-defined platform configurations, resulting in a homogeneous datacenter architecture in an attempt to make it easier to maintain and manage the infrastructure. However, homogeneous datacenters also lack adaptability, reconfigurability, malleability, and extensibility, which has an impact on the workloads.

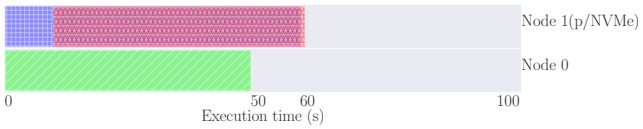
The conventional datacenter design target is to include enough infrastructure capacity to meet peak demands or to arrange the datacenter platform for bursting when needed while keeping in mind the total cost of ownership. Both of those methods depend on having enough foresight about what peak demands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

Datacenters based on Software-Defined Infrastructures (SDI) aim to disaggregate resources over a network fabric to enable increased management flexibility. On SDI, instead of pools of nodes, the pools consist of individual units of resources (storage, GPUs, FPGAs). When an application needs to be executed, SDI identifies the computational requirements and assembles all the required resources, creating a composite node. Then, the operating system is booted, and the application is instantiated. Unlike previous mechanisms to access remote resources (e.g. RDMA, or distributed filesystems), SDI resources are exposed as local to the software stack running on the composite node. Thereby, the SDI architecture virtualizes access to the pools of resources, exposing platform configurations that match the user requirements. Once the application finishes its execution, nodes are decomposed, and resources are placed back in their corresponding queues. Resource disaggregation also enables [1] increased malleability to exploit datacenter resources through sharing them across multiple workloads or by dynamically composing new ones with larger capacity or increased bandwidth.

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 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 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 has to wait until J1 finishes to run on node 0. In the second scenario, in Figure 1b, the NVMe is pooled, so either node can attach and share the resource on-demand. In this case, J3 can be placed in node 1 due it can attach 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.



(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.

Fig. 1. Timeline showing the execution of three jobs in two nodes with physically-attached NVMe (top) and disaggregated NVMe (bottom)

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

This paper uses a performance model for disaggregated workloads based on previous work on [1], 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 paper will attempt to answer 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. The main contributions of this paper are:

- Two novel model-aware placement policies: *maximize composition* and *minimize fragmentation*. The former enables resource sharing (at hardware level) and resource composition to optimize workload execution and is especially suited for IO bandwidth-bound loads. The latter aims to globally minimize resource fragmentation in the datacenter for capacity-bound loads.
- A novel *disaggregation-aware* scheduling policy that decides which of the two previously proposed placement policies should be applied at each time.
- A comparative analysis of benefits of resource disaggregation versus physically-attached resources.

II. Disaggregation Challenges

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

Figure 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. In this paper, we base our infrastructure on NVMe over Fabrics [2]. This technology allows exposing NVMe devices attached to a host node to target nodes. The target node is unaware that the NVMe is not physically attached to it and can use it as a regular block device, managing it as if it were physically-attached. It also

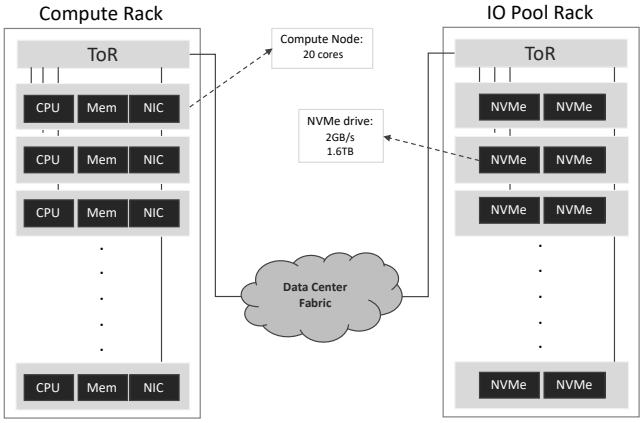


Fig. 2. Overview of datacenter with a disaggregated IO architecture, with pooled NVMe devices over the fabric

enables to compose several NVMe devices on the host node and expose the combination as a single NVMe device on the target nodes. This technology allows us to experiment with workloads and resource disaggregation in this paper.

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 paper 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 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 datacenter 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, and so current tools are not able to properly manage this kind of infrastructure yet. Morevoer 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.

The proposed infrastructure presents some challenges:

1) Resource sharing: when a node attaches a resource from any of the pools, it is not aware that it is disaggregated. It perceives the resource as physically-attached, and it enables simultaneous sharing of the same hardware resource across many nodes and workloads without any modification of their default behavior.

2) Resource composition: disaggregation also enables the possibility to attach multiple resources and expose them as if they were a single one. Thus, a node can believe it has a single physically-attached resource, when in fact has a set of resources remotely attached and composed together. This is what we define as resource composition. [1] discusses how the number of resources that can simultaneously share the device can be increased through this mechanism in some throughput-oriented workloads, even improve its performance.

3) Fragmentation: finally, resource pooling also introduces system fragmentation. Whenever a resource is requested, which one should be provided? A resource already attached to some other node, a composition of resources, or a resource not currently pooled to any other node? Our policies will explore different alternatives. Depending on the situation, it might lead to *fragmentation*. Figure 1 shows this situation. In scenario 1a, when J1 and J2 are scheduled, node 0 has its cores fully utilized, while its NVMe resources are completely free. Thus, the system has unused areas (NVMe) that are at the same time inaccessible. It has "split" half of the resources, fragmenting the system. Intermediate cases are relevant as well. Imagine in the same example, when J3 is finally scheduled, J3 utilized 10% of the cores, and it utilized the NVMe at 80%. It would be using most of the NVMe, however, at the same time leaving most of the available cores free. This situation implies that NVMe-dependent workloads will hardly be allocated in this node. 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 either most of the cores were as well used, or the NVMe was less utilized. Notice fragmentation is inherent of such systems and many times is impossible to avoid. However, our proposed policies and following evaluation will explore the usefulness of minimizing fragmentation.

Regarding the network, this paper evaluates InfiniBand transport and protocol only. The datacenter should have dedicated networking lanes between the pool of resources and the computing nodes to avoid potential conflicts of congestion with other users and data transmitted through the datacenter. Moreover, pools of resources should not have more device bandwidth than aggregated network bandwidth available for the pool. The datacenter can scale by adding more pools and InfiniBand links.

III. Strategies for SDI Orchestration

Based on the described challenges, this paper proposes two placement policies: 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. 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, such modification in the policies is low, as the strategy doesn't change. Hence the focus on homogeneous architectures.

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. This paper considers 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. 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 calculating its completion time based on the provided execution time.

A. 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, on the higher availability of bandwidth and capacity, the sharing ratio of resources without performance degradation also increases. 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 [1]. 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.

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
```

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, 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 on equation 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 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.

$$fitness = (bw_c - bw_j) + (sto_c - sto_j) \tag{1}$$

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 NVMes) 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 respect to smaller ones. Once increasing the composition size stops improving performance, the last parameter value is chosen (line 29). Line 24 also limits some parameters. A model may indicate workload will not improve its 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 its 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).

B. Minimize Fragmentation Placement Policy

This policy aims to minimize fragmentation, based on situations like shown in the introductory example in figure 1 and explained in section II-3. To minimize fragmentation we need to carefully consider the allocation. On which node the job will run in, and which NVMe resource will it use. If we are not careful enough it might be the remaining free space resulting of the allocation is such any incoming workload can use it. This is what we previously defined as fragmentation. As fragmentation is inherent on the system, we develop this policy to try minimizing, making a 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, will be a better placement than one leaving spare resources. With these considerations in mind, the ratio of fragmentation α between the NVMe bandwidth and capacity requested, plus the remaining cores after scheduling the workload, is estimated. This computation is made according to equation 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 α is defined as:

$$\alpha = \frac{1 - (\frac{bw_j}{bw_c} + \frac{sto_j}{sto_c})}{\frac{core_j}{core_n}} \tag{2}$$

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 of resources. However, when there is a need to make a new attachment among free resources, a new formula is needed. The goal of this paper 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 workloads' 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, we attempt to maximize the utilization of the physically-attached cores. To achieve this goal, we compute cores' slack using equation 3. This equation is similar to the concept of resource's slack provided by [3]. 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, β computes the percentage of utilized cores (as we made the reverse). Thus, the lower β more cores 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 II-3.

$$\beta = 1 - \frac{core_n}{tcore_n} \tag{3}$$

Algorithm 2 shows the pseudo-code for the policy:

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
```

Lines 2-15 are very similar to maximize composition policy. However, in this case, the α 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 3 (lines 21-22).

C. Disaggregation-aware Scheduling Policy

This paper 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 policy. This is a widely known policy that is good in 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 paper. 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 paper, 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.

$$\text{Policy} \equiv \begin{cases} \text{Max. Comp.} & LF_{bw} \leq 50\% \text{ and } LF_{cap} \leq 50\% \\ \text{Max. Comp.} & LF_{bw} \geq 70\% \text{ and } LF_{cap} \leq 70\% \\ \text{Min. Frag.} & \text{Otherwise} \end{cases} \tag{4}$$

IV. 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 datacenter. On the latter, further experiments are made to assess the different metrics.

A. Methodology

We built a simulator to evaluate the impact of the proposed strategies. The simulator emulates the architecture described in figure 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 the authors' 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 [4].

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

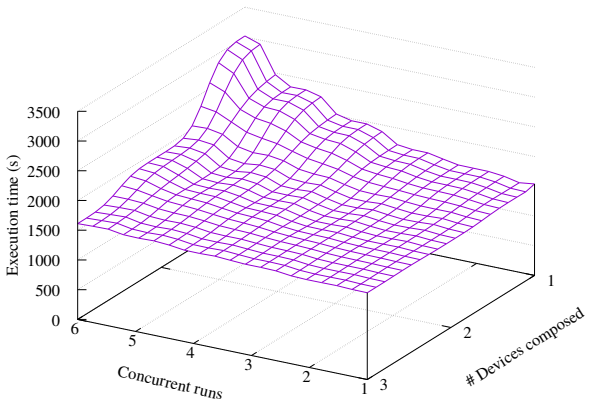


Fig. 3. Execution times for a bandwidth-bound workload showing how execution time evolves when running multiple SMUFIN instances: sharing a single device (1xNVMe), or sharing on composed nodes (2xNVMe, 3xNVMe).

TABLE I
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

performance if bandwidth capacity is exceeded. We use our work on SMUFIN [5] [6], to model the behavior of this workload and the results presented in [1], summarized in figure 3. The figure shows the execution times for real, SMUFIN runs on compositions of 1, 2, and 3 NVMe devices, with up to 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 [7], a TPC benchmark that attempts to emulate an Edge Computing scenario. It provides the mentioned characteristics: 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 paper.

The considered workloads utilize NVMe (bandwidth and capacity) and cores. Workloads specify their base performance (assuming their SLA is met). Table I 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 II 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, δ the deadline factor, and a_w the arrival time of the workload, then the deadline is computed according to equation 5:

$$deadline_w = e_w\delta + a_w \tag{5}$$

The deadline factor δ 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).

B. 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 paper 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 paper, we put particular emphasis on the CPU load factor

TABLE II
DEFAULT SIMULATION PARAMETERS ON THREE SCENARIOS

	Bandwidth-bound (%)	Capacity-bound (%)	Compute-bound(%)	Non-priority deadline factor	deadline factor	High-priority work-loads (%)	Number of nodes	Cores per node	NVMe band-width	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%								

(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 6 as the quotient between c_s and the sum c_j of all jobs J running the system.

$$LF_{cpu} = \frac{c_s}{\sum_{j \in J} c_j} \tag{6}$$

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 (λ 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.

C. 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.

This paper 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 III, IV and V 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.

TABLE III
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 IV
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.

- 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.

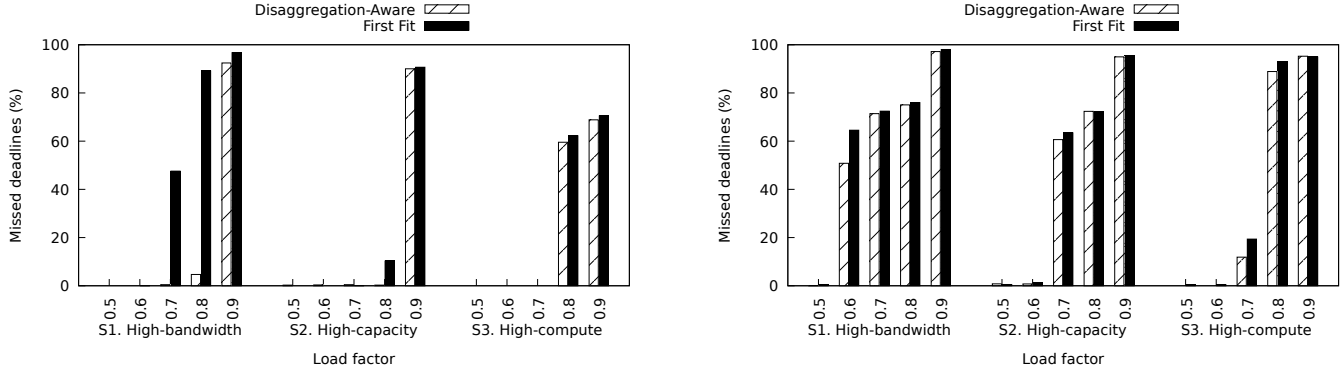
Figure 4 summarizes the results' for missed deadlines of the tables, comparing Disaggregation-Aware strategy versus First Fit. The y-axes show the strategies' missed deadlines for the different load factors (x-axes) of the three scenarios shown in the tables. On 4a disaggregated infrastructure is shown whereas 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 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 mitigating 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. Comparing 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 5 we present the amount of workloads run on each

TABLE V
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

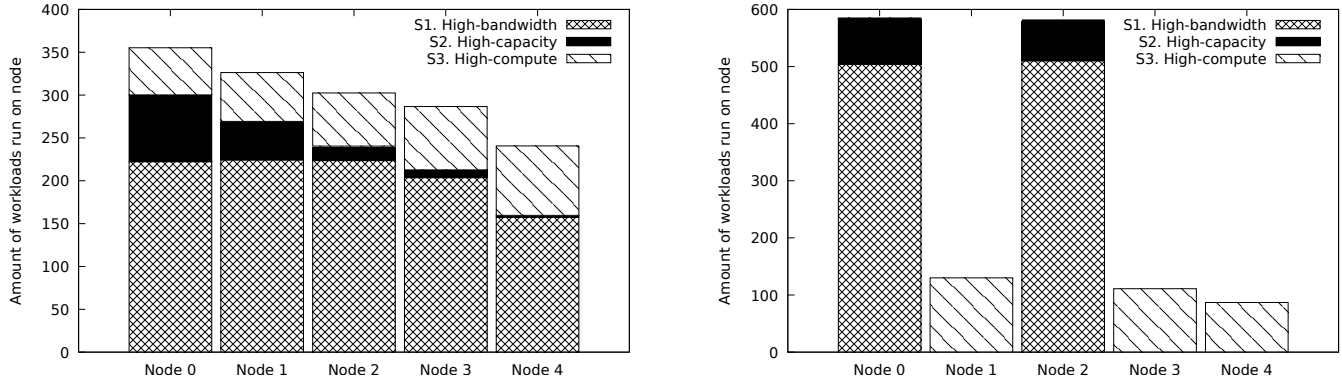
PA: Physically-attached.



(a) Disaggregated layout

(b) Physically-attached

Fig. 4. Performance of the strategies on different load factors for the three scenarios. Disaggregated and physically-attached infrastructures.



(a) Disaggregated layout

(b) Physically-attached

Fig. 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).

of the compute nodes by its kind, comparing the disaggregated infrastructure (figure 5a) and the physically-attached (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 pol-

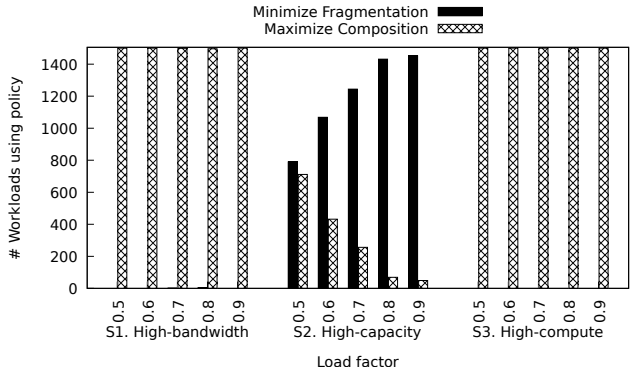


Fig. 6. Amount of workloads' resources allocated by each placement policy, load factor, and scenario.

icy is designed to shift between placement policies according to observed capacity and bandwidth utilization. To show this behavior, we depict in figure 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 behavior, as in such situations, minimizing fragmentation either slightly improves results (0.8 target CPU load factor) or does not worsen performance.

D. 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.

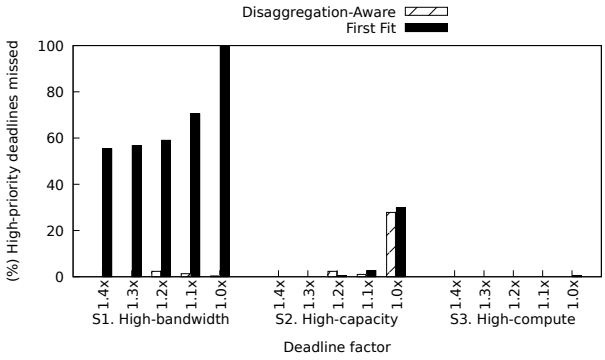


Fig. 7. High-priority workloads' deadline factor impact on high-priority deadlines missed. Results are categorized per each scenario and deadline factor.

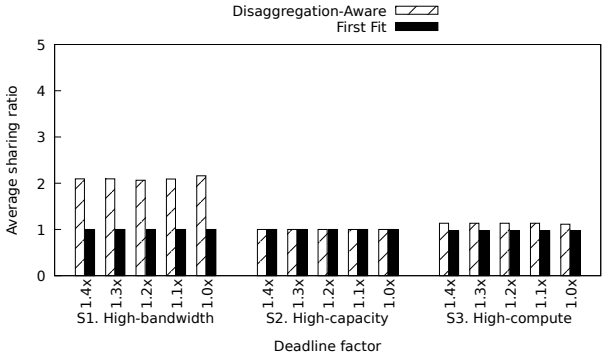


Fig. 8. High-priority workloads' deadline factor impact on workloads' sharing ratio. Results are categorized per each scenario and deadline factor.

Figure 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 8 depicts the average amount of workloads sharing the same resource. This figure allows us to observe how the sharing ratio increases when Disaggregation-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 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.

E. 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 10. On the x-axes the number of devices on the simulated system. The y-axes represent the percentage of missed deadlines. The results shown are for a fixed target CPU load factor of

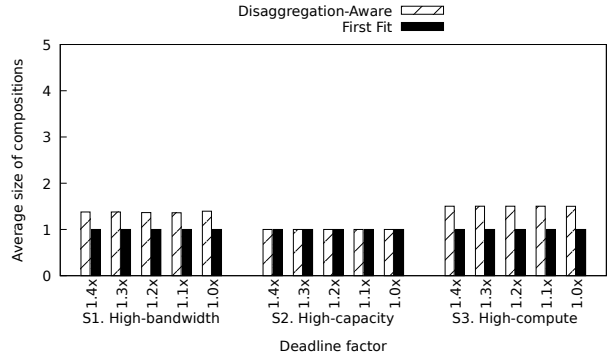


Fig. 9. High-priority workloads' deadline factor impact on average composition size. Results are categorized per each scenario and deadline factor.

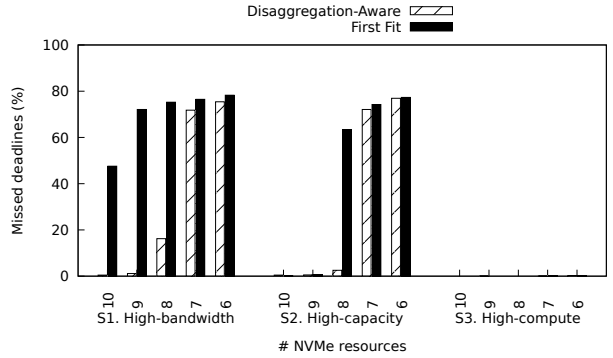


Fig. 10. Performance of strategies for different nunmber of NVMe devices on the three different scenarios. Results based on a target CPU load factor of 0.7.

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.

F. 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 [8]. 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 for the high-

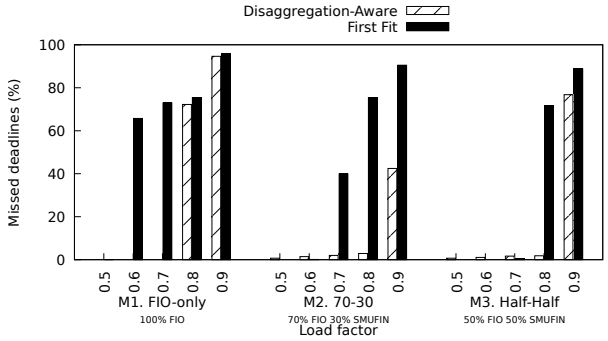


Fig. 11. Performance of the strategies on different load factors for three FIO-SMUFIN workloads' mixes. Disaggregated infrastructure.

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 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.

V. Related work

The idea of SDI has been increasingly studied in the literature over the past few years. In [9], the authors provide a conceptual definition focused on scalability and dynamic infrastructure changes depending on workload demands. Another, more recent, proposal of an SDI architecture can be found at [10], highlighting the need to scale over heterogeneous groups of resources and emphasizing the role of the SDI manager.

Intel has also developed Rack Scale [11], one of the first SDI frameworks that are closing the gap between academic research and real data centers. Rack Scale allows the dynamic composition of nodes, fully disaggregating its resources in pools, such as CPU, storage, memory, FPGA, GPU, etcetera. Facebook has engaged with Intel to explore this SDI implementation, developing the Facebook Disaggregated Rack [12]. On the other hand, HP is also conducting its own SDI data center implementation with "The Machine" [13]. An industrial prototype exposing PCIe over the network can be found on [14], additionally, a proposal for enabling resource pooling through device lending is made on [15].

In terms of resource disaggregation, [16] evaluates NVMe. Unlike the work presented in this paper, the authors do not focus on NVMe over fabrics and use a custom software layer to expose the devices. FPGAs disaggregation is explored in [17], evaluating the impact of such disaggregation. In [18], the authors examine the network requirements for

disaggregating resources at rack and data center and levels. Minimum requirements are measured in terms of network bandwidth and latency. Those requirements must be such that a given set of applications does not experiment performance degradation when disaggregating memory or other resources over the fabric. [19] presents some performance challenges and issues while using RDMA over Converged Ethernet (RoCE) due to PFC protocol. [20] presents an alternative to PFC to avoid such issues. In our paper, however, we use InfiniBand transport and protocol, which allows us to bypass any PFC issue. Finally, [21] presents a scheduling proposal for in order to help allocate resources on-demand, trying to maximize the number of resources being used and minimizing node latency. To the best of authors' knowledge, there is not literature about strategies for disaggregation on similar terms as this paper.

VI. CONCLUSIONS

This paper has shown 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 datacenters.

The paper has 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 on the scenarios where our modeled workloads are marginally represented. All of these are relevant to help meet workload requirements in datacenters. As observed through simulation, resource composition increases resource sharing up to 2x. The paper shows 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 deadline 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.

ACKNOWLEDGMENTS

This work was partially supported by the Ministry of Economy of Spain under contract TIN2015-65316-P, the Ministry of Science under contract PID2019-107255GB-C21/AEI/10.13039/501100011033, and the Generalitat de Catalunya under contract 2014SGR1051.

REFERENCES

[1] A. Call, J. Polo, D. Carrera, F. Guim, and S. Sen, "Disaggregating non-volatile memory for throughput-oriented genomics workloads," in *Euro-Par 2018 International Workshops, Revised Selected Papers*, 01 2019, pp. 613–625.

[2] N. express, "Nvme over fabrics overview," NVM express, Tech. Rep., 2017. [Online]. Available: http://www.nvmexpress.org/wp-content/uploads/nvme_over_fabrics.pdf

[3] K. Kambatla, V. Yarlagadda, I. Goiri, and A. Grama, "Ubis: Utilization-aware cluster scheduling," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2018, pp. 358–367.

[4] A. Call, "Simulator for disaggregated data-center environments," Dec. 2019. [Online]. Available: https://github.com/MortI2C/simulator

[5] V. Moncunill, S. Gonzalez, S. Beà, L. O. Andrieux, I. Salaverria, C. Royo, L. Martinez, M. Puiggròs, M. Segura-Wang, A. M. Stütz *et al.*, "Comprehensive characterization of complex structural variations in cancer by directly comparing genome sequence reads," *Nature biotechnology*, vol. 32, no. 11, pp. 1106–1112, 2014.

[6] N. Cadenelli, J. Polo, and D. 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)*. IEEE Computer Society, Dec 2017.

[7] T. P. P. Council, "Tpcx-bb," http://www.tpc.org/tpcx-bb/default.asp, 2017.

[8] FIO, "Fio," https://fio.readthedocs.io/en/latest/, 2020.

[9] G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black, "Software defined infrastructures," *IBM Journal of Research and Development*, vol. 58, March 2014.

[10] 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.

[11] Intel, "Intel rack scale design," Intel Corporation, Tech. Rep. 332937-004, aug 2016. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/guides/platform-hardware-design-guide.pdf

[12] I. Facebook, "Facebook disaggregated rack," http://goo.gl/6h2Ut, 2016.

[13] I. Hewlett-Packard, "Hp the machine," http://www.hpl.hp.com/research/systems-research/themachine, 2016.

[14] E. Consoritum, "Expether," http://www.expether.org/, 2019.

[15] J. Markussen, L. B. Kristiansen, R. J. Borgli, H. K. Stensland, F. Seifert, M. Riegler, C. Griwodz, and P. Halvorsen, "Flexible device compositions and dynamic resource sharing in pcie interconnected clusters using device lending," *Cluster Computing*, Sep 2019. [Online]. Available: https://doi.org/10.1007/s10586-019-02988-0

[16] A. Klimovic, H. Litz, and C. Kozyrakis, "Reflex: Remote flash & local flash," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17, 2017.

[17] 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*, Dec 2016.

[18] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. Berkely, CA, USA: USENIX Association, Nov 2016.

[19] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 523536. [Online]. Available: https://doi.org/10.1145/2785956.2787484

[20] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for rdma," 2018.

[21] 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)*, Dec 2016.