Memory Demands in Disaggregated HPC: How Accurate Do We Need to Be?

Felippe Vieira Zacarias
Universitat Politècnica de Catalunya
Barcelona Supercomputing Center
Barcelona, Spain
fvieira@bsc.es

Paul Carpenter
Barcelona Supercomputing Center
Barcelona, Spain
paul.carpenter@bsc.es

Vinicius Petrucci
University of Pittsburgh
Pittsburgh, United States
vpetrucci@pitt.edu

***Abstract*—Disaggregated memory has recently been proposed as a way to allow flexible and fine-grained allocation of memory capacity, mitigating the mismatch between fixed per-node resource provisioning and the needs of the submitted jobs. By allowing the sharing of memory capacity among cluster nodes, overall HPC system throughput can be improved, due to the reduction of stranded and underutilized resources. A key parameter that is generally expected to be provided by the user at submission time is the job's memory capacity demand. It is unrealistic to expect this number to be precise.**

This paper makes an important step towards understanding the effect of overestimating the job memory requirements. We analyse the implications on overall system throughput and job response time. We leverage a disaggregated simulation infrastructure implemented on the popular Slurm resource manager. Our results show that even when the cost of a 60% increase in memory demands only increases a single job's user response time by 8%, the aggregate result of everybody doing so can be a 25% reduction in throughput and a 5 times increase in response time. These results show that GB–hours should be explicitly allocated in addition to core–hours.

***Index Terms*—Disaggregation, Throughput, Response time, Resource scheduling, Resource provisioning, Slurm**

I. Introduction

Disaggregated memory has recently been proposed as a way to provide flexible and fine-grained allocation of memory capacity [1], [2], [3], [4], [5], [6]. The addition of disaggregated memory to an High Performance Computing (HPC) cluster architecture would allow applications to share system memory capacity, reducing or eliminating stranded memory resources that would otherwise be unavailable to other HPC jobs, while maintaining the cost-effectiveness and scalability of traditional HPC architectures.

Although there is some ongoing work on dynamic resource assignment and malleability [7], [8], most HPC job schedulers statically assign resources to jobs. This means that the user of a disaggregated memory system is required to provide an accurate upper bound on the job's memory demands at submission time.

This work investigates how critical such memory demand bounds are for maximising system throughput and minimising job response time (waiting time in the queue plus execution time). We analyse to what degree the users would have a natural incentive to provide accurate memory bounds. Our analysis uses Barcelona Supercomputing Center's (BSC) scalable Slurm simulator for disaggregated memory systems (see Section II-C).

978-1-5090-4023-0/17/$31.00 ©2017 IEEE

We use a simulation approach for two reasons: (1) there are no large-scale HPC systems with disaggregated memory hardware including a complete software stack, and (2) simulations allow studies to be performed more quickly without occupying resources of large-scale production systems.

In our studies, we show that, from the HPC system operators perspective, overall system throughput is conditional on accurate memory estimations, but, from the perspective of a single user on a large system, there is little incentive to provide an accurate bound on memory consumption. Even when the cost of a 60% increase in memory demands only increases a single job's user response time by 8%, the aggregate result of everybody doing so can be a 25% reduction in system throughput and a 5 times increase in average response time. We make some initial recommendations and encourage additional research in this direction. If these results are reproduced more widely, then it will almost certainly be necessary to allocate GB–hour memory capacity explicitly in addition to core–hours.

In summary, we make the following contributions:

1) We use BSC's scalable Slurm simulator for disaggregated memory to investigate how the user-specified memory upper bound affects overall system throughput
2) We introduce a simulation-based methodology to correlate the accuracy of the memory upper bound with the job's response time
3) Assuming these results can be reproduced more widely, we make recommendations that can be applicable to production systems.

The rest of the paper is organized as follows. Section II provides a brief background of our work and Section III explains the methodology to support our experiments. Section IV provides the results and discussions. Section V distinguishes our approach from the large body of related work. Finally, Section VI concludes the paper.

II. Background

A. Disaggregated memory

Although NUMAcc machines are available with hundreds of sockets, for example SGI Altix [9] and Bull Coherent Switch (BCS) [10], it is difficult to scale cache-coherent systems to thousands of nodes. Modern HPC systems are therefore typically built from thousands of NUMAcc nodes communicating via a fast interconnect such as InfiniBand or OmniPath.

Disaggregated architectures interconnect individual components such as processor, memory and storage over a network [11], [12], [13]. The EUROSERVER [12], ExaNoDe [14] and EuroEXA [15] projects have pioneered a disaggregated system architecture, which provides a global physical address space known as UNIMEM [12], with the ability for cores to access remote memory. A similar approach is taken by ThymesisFlow [16]. Computing units execute their own Operating System, and can access memory directly attached to it (local memory access) as well as memory attached to another computing unit through a global interconnect (remote memory access). The remote memory access is performed through a common Global Address Space, either using ordinary load–store instructions or via Remote Direct Memory Access (RDMA).

B. Slurm Resource manager

Slurm is a popular open-source resource and job management system used for HPC [17]. It features a plug-in module architecture that is highly configurable for a variety of extensions including workload, queueing, scheduling, and so on. Its baseline node allocation uses an exclusive mode and the selection of resources that satisfy the minimum resource request, nevertheless if not all resources within the node are utilized by a specific job, no other job is allowed to share the resource.

We use the BSC Slurm simulator [18], [19], which enables a precise and deterministic evaluation of the job scheduler, since it is built upon and uses most of the original Slurm source code. It is able to capture all parameters and behavior that occurs in a real environment, exceeding that of theoretical models.

C. Slurm simulator supporting disaggregated memory

We use the BSC simulator extensions for disaggregated memory [20], [21], which adds support for disaggregated memory, modelling the slowdown due to remote memory accesses, and extends Slurm's allocation policy to exploit disaggregated memory. The performance model is a slowdown-based method [22], extended to support MPI processes [20]. The allocation policy first selects nodes that have enough local memory to satisfy the job's requirement of memory per node to avoid unnecessary remote memory access. To improve performance when it uses dissaggregation, the approach does not use the CPU cores of nodes that have already lent memory to another node. This means that such a node effectively becomes a memory node for other jobs. The approach allocates jobs to the nodes with higher available memory, applying a weight to each node based on the free memory. This tends to reduce the slowdown due to remote memory accesses.

TABLE I
SLURM CONFIGURATIONS USED IN OUR SIMULATIONS.

Configuration parameter	Value(s)
System size	1024 nodes
Number cores per node	32
Memory per node	32 GB, 64 GB
Allocation policy	Disaggregated
Scheduling policy	Backfill
Queue and Backfill size	100
Backfill and Scheduling interval	30 s
Heterogeneous system ratio: % Large nodes	0, 15, 25

III. Methodology

A. Simulated system and input configurations

The configuration of the simulated system is given in Table I. We use multiple scenarios, each with 1024 nodes, separated into *normal* nodes, which have the typical memory capacity, and *large* nodes, which have twice the memory capacity of the *normal* nodes. The scenarios correspond to different ratios between large and normal nodes, varying from 0% (all normal nodes) to 100% (all large nodes). The simulations were conducted on a cluster of two-socket 8-core Intel Xeon SandyBridge-EP E5-2670 servers with 20 MB L3 cache (LLC) per socket, shared among all cores, and 64 GB of DDR3-1600 DIMMs.

The input job trace files were generated using the CIRNE Comprehensive Model [23] and augmented to target each of the specific heterogeneous system ratios, listed in Table I. We ensured that all traces have total *node–hours* (#nodes $\times$ runtime) of large and small jobs in the indicated ratio. The characteristics of the large and small jobs are given in Table II. The memory demand of normal jobs is less than the capacity of a normal node, whereas all large jobs demand more memory than a normal node capacity.

TABLE II
LARGE AND SMALL JOB CHARACTERISTICS

	Normal Jobs		Large Jobs	
Metric	**Memory (GB)**	**Node–hours**	**Memory (GB)**	**Node–hours**
Min	0.12	0.0	33.0	0.0
1st Qu.	1.7	0.85	48.2	0.0
Avg	6.2	52.6	48.5	24.9
3rd Qu.	3.8	15.0	49.8	2.1
Max	27.6	6412	49.8	3659.0

B. Extending the simulator with memory overestimation

We first extended the Slurm simulator with a memory overestimation module that represents the user, by determining the memory consumption bound at submission time, as a function of the actual peak memory consumption and the intended user behavior. The output of this module is the estimated upper bound, which the Slurm simulator passes to the disaggregated-aware Slurm in order to allocate resources.

The actual memory consumption, however, is still used by the slowdown model to determine the effect on performance.

In the baseline experiments, the upper bound equals the actual memory consumption. This is the unrealistic best case, in which the users perfectly estimate each job's memory consumption. Otherwise, the estimated memory bound can be either (1) overestimated by a fixed percentage, which can be used to quantify the general effect of overestimation on overall system throughput and response times, or (2) overestimated by an independent, identically distributed (i.i.d.) uniformly-random percentage between 0% and 100%, which is used by the correlation analysis to quantify the effect on single job response time.

C. Determining the effect on system throughput

To determine the effect of overestimation on system throughput, we configure the memory overestimation module to uniformly over-estimate the memory demands of all jobs, between $+0\%$, the baseline, and $+100\%$, which doubles the memory demands of all jobs. We then plot system throughput as a function of the overestimation.

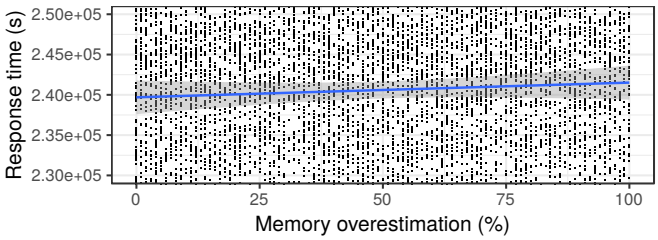
D. Correlating memory overestimation and response time

To determine the effect of overestimation on individual job response time, we may plot a typical job's response time as a function of its memory overestimation, holding everything else constant. But it is clearly not practical, in terms of simulation time, to do this one job at a time.

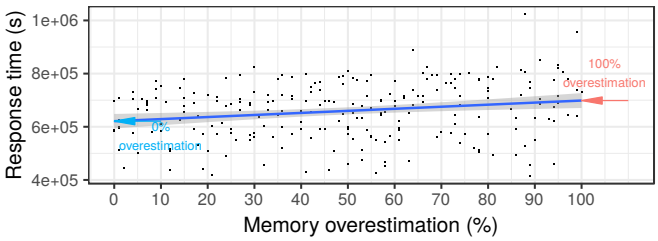
Instead, we perform a correlation analysis, by running the original trace several times, applying a uniformly-random overestimation to each job. For this correlation analysis to make sense, it is important that the degree of memory overestimation is independent of other job characteristics. This is one reason why the simulation approach is important, as it allows us to apply an i.i.d. random overestimation. Observational data may be misleading, for instance, if larger jobs systematically had a larger (or smaller) degree of overestimation.

Fig. 1a shows a direct plot of the response time as a function of the memory overestimation, across all jobs, for the scenario with 50% large jobs and 0% large nodes. The full results, for all scenarios, are in Section IV. We add a trend line using linear regression. Since the jobs have widely varying response times, even with no memory overestimation, the points on the y-axis have a very large range, of which Fig. 1a shows a small part. We therefore filter the jobs by the baseline response time, when the overestimation is $+0\%$, to obtain Fig. 1b, which is for jobs whose baseline response time was between 3×10^5 s and 4.2×10^5 s, which was the maximum baseline response time. Similar plots were obtained for each interval of baseline response times. There is still significant noise, but it is much less than before. Fig. 1b also shows the trend line, which allows us to predict the average response, for jobs with the given range of baseline response times, i.e. from $+0\%$ to $+100\%$, as a function of the memory overestimation.

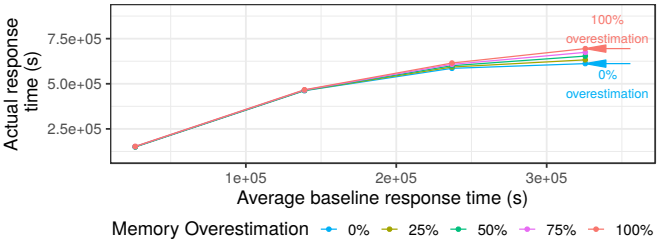
Finally, Fig. 1c assembles all the information into a single figure. The x-axis is the baseline response time, for $+0\%$ overestimation, and the y-axis is the actual response time, depending on the overestimation (five different curves). In this example, we see a large increase in the overall response times, e.g. from 2.0×10^5 s to 7.5×10^5 s for the top-rightmost point, but we see very little difference between the $+0\%$ and $+100\%$ cases. In the next section, we will show the complete results, which follow a similar trend.



(a) Response time for all jobs



(b) Filtering response time using a fixed interval range



(c) Using trend line to derive the response time

Fig. 1. Correlating memory overestimation to response time (example with 50% large jobs and 0% large nodes)

IV. Results

A. System job throughput

Fig. 2 presents the throughput of the system, in jobs per second, as a function of the memory demand overestimation, across all scenarios. We notice that, in all cases, system throughput drops as the overestimation increases. Even though for low degrees of overestimation, the impact on throughput is modest, the degradation increases with the mismatch between system and the job mix, reaching almost 40%.

We therefore conclude that, from the system operator's perspective, effective system utilization requires that the jobs generally have accurate estimations of their memory demands.

B. System job response time

Fig. 3 shows the average response time when all jobs overestimate the memory demands by the same amount. Following a similar trend as the decrease in throughput, the response

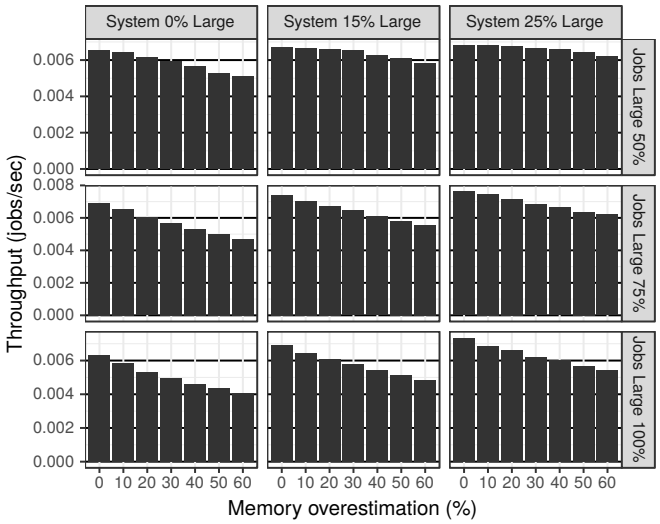


Fig. 2. System throughput (*y-axis*) when all jobs overestimate memory requirements by the same percentage (*x-axis*).

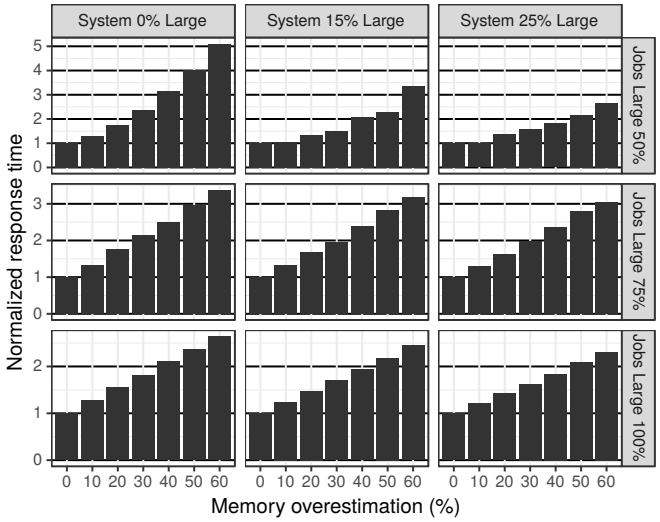


Fig. 3. Normalized response time (*y-axis*) when all jobs overestimate memory requirement by the same amount (*x-axis*).

time increases, showing that the system's response time is impacted as a whole when all jobs overestimate their requests. For such a scenario, we notice that when all users are accurate in specifying the memory usage, it would benefit the whole system, because it would decrease the load on the system in queuing time due to the lack of resources. Consequently, the system would run more efficiently by decreasing overall response time and increasing overall throughput.

C. User job response time

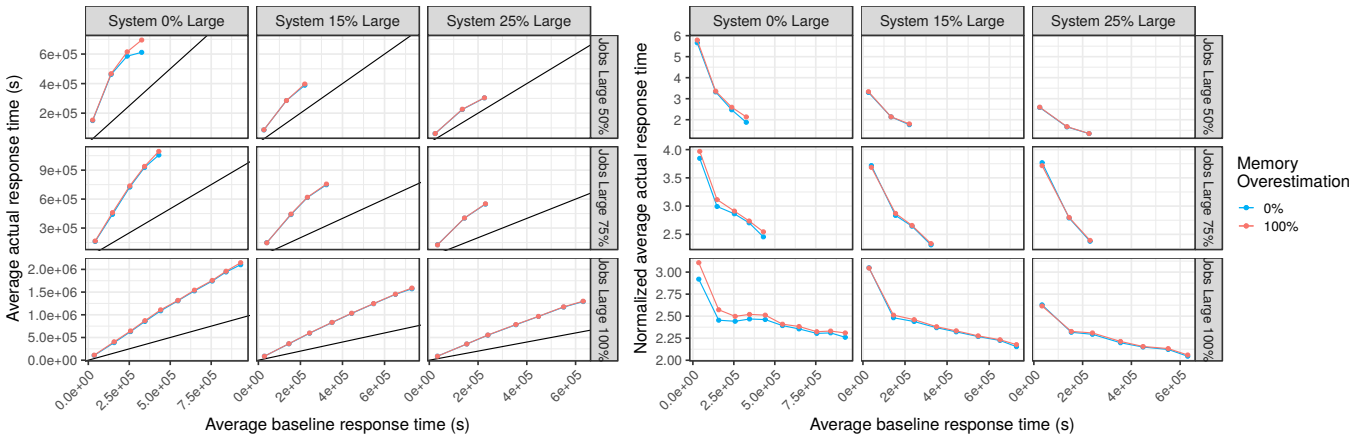
Each plot in Fig. 4 shows the average response time (y-axis) as a function of the baseline response time (x-axis) for the jobs of two types of user. The data is obtained following the methodology described in Section III. The 0% line is for a "diligent" user who accurately determines the memory consumption whereas the 100% line is for a "careless" user whose prediction is twice the actual consumption. Results are shown in a 3×3 grid, corresponding to the three different systems and job mixes. Fig. 4(a) on the left plots the actual average response time, whereas Fig. 4(b) on the right plots the normalized response time.

We see in each scenario a large increase in response time, compared with the baseline (represented by the black line). We notice, as expected, that in all scenarios the response time is impacted even for the diligent user, whose jobs do not overestimate the memory demands. For *0%* large node system, where disaggregation is more often used to accommodate the job mixes, we perceive a slight increase on response time when the job doubles its request. However there is little or no difference in the response time when we start adding large nodes to the system. We observe that being accurate on a scenario where other users are not, provides a small benefit to a single user, whose jobs will be impacted by the load the other users create on the system due their overestimation.

V. Related Work

Memory Disaggregation — Gu *et al.* [2] implement a scalable and decentralized remote memory paging solution to enable memory disaggregation. It divides the swap space of each machine and distributes the pages across many remote machines using RDMA operations for all remote I/O operations. Lim *et al.* [6], [13] propose a remote memory blade that can be used for memory capacity expansion to improve performance and for sharing memory across servers. The authors extended the Xen hypervisor to emulate a disaggregated memory design, in which remote pages are swapped into local memory. Shan *et al.* [24] propose a split kernel OS architecture to manage disaggregated systems. It breaks the OS into pieces with different functionalities, each running on and managing a hardware component. Peng *et al.* [5] implement a user-space remote paging library to allow exploration of applications using disaggregated memory. Their architecture contemplates nodes with fast but small local memories and large but slow remote memories, and it is aided by the library, which evicts local pages and fetches remote pages when the local memory is exhausted. Pinto *et al.* [16] present *ThymesisFlow*, a fully-functional software-defined disaggregated memory prototype using commercially available hardware components. The architecture introduces the concepts of a compute role, which uses remote memory, and a memory-stealing role, which donates memory. Its design leverages the latest cache-coherent attachment technology for off-chip peripherals to intercept memory transactions and realize the endpoint functionality. They analyze the impact of disaggregated memory by measuring the performance of cloud applications and demonstrate an acceptable performance.

Resource allocation — Amaral *et al.* [25] develop a dynamic loop-based controller to manage resources and a flow-network algorithm to determine the optimal placement of workloads on virtualized data-centers. Their approach disaggregates GPUs using middleware that intercepts GPU calls and offloads API-related data via the network. Amaro *et al.* [26] present a swapping mechanism that uses remote memory through RDMA and a remote memory-aware cluster scheduler to split



(a) (b)

Fig. 4. Individual job response times increase when the users overestimate job memory demands, but memory overestimation has little effect on response times ($+0\%$ curves vs $+100\%$ curves)

each job's memory demand between local and remote memory. Then, they examine the scenarios for which remote memory can increase job throughput. Papaioannou *et al.* [27] propose a resource scheduling and network management algorithm designed for a disaggregated data center. The scheduler allows the administrator to define policies, which are enforced through a set of weights. The proposed scheduler is evaluated using simulation improving the resource utilization compared to state-of-the-art algorithms and thus reducing the energy consumption. Zervas *et al.* [4] propose a set of algorithms to allocate and maximize resource utilization for a disaggregated data center based on the dRedBox architecture. They developed a simulator that performs orchestration and allocation of resources with reservation of their network bandwidth and interconnection to serve VM requests. Farias *et al.* [28] use traces of a representative production system to simulate a scheduler for disaggregated architectures. They investigate the efficiency gains when the scheduler can either create new logical servers, or increasing the capacity of those existent.

Accessing/pricing schemes — Access to large-scale HPC infrastructures usually requires submission of proposals to undergo a peer-review process describing computational resources as in [29], [30], [31]. Mahloo *et al.* [32] compare the cost in terms of capital and operational expenditures of a disaggregated architecture and one based on traditional servers. Their framework results show that disaggregation brings high savings in the presence of heterogeneous workloads. Borghesi *et al.* [33] present a model to analyze the impact of frequency scaling on energy. To assess the cost benefits for the facility and user, they propose four different pricing schemes and conclude that is possible to save energy while not penalizing users from the economic point of view. Ferretti *et al.* [34] introduce a model to help researches to understand whether is convenient using Cloud infrastructures as alternative to HPC systems for running scientific applications. Their model takes into account performance, cost, waiting time and user's preference. They concluded that the best infrastructure may be that which optimizes the user's expectations. Breslow *et al.* [35] present a runtime system to enable fair pricing for HPC clusters that run co-located applications and a new pricing model to fairly price applications when co-locations are present. The pricing model provides to the user discounts at a rate proportional to the degradation that each of their jobs experience due contention.

VI. Conclusion

Disaggregated memory is under development as a way to provide flexible fine-grained allocation of physical memory. Users of an HPC system supporting disaggregated memory would likely be expected to estimate their job's memory demands. This paper investigates how the system's overall throughput and response time would be affected, according to various assumptions on the user's ability to predict the memory consumption. We find that even when there is a large effect on system throughput (-25%) and response time (5 times higher), there is very little direct incentive for the users to be accurate in their estimates, with only an 8% increase in response time. This paper is a step towards understanding how to bring disaggregated memory to HPC, by demonstrating that users should receive incentives to provide accurate memory usage estimates. These incentives could translate to an increase in priority, number of simultaneous running jobs or larger core–hour allocations.

Acknowledgement

This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 754337 (EuroEXA); it has been supported by the Spanish Ministry of Science and Innovation (project TIN2015-65316-P and Ramon y Cajal fellowship RYC2018-025628-I), Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272), and the Severo Ochoa Programme (SEV-2015-0493).

REFERENCES

[1] D. Syrivelis, A. Reale, K. Katrinis, I. Syrigos, M. Bielski, D. Theodoropoulos, D. N. Pnevmatikatos, and G. Zervas, "A software-defined architecture and prototype for disaggregated memory rack scale systems," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2017, pp. 300–307.

[2] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 649–667.

[3] V. R. Kommareddy, A. Awad, C. Hughes, and S. D. Hammond, "Exploring allocation policies in disaggregated non-volatile memories," in *Proceedings of the Workshop on Memory Centric High Performance Computing*. ACM, 2018, pp. 58–66.

[4] G. Zervas, H. Yuan, A. Saljoghei, Q. Chen, and V. Mishra, "Optically disaggregated data centers with minimal remote memory latency: technologies, architectures, and resource allocation," *Journal of Optical Communications and Networking*, vol. 10, no. 2, pp. A270–A285, 2018.

[5] I. Peng, R. Pearce, and M. Gokhale, "On the memory underutilization: Exploring disaggregated memory on hpc systems," in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2020, pp. 183–190.

[6] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 267–278.

[7] M. D'Amico, A. Jokanovic, and J. Corbalan, "Holistic slowdown driven scheduling and resource management for malleable jobs," in *Proceedings of the 48th International Conference on Parallel Processing*. ACM, 2019, p. 31.

[8] S. Iserte, R. Mayo, E. S. Quintana-Ortí, V. Beltran, and A. J. Peña, "Efficient scalable computing through flexible applications and adaptive workloads," in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 2017, pp. 180–189.

[9] "SGI Altix UV 1000 Datasheet," https://cutt.ly/McVBAFS, 2021, accessed: 2021-01-21.

[10] "Bull Coherent Switch (BCS)," https://cutt.ly/IcVVZt3, 2021, accessed: 2021-01-21.

[11] M. Bielski, I. Syrigos, K. Katrinis, D. Syrivelis, A. Reale, D. Theodoropoulos, N. Alachiotis, D. Pnevmatikatos, E. Pap, G. Zervas *et al.*, "dReDBox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1093–1098.

[12] Y. Durand, P. M. Carpenter, S. Adami, A. Bilas, D. Dutoit, A. Farcy, G. Gaydadjiev, J. Goodacre, M. Katevenis, M. Marazakis *et al.*, "Euroserver: Energy efficient node for european micro-servers," in *2014 17th Euromicro Conference on Digital System Design*. IEEE, 2014, pp. 206–213.

[13] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, 2012, pp. 1–12.

[14] A. Rigo, C. Pinto, K. Pouget, D. Raho, D. Dutoit, P.-Y. Martinez, C. Doran, L. Benini, I. Mavroidis, M. Marazakis *et al.*, "Paving the way towards a highly energy-efficient and highly integrated compute node for the exascale revolution: the exanode approach," in *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE, 2017, pp. 486–493.

[15] EuroEXA project, "H2020 project number 754337," 2009, accessed: 2021-09-20. [Online]. Available: https://euroexa.eu/

[16] C. Pinto, D. Syrivelis, M. Gazzetti, P. Koutsovasilis, A. Reale, K. Katrinis, and H. P. Hofstee, "Thymesisflow: a software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 868–880.

[17] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *JSSPP*. Springer, 2003, pp. 44–60.

[18] "BSC slurm simulator," https://github.com/BSC-RM/slurm_simulator, 2021, accessed: 2021-01-20.

[19] A. Jokanovic, M. D'Amico, and J. Corbalan, "Evaluating slurm simulator with real-machine slurm and vice versa," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2018, pp. 72–82.

[20] F. V. Zacarias, P. Carpenter, and V. Petrucci, "Improving hpc system throughput and response time using memory disaggregation," in *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2021.

[21] "Disaggregated memory slurm simulator and allocation policy." https://github.com/felippezacarias/slurm_simulator, 2021, accessed: 2021-04-08.

[22] F. V. Zacarias, R. Nishtala, and P. Carpenter, "Contention-aware application performance prediction for disaggregated memory systems," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020, pp. 49–59.

[23] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 140–148.

[24] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed OS for hardware resource disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 69–87.

[25] M. Amaral, J. Polo, D. Carrera, N. Gonzalez, C.-C. Yang, A. Morari, B. D'Amora, A. Youssef, and M. Steinder, "Drmaestro: orchestrating disaggregated resources on virtualized data-centers," *Journal of Cloud Computing*, vol. 10, no. 1, pp. 1–20, 2021.

[26] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker, "Can far memory improve job throughput?" in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[27] A. D. Papaioannou, R. Nejabati, and D. Simeonidou, "The benefits of a disaggregated data centre: A resource allocation approach," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–7.

[28] G. Farias, F. Brasileiro, R. Lopes, M. Carvalho, F. Morais, and D. Turull, "On the efficiency gains of using disaggregated hardware to build warehouse-scale clusters," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 239–246.

[29] "Partnership for Advanced Computing in Europe," https://prace-ri.eu/about/introduction/, 2021, accessed: 2021-01-20.

[30] "Extreme science and engineering discovery environment," www.xsede.org, 2021, accessed: 2021-01-20.

[31] "Innovative and novel computational impact on theory and experiment," https://www.doeleadershipcomputing.org/proposal/call-for-proposals/, 2021, accessed: 2021-01-20.

[32] M. Mahloo, J. M. Soares, and A. Roozbeh, "Techno-economic framework for cloud infrastructure: A cost study of resource disaggregation," in *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2017, pp. 733–742.

[33] A. Borghesi, A. Bartolini, M. Milano, and L. Benini, "Pricing schemes for energy-efficient hpc systems: Design and exploration," *The International Journal of High Performance Computing Applications*, vol. 33, no. 4, pp. 716–734, 2019.

[34] M. Ferretti and L. Santangelo, "Cloud vs on-premise hpc: a model for comprehensive cost assessment," *Parallel Computing: Technology Trends*, vol. 36, p. 69, 2020.

[35] A. D. Breslow, A. Tiwari, M. Schulz, L. Carrington, L. Tang, and J. Mars, "Enabling fair pricing on hpc systems with node sharing," in *Proceedings of the international conference on high performance computing, networking, storage and analysis*, 2013, pp. 1–12.