Gestalt Computing: Hybrid Traditional HPC and Cloud Hardware and Software Support

Jay Lofstead
Sandia National Laboratories
Albuquerque, NM, USA
email: gflofst@sandia.gov

Andrew Younge
Sandia National Laboratories
Albuquerque, NM, USA
email: ajyoung@sandia.gov

***Abstract*—Traditional modeling and simulation-based HPC workloads demand a platform optimized for low latency node-to-node communication and a write intensive IO load. Cloud-based applications tend to not need the fast communication and are frequently heavily read dependent. Both of these workloads are immensely important and demand larger and larger machines to handle the demand. Unfortunately, budgets demand that large scale compute be a shared resource for both workloads in spite of this opposed design priority. Through the use of persistent memory (PMEM) devices on node, dynamically we can reconfigure the nodes to either support fast reading through using the on node-PMEM as a reading cache or as slow DRAM to reduce the time spent communicating with logical neighbor nodes. We outline a vision for a machine hardware and software architecture dynamically shifting to simultaneously support both workloads as demand dictates with minimal additional cost and complexity.**

***Index Terms*—HPC, Cloud, hybrid computing, hybrid workloads, persistent memory, job scheduling**

I. Introduction

Modern large scale computing is no longer just used for uniform modeling and simulation (modsim) workloads, but also incorporates data intensive workloads like machine learning and other data analytics techniques. The tools on both sides have matured considerably and both offer excellent support for their respective workloads. The new challenge is the realization that traditional HPC workload data is being processed by these newer data analytic techniques. The arriving challenge is that the generated modsim data needs to be processed as it is generated eliminating the need to store raw data.

Traditional HPC platforms are organized to best support *large-scale, scale-up* workloads. These are a single task that work on a single or a small number of very large data items iteratively to explore physics phenomena in some way. These very large data items can be 1 PB or larger demanding spreading the computation across many nodes not just for processing speed, but also simply to be able to hold the data in working memory.

These scale-up platforms use high performance, low latency interconnect networks, such as InfiniBand [1] and Slingshot [2], to reduce the communication overheads of the frequent data exchange operations. Each large data item represents something, such as a volume of air in a room. To look at air flow patterns, the individual air molecules, or some representative value, must be tracked on a short distance periodic basis in the entire volume. The aggregate size can be extremely large if extended to something like the global atmosphere. To evolve the computation, some force in the form of air molecules with directional velocity are inserted pushing the existing molecules forming air currents. To perform the calculations, the data is split into sub-volumes, each assigned to a compute unit of some sort, and calculated independently. Since the force is not confined to any given sub-volume, after each calculation round, the edges are exchanged with the logical neighbors to enable more independent computation. Periodically, all of the data is written to persistent storage for offline analysis of the simulation evolution. One challenge being faced today and continuing to worsen is that the storage IO bandwidth is not growing fast enough to absorb data at the rate scientists are willing to pay for. In essence, unless the scientists want to spend the vast majority of their compute time allocation performing IO to storage, they need to write far less often than they would prefer. Instead, they want to perform their data analysis and processing tasks, which can reduce data volumes dramatically, as part of their computation process.

The new generation HPC workloads run on platforms organized to best support *large-scale, scale-out* workloads. These are many tasks that are used to process a very large number of relatively tiny data items and use parallelism to accelerate the computation. Additional compute nodes are used to increase compute speed, but each computation is largely independent, allowing seemingly unlimited scaling with linear speed increases simply by adding more compute capacity.

This new generation, as it encompasses such a large number of new markets and types of data processing, has prompted developing rich, accessible tools that with a little work, can also process the scale-up data. The quality of these tools has prompted scientists to demand support for incorporating them directly into their workflows with no regard to the underlying hardware and software system support. System administrators are left trying to best support user demands, but with the wrong tools.

The underlying software infrastructure required for these scale-out tools generally fits better in an orchestration system. Current examples such as Kubernetes [3], OpenStack [4], and Docker Swarm [5] are optimized to dynamically start and stop a number of service instances on an as demanded basis.

 ISBN: 978-1-61208-948-5

Resource sharing performance penalties are a lower concern since processing is largely independent.

Further motivating this situation are budgetary concerns from funding agencies. High-end machines are very expensive to field and operate and the thought of fielding two machines of similar scale, but a bit different design just to support two different workloads is not seen as a responsible use of budget. Instead, the funding agencies have been prompting the laboratories to find ways to make these workloads co-exist on a single, slightly larger platform. This will be slightly more expensive than one to both field and operate and still be able to handle the aggregate computational needs.

While cloud bursting [6] and multi-cloud deployments may be able to address the needs in theory, other policy or security concerns may demand a fully secured, on site deployment. Export controlled, sensitive, or classified data or processing have special requirements of the cloud platform and the connection with it. Certification [7] can allow workloads and data for some while higher consequence and more sensitive information still cannot use these infrastructures. This prompts a need for hybrid on-site resources that can easily handle both the traditional modsim workloads as well as the data analytics workloads.

The cost of provisioning a system to fully support both workloads is prohibitive. Excess DRAM capacity would go unused for many workloads while node-local high capacity, low cost storage (e.g., disk or equivalent) introduces even more costs and failure points. This excess is financially impossible in essentially all cases. Instead, there needs to be a technology that can serve both to expand DRAM capacity as well as serve as node-local storage. Persistent Memory (PMEM) offers such a solution.

PMEM devices live on the memory bus and are accessed via load/store instructions. At the same time, compared to DRAM prices, they are cheaper for capacity. This makes them ideal for expanding DRAM capacity at a reasonable cost. PMEM devices, because they offer the additional persistence property, also offer a way to store larger data quantities on node with less concern about failures. This dual property makes PMEM an intriguing solution to the hybrid device need. Unfortunately, hardware alone is not sufficient to realize the gestalt platform ideal.

This paper explores a potential systems and software architecture leveraging persistent memory devices, such as the Intel Optane [8], that could offer higher memory capacity for heavily compute intensive workloads and near-compute storage for data throughput-intensive workloads.

The rest of this paper is structured as follows. First, in Section II, we discuss persistent memory devices, their characteristics, and how we propose they can be used. Section III, we describe the software environment required to support these hybrid workloads. Two use cases are presented demonstrating the potential of this design in Section IV. Next in Section V, we discuss some related work. Finally, Section VI offers takeaway requirements and challenges we need to address as a community to achieve a true gestalt platform capable of easily supporting any mix of traditional HPC mod-sim and data analytics workloads simultaneously.

II. Persistent Memory

Persistent memory (PMEM) devices offer higher capacity at lower performance and cost than traditional DRAM devices. One commercially available option is the Intel Optane [8]. Unfortunately, confusing naming hides these devices among different technology suppressing marketplace visibility. In spite of this, considerable work [9]–[16] has been done exploring the potential for use as both extended memory (volatile-style) and storage (persistent-style) devices. This hybrid nature offers a dual use technology that can be the key enabling hardware for a gestalt workload.

Typically, to extend memory into a persistent storage media requires going through a special interface, such as SATA [17] or the PCIe [18] bus based NVMe [19] devices. Fast, solid state devices can be deployed through these interfaces, but interacting with them is fundamentally different from accessing regular memory. Instead of a typical load/store instruction, working through a device driver of some sort is required. The access granularity is typically a block, which may range from 4 KB to even 1 MB. Reading a single byte from these devices causes accessing an entire block at a time. The block is then managed in the device driver's memory and the single byte is returned to the caller.

While systems like `mmap` [20] can hide a block-storage device behind a memory-like standard API, all it does is to translate the memory accesses into block-storage accesses with a memory page granularity. Systems like DAX [21] can more directly map devices for access, but it is specialized hardware for accessing slower devices than PMEM.

Prior to PMEM devices, NVMe devices held the fastest storage devices title. While NVMe is relatively expensive compared to spinning media (i.e., disks), the performance is dramatically different making it a difficult choice to deploy disks except for large capacity, slow access devices. NVMe devices are not without their limitations. Ceph [22] rewrote their low-level device access code because their disk and SSD (over SATA) optimized code was too slow to enable streaming to NVMe devices at hardware rates. Achieving the full performance potential is possible, but takes some work.

Compare that configuration against how PMEM devices are deployed. PMEM devices are placed on the memory bus and are accessed via load/store instructions within the CPU. The access granularity is a cache line. Compatible CPUs have been modified to tolerate the longer latencies involved with using PMEM devices to avoid timeouts or other failure conditions. As far as the machine code is concerned, these are normal main memory devices with no special software drivers or other interfaces necessary. The performance characteristics are 3x-5x faster than an NVMe device.

Alternatively, technologies like CXL [23] offer a way to place these devices across the interconnect for shared use. Through a CXL accessible pool, a large scale, shared, persistent store can be configured for the machine simplifying and

 ISBN: 978-1-61208-948-5

accelerating data sharing among nodes. A recent announcement that Pacific Northwest National Laboratory and Micron are partnering to investigate this setup to accelerate HPC with ML/AI workloads [24].

Standard IO routines are tuned for standard IO interfaces making NVMe devices easy to deploy. A little software tuning and significant performance advantages can be realized. PMEM devices require specialized interface code to achieve the full potential. The pMEMCPY [16] library demonstrates that without special tuning, even a highly optimized IO library can only achieve about half the available performance of a PMEM device. The extra copies in the OS kernel must be avoided to get the full performance benefits.

A. Machine Architecture

Our proposed hardware architecture is illustrated in Figure 1. It consists of a CPU and GPU both with high bandwidth memory and additional PMEM devices capable of holding as much as 3x the CPU and/or GPU memory. This 3x capacity has been used for several DOE Leadership Computing capacity calculations for burst buffers (e.g., Summit at ORNL). Adopting this standard makes reasonable sense based on this standard. This enables compute to swap to the PMEM devices for significantly longer independent computation phases, the ability to write data for slower migration off node, and to hold a significant corpus for data analytics tasks to avoid going off node to load data.

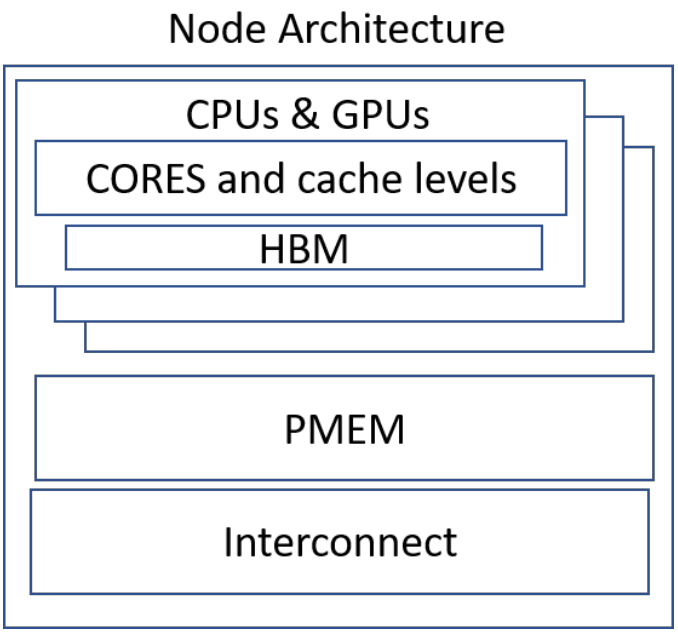

Fig. 1. Proposed Node-level architecture

The interconnect would need to be a traditional HPC-style interconnect optimized for low latency, high bandwidth operation, such as InfiniBand. Storage would be in a large, shared pool that would serve as scale-up scratch for offloading from the compute nodes or for staging into the compute area prior to running an analytics job.

Otherwise, the machine architecture could be relatively "standard". For example, the scratch/shared storage pool could be a large scale distributed or parallel storage system with various tiers from scratch to campaign storage to archive. This part of the architecture is typically very similar for cloud or HPC deployments. The main differentiator would likely be in a cloud deployment, there is more redundancy with assumed failures while HPC would focus on more reliability and ability to recover from failures. With standard components used across both types of machines, the costs are essentially the same with the software layer differentiating the operational aspects.

B. Discussion

PMEM devices can fit into the niche needed to take largely similar platforms and enable them to dynamically adapt to whatever workload is deployed on them. With an adequate software layer on top, this gestalt can be achieved.

Complicating this picture are the pricing differences. PMEM can achieve around 3x the performance of NVMe devices, but cost 5x as much. Unless that performance gain is essential and tuning the IO libraries to fully take advantage of PMEM is possible, it is unlikely worth the extra cost except in specialized cases. For the purposes of this paper, we are considering two cases. The first is where the cost is less important than the raw achievable performance for time sensitive applications. The second is the potential for eliminating an entire platform's cost by deploying this more expensive technology into an existing platform and adapting the software infrastructure to form the gestalt.

III. Software

Software infrastructure has a long way to go. The cloud native structure of using a minimal virtual machine to host containers for the functionality is the right start. For this system design, we could configure for deployment a minimum of two Virtual Machines (VMs) or a spectrum with different ratios of memory-configured or storage-configured PMEM devices. HPC system administrators are starting to understand the benefit of this approach, but are still concerned about performance loss. Given the financial realities, accepting a tiny performance loss to support diverse workloads may be the price paid.

While the cloud-native software infrastructure may seem to be adequate, it does not support bulk-synchronous parallel workloads well. Cloud-like infrastructure focuses on spinning up or down independent service instances, potentially all of the same type. Scale-up HPC best supports a single large instance spread across potentially all of the nodes. Startup times should be very similar to avoid wasted compute times as no process can really get started until all of them have started. The same is true for the synchronization points. All task processes communicate to exchange data after each computation phase. While alternative programming models, such as Charm++ [25] and Legion [26] seek to break this dependency, they instead rely on oversubscription to cores and delayed computation

 ISBN: 978-1-61208-948-5

to handle communication overheads and delays. That way computation is never waiting for data to arrive. Net compute time may reduce, but wall clock time may increase, but on fewer resources. Further, re-architecting codes into this model is considerable, non-trivial effort. Finally, the compatibility of this scale-up model with cloud software infrastructures is unproven.

Using containers for software deployment makes sense for many reasons. First, immutable, stored containers offer a precise reference to what code was run for future publications and reproducibility. Second, containers eliminate missing or conflicting third party library problems. As long as the container itself can work, then it can work in any compatible container hosting environment. Third, Continuous Integration/Continuous Deployment infrastructure is often designed to work with containers to perform the regular automated testing tasks. We propose using containers for all of the software elements given the natural deployment capabilities developed in the cloud space in addition to others. A simple picture of this software architecture is presented in Figure 2.

Ideally, all data would be in containers as well. A prototype of this model called Data Pallets [27] later updated in Olaya's Master's thesis [28], shows the potential for enhancing not just reproduciblity, but also traceability and understandability for workflows.

Immutable rather than ephemeral containers are a key element. By preserving, unchanged, the containers, reproducibility is easier to verify. Associating container hash values with the resulting output offers a strong connection to how the data is created.

This reproducibility benefit can be enhanced by storing the virtual machine images used, along with the container hash codes, to offer a stronger reproducibility foundation for research. Long term, hardware emulators would be necessary to replace obsolete/unavailable CPUs or other infrastructure, but that is a solvable problem.

Finally, combining all of these together requires a system such as a fully realized Fuzzball [29], to manage deployment. Better effort on simplifying the system and improving performance is needed. Application startup time should be at most seconds rather than several minutes, as it stands in the prototype version available today. Fuzzball seeks to support both traditional scale-up and scale-out workloads simultaneously with a ground-up re-build of the support infrastructure. This kind of rethinking of the software layer is essential to a hybrid platform and it requires PMEM, or some similar concept, to be able to contain costs while meeting the true gestalt.

IV. Use Cases

The first use case is a traditional modsim workload that couples with data analytics code on separate, or even the same, node(s) to form a single workflow. The second use case is a traditional cloud data analytics workflow. A third is a traditional modsim alone.

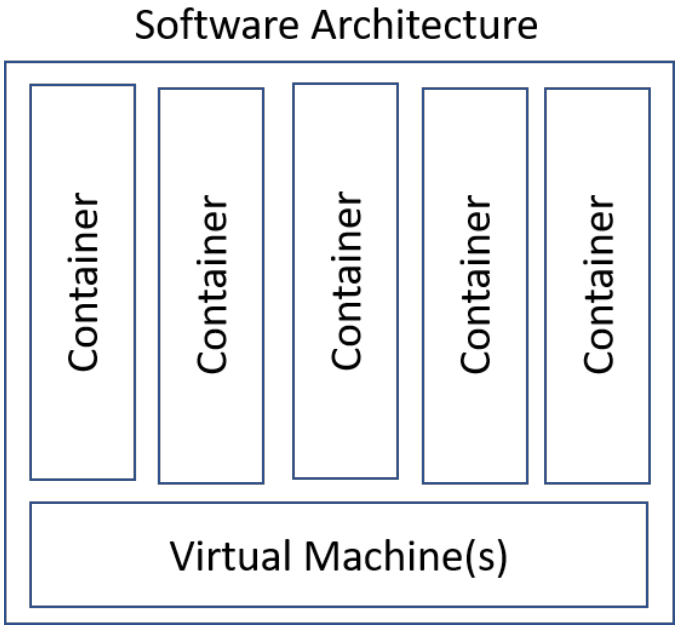

Fig. 2. Proposed software architecture

A. Use Case 1: Modsim + Data Analytics

Consider a case with a large scale physics simulation, such as LAMMPS [30]. LAMMPS is a molecular dynamics simulator that shows the behavior of molecules or atoms under various conditions. Depending on the physics complexity, the amount of data per process can vary widely. For this use case, assume the physics is moderately complex and LAMMPS is configured to run on a few dozen nodes. For example, a fracture simulator or a material melting would both be appropriate. In both cases, the material volume would be split across many processes, but the interaction with neighboring atoms and molecules affects what happens with each other. On several additional nodes, analysis code evaluates the data to determine the presence or absence of a fracture, determined by inter-atom distances, or does a visualization enabling storing just the image rather than the raw data.

LAMMPS would run normally pushing data to the analysis nodes periodically. With moderate physics, LAMMPS can use more on-node memory swapping from HBM to the PMEM to reduce the communication overheads. For the data analysis/visualization processes, they need the data to do their analysis and rendering making storage a better choice. For many of these routines, they are already written assuming they will read from storage rather than memory. Reconfiguring the PMEM to be storage devices enables these codes to run against PMEM as storage without modification.

B. Use Case 2: Cloud Data Analytics

For a large scale machine learning model generator, reading and re-reading LOTS of different data items can be necessary. To avoid contention in both the interconnect and on storage, it is best to pull the data down to the nodes once and then read

 ISBN: 978-1-61208-948-5

and re-read as appropriate. In this case, deploying all of the PMEM as storage devices makes the most sense.

C. Use Case 3: Traditional Modsim

Using the LAMMPS example again from above, consider if LAMMPS is running alone and just needs to push output to storage periodically. In this case, the PMEM could be configured partially as memory and partially as storage to serve as a staging area (burst buffer) that is sent asynchronously to off-node storage during the computation intensive phase.

D. Discussion

For all cases, the underlying hardware can be adapted to best suit the needs of the particular software involved. In the first case, the simulation code uses the PMEM as on-node fast swap, but pushes output to the PMEM on the nodes with data analytics code for further processing and ultimately permanent storage. In the second case, the PMEM is all allocated for read-ahead buffers accelerating the processing. The third case balances use between two needs enabling both more memory and node-local fast storage. These cases show the potential of the hardware/software design.

V. Related Work

Related work focuses on scheduling systems and some specialized platforms. First the various kinds of scheduling systems are briefly discussed and then the platforms are examined. This examined more in depth in previous work [6]. Task scheduling system generally fall into one of three categories with some newer systems attempting to bridge either the categories, platforms supported, or both.

A. Grid Scheduling

Grid systems predate clouds and were limited by deploying on system specific hardware and software. Cloud abstracted much of that away making it easier to move code from one system to another. In spite of that, the grid era has technology that is still relevant and useful today.

Globus [31] offers a tools suite to handle much of the data movement needs, but it does not offer the mature compute differentiation needed by these platforms.

To handle hybrid workloads, cross-platform grid scheduling was attempted [32]. This work, and all similar work, all suffered from complexity of different back-end system features and architectures, among other limitations. In short, it was unable to expose the desired functionality at a complexity level acceptable to users.

B. Scale Up Task Scheduling

Traditional HPC, scale-up workloads use long standard schedulers, such as Slurm [33]. As an open source tool with strong community support, it has grown to handle most HPC compute management needs. Slurm has grown new features, such as multi-resource scheduling [34], but that is different than what the gestalt is trying to do. For multi-resource scheduling, it assumes that a resource has a single purpose and just needs to be scheduled along with another resource (e.g., burst buffer capacity along with compute notes). We are proposing a different model with uniform hardware that is dynamically configured based on the workload needs. This eliminates the need for the more complex multi-resource scheduling.

A next generation HPC scheduler, Flux [35] is trying to offer a hybrid HPC and cloud scheduling capability. However, Flux is just starting to explore how to incorporate cloud resources and has only scratched the surface of the vast complexities involved. It will take considerable time and effort for Flux to successfully integrate a single cloud platform. Much like Slurm, this is focused on placing workloads on uniform platforms strictly optimized for one workload type or the other. Data movement issues are a daunting problem they have not addressed.

C. Scale Out Task Scheduling

The scale-out schedulers focus on large numbers of small tasks that often run quickly and exit. Scheduling throughput of 1000s of tasks per second is needed to keep the machine busy. Orchestration systems, discussed later, are a variation on this approach with some different constraints.

The need to handle scheduling a large number of short running tasks prompted the creation of Sparrow [36] and similar systems. This shift from the traditional very large single task with a long run time demanded these new tools to better handle resource use.

Spark [37] is another popular system that generally scales well. Mesos [38] with systems like Aurora [39] and Yarn [40] offer examples of high throughput task oriented schedulers.

Other systems like Omega [41] were built in frustration of the need to support heterogeneous clusters that evolved as new hardware was added over time with broken and obsolete hardware decommissioned. The priority for a system like this is to support a wide variety of hardware features and enable a reasonably efficient mapping from job requirements onto the available hardware balancing needs against availability. This is different from our goal of uniform hardware, but varying software.

Some modern software engineering architectures, such as function-as-a-service [42], embrace this kind of short task execution model as a central feature.

D. Container Orchestration

The alternative approach for job scheduling has shifted more to container management rather than task management. Systems like Kubernetes [3] and Docker Swarm [5] offer increasingly rich and complex environments for deploying long-lived services that can dynamically scale on demand. This architecture has compatibility issues with traditional HPC workloads, such as deployment time and potentially resource sharing. For service orchestration, all instances being fully deployed at nearly the same time is not as important as all of them running eventually, but soon. That makes this model less attractive to those wanting more efficient machine usage.

 ISBN: 978-1-61208-948-5

E. Hybrid Schedulers

New systems, like Fuzzball [29], intend to work like Flux, but are rethinking the environment from the ground up. While the ideas are inspiring, these systems need considerable work before they can achieve the goals outlined above. For example, the software system configuration for Fuzzball requires a full Kubernetes system install to manage the scale out workload and it does not address the very large scale data items in terms of moving compute to different kinds of resources. While these are planned to be addressed eventually, the fully needed functionality is still an unsolved and unimplemented problem.

F. Other Related Work

The literature that discusses HPC systems at national laboratories [43]–[45] provides rich information about HPC usage trends, resource utilization metrics, evolution of supercomputers over time, among many other user- and system-centered topics. Most of the existing studies on HPC environments—both historic and also recent—pay little attention to cloud computing and its benefits, considering integration with clouds to be secondary or optional in nature. With the shifting workload demands, revisiting these investigations is an important priority.

Among the counterexamples, a study of computing resources at the Texas Advanced Computing Center (TACC) [46] stands out as it describes a considerable number of cloud-style jobs being processed as a result of integration of TACC facilities with Jetstream [47], a cloud computing facility sponsored and managed by the USA's National Science Foundation (NSF).

Chameleon Cloud [48] offers a bare-metal-as-a-service cloud option. While this is an NSF supported effort focused on supporting both research and education, the resources are not as extreme as leadership computing facilities. Instead, the focus is on supporting smaller scale efforts with a strong tie to educational environments rather than production-style workloads. The bare metal aspect is quite appealing for our model as the full software stack is deployed at runtime on a per job basis. Shifting this to be more dynamic to a per node or per core basis would make this a fine contender for supporting the gestalt we propose.

An additional advantage of this approach concerns power usage [49]. By using PMEM devices, the energy footprint can be reduced. This also applies to ML applications [50].

The final part of the related work concerns cloud bursting. These systems look at how to use a cloud resource as "overflow" for an onsite or just another large scale compute resource. Work on these bursting approaches [51]–[53] show the challenges and potential for making these systems work. Microsoft, with the Azure platform, offer this as a fundamental part of their cloud strategy. They encourage users to install an Azure instance at their premises and to use Microsoft's private cloud instances as bursting capacity. This enables customers to right size their on site compute resources to control costs while not hitting limits for transient peak workloads. Achieving this kind of balance for HPC and Cloud would offer an excellent balance by deploying jobs that do not need the HPC platform characteristics onto the associated cloud when there is demand for the HPC platform specific characteristics by jobs. However, these capabilities still do not exist. Further, the most daunting problem of moving large data items from one platform to another is left completely unaddressed.

G. Discussion

The kinds of workloads each of these system classes addresses is different and difficult to address with a single scheduler and resource management system. This has led to the fragmentation of platform development efforts, where each platform is essentially treated as an independent direction for research and development and optimized to best address its own, particular subset of the workloads.

The need for a hybrid use platform as we describe exists today. With machine learning tools being incorporated into scientific simulations, both worlds must co-exist simultaneously on the same platform or latency will dominate the computations. For climate simulations [54], [55], machine learning models are substituting for parts of the model that may have too many parameters or the physics is not fully understood. Using models generated from observational data, reasonable estimates of these effects improve the simulation model quality overall. Using ML to perform initial analytics is also growing in popularity.

VI. Conclusions

Overall, we propose that using persistent memory devices, along with appropriate system software that can dynamically on a node-by-node, job-by-job basis make a single platform capable of efficiently handling both traditional modsim scale-up workloads coupled with data analytics scale out workloads. While persistent memory devices are currently cost prohibitive, NVMe devices offer a more affordable, and still relatively performant option that can work similarly with a little software help. We propose that this architecture be adopted for future systems.

Acknowledgements

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[1] G. F. Pfister, "An introduction to the infiniband architecture," *High performance mass storage and parallel I/O*, vol. 42, no. 617-632, p. 102, 2001.

[2] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the slingshot interconnect," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, IEEE, 2020.

[3] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the sixth ACM symposium on cloud computing*, pp. 167–167, 2015.

 ISBN: 978-1-61208-948-5

[4] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, IDEAS '14, (New York, NY, USA), p. 366–367, Association for Computing Machinery, 2014.

[5] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[6] J. Lofstead and D. Duplyakin, "Take me to the clouds above: Bridging on site hpc with clouds for capacity workloads," *CLOUD COMPUTING 2021*, p. 63, 2021.

[7] D. I. S. Agency, "Department of defense cloud computing security requirements guide." https://rmf.org/wp-content/uploads/2018/05/Cloud_Computing_SRG_v1r3.pdf, 2017.

[8] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, "Performance characterization of a dram-nvm hybrid memory architecture for hpc applications using intel optane dc persistent memory modules," in *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, (New York, NY, USA), p. 288–303, Association for Computing Machinery, 2019.

[9] D. Li, J. S. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, and W. Yu, "Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pp. 945–956, IEEE, 2012.

[10] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1537–1550, 2015.

[11] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proceedings of the Ninth European Conference on Computer Systems*, pp. 1–15, 2014.

[12] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 133–146, 2009.

[13] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, "An empirical guide to the behavior and use of scalable persistent memory," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 169–182, 2020.

[14] Y. Shan, S.-Y. Tsai, and Y. Zhang, "Distributed shared persistent memory," in *Proceedings of the 2017 Symposium on Cloud Computing*, pp. 323–337, 2017.

[15] A. Kalia, D. Andersen, and M. Kaminsky, "Challenges and solutions for fast remote persistent memory access," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 105–119, 2020.

[16] L. Logan, J. Lofstead, S. Levy, P. Widener, X.-H. Sun, and A. Kougkas, "pmemcpy: a simple, lightweight, and portable i/o library for storing data in persistent memory," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 664–670, 2021.

[17] K. Grimsrud and H. Smith, *Serial ATA Storage Architecture and Applications: Designing High-Performance, Cost-Effective I/O Solutions*. Intel press, 2003.

[18] D. Mayhew and V. Krishnan, "Pci express and advanced switching: evolutionary path to building next generation interconnects," in *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, pp. 21–29, 2003.

[19] T. Coughlin, "Evolving storage technology in consumer electronic products [the art of storage]," *IEEE Consumer Electronics Magazine*, vol. 2, no. 2, pp. 59–63, 2013.

[20] L.-x. Wang and J. Kang, "Mmap system transfer in linux virtual memory management," in *2009 First International Workshop on Education Technology and Computer Science*, vol. 1, pp. 675–679, IEEE, 2009.

[21] L. K. Foundation, "Linux Kernel Documentation: Direct Access for Files." https://www.kernel.org/doc/Documentation/filesystems/dax.txt, 2014.

[22] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320, 2006.

[23] S. Van Doren, "Hoti 2019: Compute express link," in *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pp. 18–18, IEEE, 2019.

[24] J. Russell, "PNNL, Micron Work on New Memory Architecture for Blended HPC/AI Workflows." https://www.hpcwire.com/2022/03/09/pnnl-micron-work-on-new-memory-architecture-for-blended-hpc-ai-workflows/, March 2022.

[25] L. V. Kale and S. Krishnan, "Charm++ a portable concurrent object oriented system based on c++," in *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pp. 91–108, 1993.

[26] M. E. Bauer, *Legion: Programming distributed heterogeneous architectures with logical regions*. Stanford University, 2014.

[27] J. Lofstead, J. Baker, and A. Younge, "Data pallets: containerizing storage for reproducibility and traceability," in *International Conference on High Performance Computing*, pp. 36–45, Springer, 2019.

[28] P. Olaya, J. Lofstead, and M. Taufer, "Building containerized environments for reproducibility and traceability of scientific workflows," *arXiv preprint arXiv:2009.08495*, 2020.

[29] CIQ, "Fuzzball: HPC-2.0." https://ciq.co/fuzzball/, 2022.

[30] S. Plimpton, P. Crozier, and A. Thompson, "Lammps-large-scale atomic/molecular massively parallel simulator," *Sandia National Laboratories*, vol. 18, p. 43, 2007.

[31] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997.

[32] D. M. Batista, N. L. S. da Fonseca, and F. K. Miyazawa, "A set of schedulers for grid networks," in *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, (New York, NY, USA), p. 209–213, Association for Computing Machinery, 2007.

[33] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*, pp. 44–60, Springer, 2003.

[34] Y. Fan, Z. Lan, P. Rich, W. E. Allcock, M. E. Papka, B. Austin, and D. Paul, "Scheduling beyond cpus for hpc," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 97–108, 2019.

[35] D. H. Ahn, J. Garlick, M. Grondona, D. Lipari, B. Springmeyer, and M. Schulz, "Flux: A next-generation resource management framework for large hpc centers," in *2014 43rd International Conference on Parallel Processing Workshops*, pp. 9–17, IEEE, 2014.

[36] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 69–84, 2013.

[37] D. Cheng, Y. Chen, X. Zhou, D. Gmach, and D. Milojicic, "Adaptive scheduling of parallel jobs in spark streaming," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9, IEEE, 2017.

[38] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center.," in *NSDI*, vol. 11, pp. 22–22, 2011.

[39] F. Pfeiffer, "A scalable and resilient microservice environment with apache mesos and apache aurora," *SREcon15 Europe*, May 2015.

[40] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, pp. 1–16, 2013.

[41] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 351–364, 2013.

[42] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 162–169, 2017.

[43] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari, "Job characteristics on large-scale systems: Long-term analysis, quantification and implications," in *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1186–1202, IEEE Computer Society, 2020.

[44] G. P. Rodrigo, P.-O. Östberg, E. Elmroth, K. Antypas, R. Gerber, and L. Ramakrishnan, "Towards understanding hpc users and systems: a nersc case study," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 206–221, 2018.

[45] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "On the diversity of cluster workloads and its impact on research results," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 533–546, 2018.

 ISBN: 978-1-61208-948-5

[46] N. A. Simakov, J. P. White, R. L. DeLeon, S. M. Gallo, M. D. Jones, J. T. Palmer, B. Plessinger, and T. R. Furlani, "A workload analysis of nsf's innovative hpc resources using xdmod," *arXiv preprint arXiv:1801.04306*, 2018.

[47] C. A. Stewart, T. M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, S. Tuecke, G. Turner, *et al.*, "Jetstream: a self-provisioned, scalable science and engineering cloud environment," in *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, pp. 1–8, 2015.

[48] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, *et al.*, "Lessons learned from the chameleon testbed," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pp. 219–233, 2020.

[49] R. Farber, "BOOSTING MEMORY CAPACITY AND PERFORMANCE WHILE SAVING MEGAWATTS." https://www.nextplatform.com/2020/12/08/boosting-memory-capacity-and-performance-while-saving-megawatts/, 2020.

[50] N. Hemsoth, "STORAGE PIONEER ON WHAT THE FUTURE HOLDS FOR IN-MEMORY AI." https://www.nextplatform.com/2021/01/18/storage-pioneer-on-what-the-future-holds-for-in-memory-ai/, 1 2021.

[51] T. Bicer, D. Chiu, and G. Agrawal, "A framework for data-intensive computing with cloud bursting," in *2011 IEEE international conference on cluster computing*, pp. 169–177, IEEE, 2011.

[52] A. Gupta, P. Faraboschi, F. Gioachin, L. V. Kale, R. Kaufmann, B.-S. Lee, V. March, D. Milojicic, and C. H. Suen, "Evaluating and improving the performance and scheduling of hpc applications in cloud," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 307–321, 2014.

[53] W. C. Proctor, M. Packard, A. Jamthe, R. Cardone, and J. Stubbs, "Virtualizing the stampede2 supercomputer with applications to hpc in the cloud," in *Proceedings of the Practice and Experience on Advanced Research Computing*, pp. 1–6, ACM, 2018.

[54] P. A. O'Gorman and J. G. Dwyer, "Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events," *Journal of Advances in Modeling Earth Systems*, vol. 10, no. 10, pp. 2548–2563, 2018.

[55] V. M. Krasnopolsky and M. S. Fox-Rabinovitz, "Complex hybrid models combining deterministic and machine learning components for numerical climate modeling and weather prediction," *Neural Networks*, vol. 19, no. 2, pp. 122–134, 2006.

 ISBN: 978-1-61208-948-5