Towards OS Heterogeneity Aware Cluster Management for HPC

Zhida An
Tsinghua University

Ding Li
Peking University

Yao Guo
Peking University

Guijin Gao
Huawei Technologies

Yuxin Ren
Huawei Technologies

Ning Jia
Huawei Technologies

Xinwei Hu
Huawei Technologies

ABSTRACT

To achieve extremely high performance in HPC, many researchers have proposed customized operating systems that are tailored to HPC workload characteristics and emerging hardware. Hence, we argue that the HPC cluster will move away from the single OS environment to a cluster with numerous heterogeneous OSes. However, existing HPC cluster management still assumes that all nodes are equipped with the same OS and fails to consider OS heterogeneity during job scheduling. As a result, such unawareness loses most performance benefits provided by specialized OSes.

This paper quantitatively investigates the problem of ignoring OS heterogeneity in the current HPC cluster management and analyzes performance trade-offs inside heterogeneous OSes. Preliminary results on a variety of HPC OSes and applications confirm the performance penalty of the existing cluster scheduler. We then propose a cluster scheduler prototype that incorporates OS heterogeneity into cluster configuration, resource monitoring, and job placement. We also present open challenges for future research on OS heterogeneity aware HPC clusters.

CCS CONCEPTS

• **Software and its engineering** → *Ultra-large-scale systems*; **Multiprocessing / multiprogramming / multitasking**.

KEYWORDS

HPC, Cluster Management, OS Heterogeneity, Job Scheduler

APSys '23, August 24–25, 2023, Seoul, Republic of Korea

ACM ISBN 979-8-4007-0305-8/23/08.
https://doi.org/10.1145/3609510.3609819

ACM Reference Format:
Zhida An, Ding Li, Yao Guo, Guijin Gao, Yuxin Ren, Ning Jia, and Xinwei Hu. 2023. Towards OS Heterogeneity Aware Cluster Management for HPC. In *14th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '23), August 24–25, 2023, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3609510.3609819

1 INTRODUCTION

High-performance computing (HPC) has become a fundamental infrastructure in many fields, such as scientific computing, digital finance and social governance. As it is super large-scale, a 1% performance improvement over HPC applications can save millions of dollars in computing expenses. To obtain optimal performance, customized OSes for HPC become increasingly popular in both industry and academia [7, 9, 10, 23]. However, customizing OS sacrifices generality and requires different OSes to accelerate different types of HPC applications. As a result, the HPC cluster will move away from the single general OS environment to a cluster with multiple heterogeneous specialized OSes [22].

In the current HPC scenarios, with the increasing complexity of applications and the growing demand for parallel computing, large-scale supercomputers and similar hardware capabilities are being divided into computational units through methods such as cloud computing and dynamic partitioning. Considering the existence of such computing scenarios, deploying a single OS on all nodes will inevitably result in performance degradation or functional deficiencies at the kernel level. Therefore, it is more appropriate to replace it with the deployment of diverse computing OSes on different computing nodes to cope with various potential computing tasks.

OS heterogeneity brings many challenges for HPC cluster management. Based on our industry practice, there are two key research problems that need more exploration.

1) *Existing cluster job scheduling is unaware of OS heterogeneity.* The cluster scheduler assumes that all computing nodes have the same OS installed with the same OS functionalities, capabilities, and features. Thus, the cluster scheduler fails to place jobs on nodes equipped

with the most suitable OS. According to our study, the lack of OS heterogeneity awareness may lose most performance benefits provided by specialized OSes and can even lead to negative performance penalties due to a mismatch between applications and OSes.

2) *It is difficult to configure, tune, and analyze OS heterogeneity.* Unlike the general OS, such as Linux, customized HPC OSes introduce lots of trade-offs tailored to HPC applications. Without proper specification, understanding, and adaptation of these trade-offs, cluster management cannot determine the best scheduling policy. Even worse, heterogeneous OSes involve much more factors and parameters to balance and tune, significantly increasing the complexity of scheduler management.

This paper conducts a quantitative study to investigate performance penalties due to OS heterogeneity unawareness in existing HPC cluster management and performance trade-offs inside heterogeneous OSes. Experiments on several HPC applications and OSes show that current cluster management systems (e.g., Slurm [34], a widely-used job scheduler) incur about 10% performance degradation compared to the ideal job placement (§3.1). Evaluation using mOS [33], a representative OS customized for HPC, demonstrates application performance trade-offs caused by different OS configurations. Improper configurations of specialized OSes even cause more than 230% performance variance (§3.2). Our studies confirm that the best scheduling mechanism for future HPC clusters depends on the awareness of OS heterogeneity and appropriate matching between OSes and applications.

Based on our studies, this paper advocates that OS heterogeneity needs to be incorporated into the whole scheduling cycle of HPC clusters, including user configuration, resource monitoring, and job placement (§3.3). During the design process of our cluster scheduler prototype, we identify some open challenges regarding efficient OS heterogeneity profiling, dynamic heterogeneity tuning, and support for more complicated scenarios. We discuss these challenges in §4.

The contributions of this paper include:

- We identify an unexplored research problem that OS heterogeneity is ignored by current HPC cluster management, causing significant performance degradation.
- We quantify the extent of performance loss due to OS heterogeneity unawareness and performance trade-offs of heterogeneous OSes with a variety of HPC OSes and applications.
- We propose an HPC job scheduling prototype and open research challenges to bring awareness of OS Heterogeneity to the HPC cluster.

2 BACKGROUND

2.1 Heterogeneous OS for HPC

Faced with a wide range of diverse HPC workloads, achieving the optimal performance requires for heterogeneous OSes to match OS implementation and policies with application characteristics. There are many research and industry projects to build specialized OSes. Some OSes customized kernel implementation based on Linux (e.g., CNK [11]Argo [26] and KPGO [35]), while other projects build new systems with different OS architectures, such as micro-kernel [12, 15], lightweight kernel [16, 29], and unikernel [17]. Furthermore, we can integrate different kernels into one OS via multi-kernel [7, 10] or kernel co-location [9, 23] approaches to generate more heterogeneous OSes, such as FusedOS[24], Hobbes[3], mOS[33], IHK/McKernel[8], and MySyS [14].

Unlike the general-purpose OSes that try to balance the performance of all possible application usage, specialized OSes usually optimize only a few types of applications and customize parts of OS functionalities.

For those specialized OSes, the configuration of kernel parameters has a significant impact on its performance. Different kernel parameter settings will have different effects on system resource allocation, functional characteristics, and other attributes. This results in varying performance and affinity for different application scenarios and tasks. In existing customized OSes targeting the HPC field, it is difficult to identify a unified trend or determine an optimal configuration for kernel parameters. Therefore, even with the same specialized kernel, OS heterogeneity can still be observed due to differences in kernel parameter configuration.

As a result, different OSes, and specialized OS with different kernel parameter configuration, exhibit a wide range of heterogeneity in many OS services and subsystems, such as scheduling policy, memory management, and inter-process communication. To facilitate heterogeneous OS development and deployment, some researchers propose a development, compilation, and deployment toolchain to help users easily implement and customize new systems [18]. However, heterogeneous OS management under a cluster environment is still an unexplored research problem.

2.2 Node Allocation in HPC

The management and allocation of massive computing resources play a critical role in optimizing the performance of HPC. For cluster management tools, the types and scales of potential tasks are unknown and uncertain. Therefore, it is difficult to incorporate multiple different node schedulers and corresponding scheduling algorithms through pre-defined methods. Additionally, considering that node allocation itself requires the occupation of computational resources on the

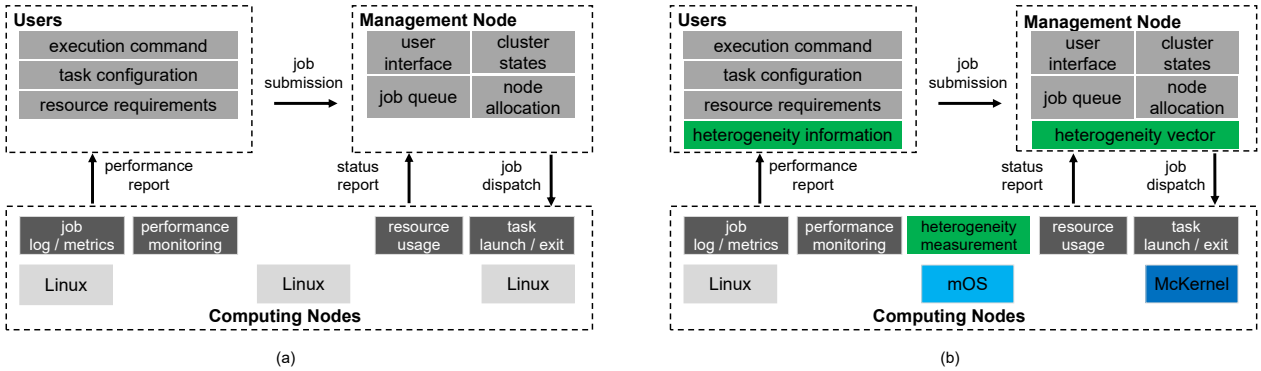

Figure 1: The general workflow of cluster scheduling. (a) represents existing schedulers that manage a cluster with a single OS; (b) is the proposed scheduler that incorporates OS heterogeneity throughout the entire management workflow.

management node, deploying multiple hierarchical schedulers would also result in a loss of computational resources. The above considerations have led to the adoption of various common default algorithms and a single scheduler in current mainstream strategies. Figure 1(a) shows the overall architecture of the general cluster scheduling workflow. Users submit jobs to the management node and provide task configuration and requirements via user interfaces. The management node realizes the core logic of cluster scheduling. It includes three key functionalities. 1) It sorts the job queue which essentially determines job priority. 2) It receives runtime statistics of computing nodes and analyzes cluster status, such as resource utilization and load balance. 3) It distributes jobs to computing nodes based on various scheduling policies. Computing nodes form the resource pool of the cluster. The OS on each node carries out the actual job execution. Additionally, a daemon collects and monitors system status, and communicates with the management node.

Existing HPC scheduling policies consider different factors to place jobs, such as multi-objective performance optimization [2], node topology [37], and resource fairness [19, 32]. However, all current strategies assume that a single OS is used by all computing nodes, they do not take OS heterogeneity into account when allocating nodes.

3 AWARENESS OF OS HETEROGENEITY

3.1 Cluster-wide OS Heterogeneity

Due to performance specificity, an HPC cluster will deploy multiple heterogeneous OSes to meet different application requirements. However, existing cluster management strategies ignore the OS heterogeneity of different nodes. Therefore, the current cluster scheduler fails to determine an optimal node allocation, resulting in performance losses.

To quantify the performance degradation in current cluster scheduling, we conducted evaluations using several HPC benchmarks in a heterogeneous cluster consisting of a management node and four computing nodes. Each node is a virtual machine with 16GB memory, 32 cores (AMD EPYC 7742 processor), and 100GB storage. The heterogeneous OS chosen is mOS with 16 isolated cores. mOS [33] runs both a Linux and a lightweight kernel (LWK) simultaneously on the same computer. A configurable number of cores can be reserved by the lightweight kernel, while other cores run Linux. The lightweight kernel executes performance-critical system calls and fast-path operations, and schedules applications in a run-to-completion, non-preemptive way. Heavyweight kernel operations and control interfaces are offloaded to Linux cores. A special command is needed to run applications with LWK features enabled, otherwise, the application will be executed as in normal Linux. We evaluate three typical HPC application benchmarks (HPCG [4], HPL [27], and miniFE [1]), all of which request two nodes. We compare two cases to account for the effects of OS heterogeneity: 1) Slurm. The cluster is scheduled by the default policy in Slurm, which ignores OS heterogeneity. 2) Best placement. We search for the best job placement and manually assign nodes accordingly.

Figure 2 illustrates the performance of applications as the number of mOS nodes increases, highlighting performance issues and management challenges in existing scheduling strategies. 1) Slurm is unaware of the specified OS. The scheduling policy of Slurm prioritizes the first available node regardless of whether it is a heterogeneous OS. For instance, Slurm selects Linux nodes when the cluster has less than 2 mOS nodes. Neglecting the mOS nodes leads to obvious performance penalty, and the performance of HPCG (HPL) is decreased by 10% (3%) compared to the best placement. This performance difference stems from the optimization of commonly used system calls in compute-intensive tasks by LWK, the dedicated scheduler, and the resource isolation features of mOS that reduce OS noise. 2) Slurm fails to leverage optimized features provided by customized OS. Even when

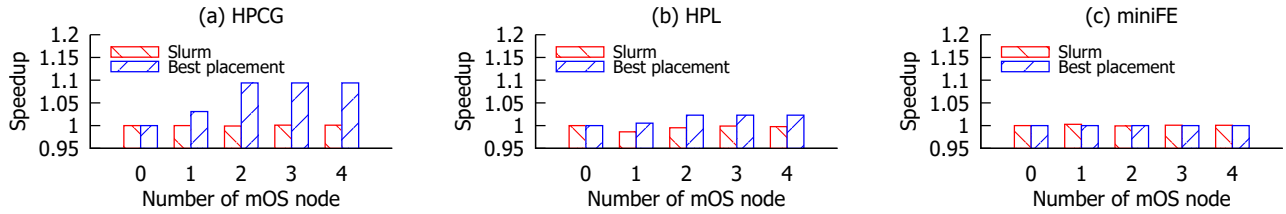

Figure 2: Application performance with the increasing number of mOS nodes. All data are normalized to the Slurm result with 4 Linux nodes.

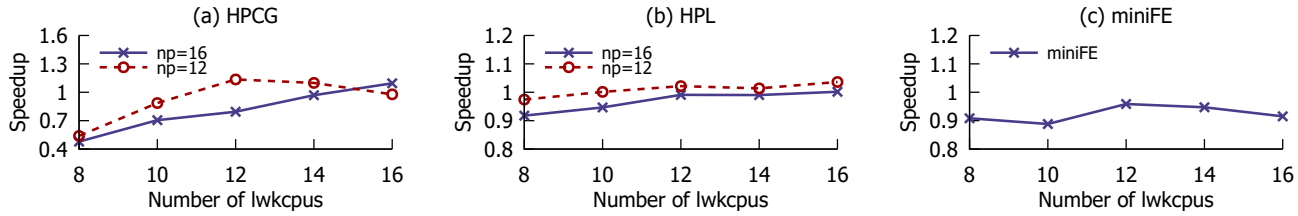

Figure 3: Application performance with different mOS configurations. All data are normalized to the result under Linux.

Slurm allocates mOS nodes (e.g., there are more than 3 mOS nodes), it fails to use the customized features in mOS, and dispatch jobs as in normal Linux instead, thereby negating most performance improvements realized through mOS. This is indicated by the fact that Slurm results are almost identical in the figure no matter how many mOS nodes the cluster has. 3) The best placement depends on the matching between OSes and applications. While the best strategy for HPCG and HPL is to allocate mOS node, miniFE requires Linux nodes to achieve best performance (Figure 2 (c)). One possible reason for this phenomenon is that miniFE has a higher communication intensity compared to others. LWK adopts an offloading or unmodified design for communication-related system calls. Consequently, miniFE frequently experiences inter-core interruptions during runtime, resulting in performance degradation. This demonstrates scheduling jobs among heterogeneous OSes needs to deeply understand the characteristics of both applications and OSes.

3.2 Trade-offs in Heterogeneous OS

Optimizations and features in customized OSes are typically designed specifically for certain application scenarios. While they can perform well in some specific cases, they may have limitations or even performance degradation in others. Additionally, heterogeneous OSes introduce much more parameters and configurations that need to be adjusted to obtain the best performance under different workloads. Thus, the cluster manager and scheduler have to comprehensively analyze heterogeneous performance properties and accurately configure the system, leading to a substantial increase in management complexity.

This section conducts a few experiments to study the performance sensitivity of heterogeneous OS configurations and performance trade-offs among different types of applications. The experiment utilized a single node with identical hardware configurations and HPC application benchmarks as discussed in §3.1. For HPCG and HPL, we measure the performance with parallelism degrees of 16 and 12. We adjust the 'lwkcpus' kernel parameter of mOS, which represents the number of isolated cores. Figure 3 reports the application performance that is normalized to the result of running benchmarks under Linux on the same node. The 'np' mark in the figure means parallelism. From the results, we can observe great performance variance caused by heterogeneous OS configurations and investigate the difficulties of scheduling heterogeneous OSes.

Properly configuring OS is challenging since different configurations have large impacts on application performance. As shown in Figure 3, the performance of all benchmarks varies when the number of isolated cores changes. mOS employs a non-preemptive execution model on isolated cores to maximize application throughput. Consequently, an application performs best when its parallelism is close to the number of isolated cores, otherwise, it may suffer substantial performance degradation. For example, HPCG for parallelism-12 achieves the best performance at 12 isolated cores and has 2.32 times higher performance than the worst configuration (8 isolated cores). When HPL has parallelism-16, its performance will decrease by 9.8% if using incorrect parameters.

Accurately matching application with OS is challenging since different applications require different configurations and even OS implementation. There are many factors determining if a task is suitable to execute on a specific OS. For example, task parallelism significantly impacts HPCG performance. With 12 isolated cores, there is a 43.6% performance improvement when changing the parallelism from 16 to 12. However,

the parallelism has little performance impact on HPL, which has at most 6% performance change. Another factor is the frequency of inter-process communication. Communication in mOS involves slow operations offloaded on the Linux core and incurs lots of system overhead to slow down the application. This is evident in the experimental results of miniFE (figure 3 (c)), where mOS always has worse performance than Linux, because miniFE has too many communication between processes.

In summary, deploying a diverse range of heterogeneous OSes and matching applications with appropriate OSes are essential for HPC clusters to guarantee optimal performance under different scenarios. Thus it is necessary to design new cluster schedulers that integrate OS heterogeneity effectively.

3.3 OS Heterogeneity Aware Cluster Scheduling

To deal with the OS heterogeneity of HPC, we propose a new idea for cluster management and scheduling. Based on the above analysis, we advocate that OS heterogeneity awareness should be incorporated into all key components along the whole workflow in cluster scheduling. Figure 1 (b) highlights enhancements and optimizations we propose.

Cluster configuration with OS information. The first step for achieving OS heterogeneity awareness in cluster scheduling involves extending the cluster configuration by having administrators input more information about heterogeneous OSes setup into a configuration file. In addition to basic node information such as naming, network, and hardware properties, the configuration file should also capture OS-related information deployed on each node, such as the OS type, features, and additional functional commands. These pieces of information are input into the scheduling system through a configuration file or other user interfaces, which are managed by the management node for subsequent node allocation processes.

Daemon process. The daemon process on each computing node monitors resource status, receives management instructions, and controls task execution. In our design, the daemon process collects roughly two categories of information: OS-related and runtime information about heterogeneous OS. OS-related information about heterogeneity refers to specialized functionalities and features provided by the OS, such as OS kernel parameters, resource quota, and system call specifications. The daemon process obtains this information from the configuration file and OS-exposed interfaces like `procfs` and `sysfs`, then sends it to the control center on the management node. The second category, runtime information, focuses on the node's heterogeneous characteristics during task execution (e.g., performance monitoring, metrics accounting, and usage of specific operations). The daemon process periodically obtains this data and sends it to the management node. Profiling on heterogeneous OSes is more challenging, as discussed in §4, requiring a new tracing framework and collection method.

Node allocation algorithm. Traditional policies focus on load balancing and matching resources with job requirements. We want to incorporate OS heterogeneity awareness into the traditional algorithm for improved job placement while ensuring compatibility with load balancing and execution correctness. The design principle is to treat OS heterogeneity as a resource characteristic and use the same matching algorithm to match it with the features of the job. The management node quantifies the received node information into a numerical value, builds a vector for each OS of the cluster's size, and places the value in the corresponding dimension to construct the node feature vector. Similarly, a feature dependence vector can be constructed based on the collected runtime data for each task using the same method. Based on these two vectors, it is possible to leverage the existing matching algorithm or use other heuristic vector operations to complete job distribution. A simple design of matching would be to use normalization and Euclidean distance matching. By normalization and calculating the Euclidean distance, the match between tasks and nodes can be quantified. This allows us to select the set of nodes with the smallest distance and highest match for task distribution. Certainly, this design is just a preliminary idea. More heuristic algorithms can be applied to improve the process.

Summary and discussion. The main difference between our proposal and existing methods, such as Slurm, is the collection and usage of more OS heterogeneity information. The types of target information to be collected are obtained through cluster configuration, while the information collection is performed by daemon processes deployed on computing nodes. Although the information collection process brings additional performance overhead, the collection of OS-related information for each node only needs to be done once during initialization, and the runtime information is only collected and statistically summarized once for each task. Therefore, it does not increase the time complexity. With the matching algorithm using the additional information, it is feasible to improve overall performance by leveraging the performance advantages to cover the associated costs. The possible open challenges will be discussed in §4.

Prototype implementation. Based on the proposed design, we implement a prototype by modifying Slurm. The current implementation uses mOS as an example of a heterogeneous OS. In the current prototype, we expand the configuration file to include the input of heterogeneous OSes and their additional functional instructions on different nodes. Cluster administrators can set the name, additional kernel parameters and acquisition method, extra functions, and instructions

of the heterogeneous OS on each node through the configuration file. When a job is dispatched to a node, the daemon is able to use the additional functions and special commands of the corresponding OS to launch the job. Additionally, we also instrument the daemon to collect more metrics about heterogeneous information in the OS. For instance, in mOS, we monitor the OS noise, the frequency of system call executed on isolated cores, and the utilization of Linux cores. On the management node, we implement a simple static matching strategy to place jobs on appropriate nodes. Preliminary results show that static matching can achieve similar performance to the best placement we studied in §3.1. Fully implementing the entire cluster scheduler with better user interaction, more efficient runtime heterogeneity analysis, and adaptive job matching is ongoing work. Supporting more heterogeneous OSes is left as future work.

4 OPEN CHALLENGES

We discuss some technical issues encountered during the design process of our prototype. These open challenges require more research effort in the future.

Efficient OS heterogeneity profiling. Profiling and analyzing OS heterogeneity is the foundation of HPC cluster management. However, there are a few difficulties in efficient heterogeneous perception. 1) There are no unified tracing frameworks across heterogeneous OSes. In Linux, there are many mature tools (e.g., `perf` and `ftrace`) to collect system statistics, but heterogeneous OSes have limited support for these tools. 2) Profiling on heterogeneous OSes incurs more performance overhead. Due to the heterogeneity, we have to monitor more events to understand all features and differences among heterogeneous OSes. 3) Analysis methods in the general system are no longer applicable. On heterogeneous systems, even similar metrics may reflect different properties of applications; thus it is necessary to design specific algorithms for different systems.

Dynamic heterogeneity tuning. Online adjusting system configurations [13, 36] is widely used to optimize the cluster performance under dynamic environments and changing workloads. For example, ACIC [20, 21] automatically searches for optimized I/O system configurations to accelerate HPC applications. OS heterogeneity brings more opportunities to achieve effective and flexible online tuning: 1) Heterogeneous OSes provide more factors to tune. In addition to system parameters, it is possible to tune policies of OS services (e.g., scheduling and page cache), subsystem implementation, or even OS architecture. With the expanded adjustment range, we have more feasible combinations of heterogeneous features to better respond to workload dynamics. 2) We can integrate more tuning tools under heterogeneity tuning. Existing tuning systems almost adjust parameters via `sysctl`, but in order to change other aspects of an OS, we need to integrate more approaches, such as hot patch, kernel hot upgrade with CRIU, and re-imaging system images.

Support for more complicated scenarios. Existing HPC clusters are always used in a partitioned and exclusive way, and assume HPC workload are long-running rigid applications. When we deploy multiple heterogeneous OSes in a single cluster, cluster-wide scheduling provides more chances to combine different capabilities of various OSes to support more diverse HPC application scenarios. 1) Task co-location. Consolidating multiple tasks onto the same node can significantly improve resource utilization [25, 30]. However, Linux incurs much interference among co-located tasks and decreases their performance [5]. With heterogeneity-aware scheduling, we are possible to achieve a better balance between QoS and resource utilization. 2) Rack-scale resource disaggregation. Rack-scale machines [6] promise efficient resource disaggregation but pose scalability and cache coherence challenges to existing OSes. Several systems have been proposed to manage disaggregated clusters to achieve great resource elasticity and sharing [12, 28, 31].

5 CONCLUSIONS

Specialized OSes become increasingly popular in HPC to achieve extremely high performance. Due to insufficient generality, single HPC cluster will deploy multiple different OSes to optimize different application scenarios. Hence, HPC clusters will be composed of more heterogeneous OSes. OS heterogeneity breaks a basic assumption in cluster management that all computing nodes have the same OS. Existing scheduling frameworks are not aware of various characteristics and limitations of heterogeneous OSes, leading to suboptimal resource allocation or misconfigured OS usage.

This paper quantitatively investigates performance degradation caused by OS heterogeneity unawareness. Evaluation using mOS and typical HPC benchmarks show about 10% performance penalties in a popular cluster scheduler, Slurm, and illustrate more than 230% performance variance in case of inappropriate OS configurations. These results confirm the necessity of introducing heterogeneity awareness into the whole cluster scheduling workflow. We propose potential enhancements to the existing cluster scheduler to realize the optimal resource allocation under heterogeneous OSes. We believe our proposal reveals more research challenges and opportunities for future heterogeneous HPC clusters.

ACKNOWLEDGMENT

This work was partly supported by National Key R&D Program of China (2022YFB4501802), NSFC (62141208), and a Huawei Research Fund.

REFERENCES

[1] 2017. miniFE Finite Element Mini-Application. https://github.com/Mantevo/miniFE. (2017).

[2] Sergey Blagodurov, Alexandra Fedorova, Evgeny Vinnik, Tyler Dwyer, and Fabien Hermenier. 2015. Multi-Objective Job Placement in Clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*.

[3] Ron Brightwell, Ron Oldfield, Arthur B. Maccabe, and David E. Bernholdt. 2013. Hobbes: Composition and Virtualization as the Foundations of an Extreme-Scale OS/R. In *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'13)*.

[4] Jack Dongarra, Michael Heroux, and Piotr Luszczek. 2019. High Performance Conjugate Gradient Benchmark (HPCG). https://github.com/hpcg-benchmark/hpcg. (2019).

[5] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, and Jian Pei. 2012. A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*.

[6] Paolo Faraboschi, Kimberly Keeton, Tim Marsland, and Dejan Milojicic. 2015. Beyond Processor-Centric Operating Systems. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems (HOTOS'15)*.

[7] Balazs Gerofi, Yutaka Ishikawa, Rolf Riesen, Robert W. Wisniewski, Yoonho Park, and Bryan Rosenburg. 2016. A Multi-Kernel Survey for High-Performance Computing. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'16)*.

[8] Balazs Gerofi, Masamichi Takagi, Atsushi Hori, Gou Nakamura, Tomoki Shirasawa, and Yutaka Ishikawa. 2016. On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*.

[9] Balazs Gerofi, Masamichi Takagi, Yutaka Ishikawa, Rolf Riesen, Evan Powers, and Robert W. Wisniewski. 2015. Exploring the Design Space of Combining Linux with Lightweight Kernels for Extreme Scale Computing. In *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'15)*.

[10] Balazs Gerofi, Kohei Tarumizu, Lei Zhang, Takayuki Okamoto, Masamichi Takagi, Shinji Sumimoto, and Yutaka Ishikawa. 2021. Linux vs. Lightweight Multi-Kernels for High Performance Computing: Experiences at Pre-Exascale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'21)*.

[11] Mark Giampapa, Thomas Gooding, Todd Inglett, and Robert W. Wisniewski. 2010. Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*.

[12] Matthias Hille, Nils Asmussen, Hermann Härtig, and Pramod Bhatotia. 2020. A Heterogeneous Microkernel OS for Rack-Scale Systems. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'20)*.

[13] Ajaykrishna Karthikeyan, Nagarajan Natarajan, Gagan Somashekar, Lei Zhao, Ranjita Bhagwan, Rodrigo Fonseca, Tatiana Racheva, and Yogesh Bansal. 2023. SelfTune: Tuning Cluster Managers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*.

[14] Yauhen Klimiankou. 2022. Towards Practical Multikernel OSes with MySyS. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'22)*.

[15] Adam Lackorzynski, Carsten Weinhold, and Hermann Härtig. 2016. Decoupled: Low-Effort Noise-Free Execution on Commodity Systems. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'16)*.

[16] John Lange, Kevin Pedretti, Trammell Hudson, Peter Dinda, Zheng Cui, Lei Xia, Patrick Bridges, Andy Gocke, Steven Jaconette, Mike Levenhagen, and Ron Brightwell. 2010. Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing. In *2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS'10)*.

[17] Stefan Lankes, Simon Pickartz, and Jens Breitbart. 2016. HermitCore: A Unikernel for Extreme Scale Computing. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'16)*.

[18] Conghao Liu and Kyle C. Hale. 2019. Towards a Practical Ecosystem of Specialized OS Kernels. In *Proceedings of the 9th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'19)*.

[19] Haikun Liu and Bingsheng He. 2014. Reciprocal Resource Fairness: Towards Cooperative Multiple-Resource Fair Sharing in IaaS Clouds. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*.

[20] Mingliang Liu, Ye Jin, Jidong Zhai, Yan Zhai, Qianqian Shi, Xiaosong Ma, and Wenguang Chen. 2013. ACIC: Automatic cloud I/O configurator for HPC applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*.

[21] Mingliang Liu, Jidong Zhai, Yan Zhai, Xiaosong Ma, and Wenguang Chen. 2011. One Optimized I/O Configuration per HPC Application: Leveraging the Configurability of Cloud. In *Proceedings of the Second Asia-Pacific Workshop on Systems (APSys'11)*.

[22] Arthur B. Maccabe. 2017. Operating and Runtime Systems Challenges for HPC Systems. In *Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'17)*.

[23] Jiannan Ouyang, Brian Kocoloski, John R. Lange, and Kevin Pedretti. 2015. Achieving Performance Isolation with Lightweight Co-Kernels. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'15)*.

[24] Yoonho Park, Eric Van Hensbergen, Marius Hillenbrand, Todd Inglett, Bryan Rosenburg, Kyung Dong Ryu, and Robert W. Wisniewski. 2012. FusedOS: Fusing LWK Performance with FWK Functionality in a Heterogeneous Environment. In *Proceedings of the 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'12)*.

[25] Tirthak Patel and Devesh Tiwari. 2020. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*.

[26] Swann Perarnau, Judicael A. Zounmevo, Matthieu Dreher, Brian C. Van Essen, Roberto Gioiosa, Kamil Iskra, Maya B. Gokhale, Kazutomo Yoshii, and Pete Beckman. 2017. Argo NodeOS: Toward Unified Resource Management for Exascale. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS'17)*.

[27] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. 2018. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. https://netlib.org/benchmark/hpl/. (2018).

[28] Yuxin Ren, Gabriel Parmer, and Dejan Milojicic. 2020. Ch'i: Scaling Microkernel Capabilities in Cache-Incoherent Systems. In *2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'20)*.

[29] Rolf Riesen, Arthur Barney Maccabe, Balazs Gerofi, David N. Lombard, John Jack Lange, Kevin Pedretti, Kurt Ferreira, Mike Lang, Pardo Keppel, Robert W. Wisniewski, Ron Brightwell, Todd Inglett, Yoonho Park, and Yutaka Ishikawa. 2015. What is a Lightweight Kernel?. In *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'15)*.

[30] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant Kale. 2014. Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*.

[31] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*.

[32] Xiongchao Tang, Haojie Wang, Xiaosong Ma, Nosayba El-Sayed, Jidong Zhai, Wenguang Chen, and Ashraf Aboulnaga. 2019. Spread-n-Share: Improving Application Performance and Cluster Throughput with Resource-Aware Job Placement. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'19)*.

[33] Robert W. Wisniewski, Todd Inglett, Pardo Keppel, Ravi Murty, and Rolf Riesen. 2014. MOS: An Architecture for Extreme-Scale Operating Systems. In *Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'14)*.

[34] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing, 9th International Workshop (JSSPP'03) (Lecture Notes in Computer Science)*, Vol. 2862. Springer, 44–60. https://doi.org/10.1007/10968987_3

[35] Pengfei Yuan, Yao Guo, Lu Zhang, Xiangqun Chen, and Hong Mei. 2018. Building application-specific operating systems: a profile-guided approach. *Science China Information Sciences* 61, 9 (13 Aug 2018), 092102.

[36] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards Dynamic and Safe Configuration Tuning for Cloud Databases. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD'22)*.

[37] Christopher Zimmer, Saurabh Gupta, Scott Atchley, Sudharshan S. Vazhkudai, and Carl Albing. 2016. A Multi-faceted Approach to Job Placement for Improved Performance on Extreme-Scale Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16)*.