SLURM Deployment in Cloud Environments: Enhancing Utilization and Scalability

Márk Emődi[1,2,*], Konrád Bánfi[1], József Kovács[1]

[1] HUN-REN Institute for Computer Science and Control (HUN-REN SZTAKI), Hungarian Research Network, Kende utca 13-17, H-1111 Budapest, Hungary; {mark.emodi, banfi.konrad, jozsef.kovacs}@sztaki.hun-ren.hu

[2] John von Neumann Faculty of Informatics, Óbuda University, Bécsi út 96/b, H-1034 Budapest, Hungary; emodi.mark@nik.uni-obuda.hu

* Corresponding author

Abstract: Addressing the challenges of managing high-performance computing workloads in dynamic cloud environments, this paper presents a SLURM-based reference architecture. We elaborated Infrastructure as Code (IaC) to automate the deployment and management of SLURM, enabling efficient resource allocation and scalability. The basic scheduler architecture descriptor was further extended with computational tools and frameworks required by the HUN-REN Cloud scientific community. Results from benchmark experiments show significant performance improvement through parallelization, demonstrating SLURM's ability to utilize cloud resources for fair workload management of calculation-heavy tasks. Our AlphaFold protein structure prediction experiments demonstrate an 82.1% reduction in computational runtime when scaling from 1 to 8 worker nodes, with execution time decreasing from 3154.75 seconds to 563.75 seconds.

Keywords: SLURM; Cloud Computing; HPC; Infrastructure as Code; IAC; Scalability; Reference Architecture

1 Introduction

The HUN-REN Cloud [1] provides high-performance scientific computing for the Hungarian research community. This scientific cloud has already supported around 400 research projects by providing resources and support to researchers. To support the rapid development of AI research, there is a growing demand for computational capacities. As resources within the HUN-REN Cloud are limited, it is crucial to ensure that researchers have equal access to them. The purpose of the job scheduler is to enable the efficient, dynamic, and fair allocation of resources among researchers in this multi-user computing environment.

These scheduling mechanisms are critical components in modern computing environments, particularly in HPC and distributed systems. They allocate capacities efficiently, manage job execution sequences, and optimize system utilization. These factors are essential to improve performance, reduce wait times, and ensure fairness among users [2]. To understand why introducing such scheduling logic is beneficial to workloads, we need an examination of some interconnected aspects, including resource efficiency, adaptability to workload changes, and optimization of job completion times, as these aspects are all influencing performance metrics in distributed system environments.

The main purpose of job scheduling involves maximizing resource efficiency along with reducing the duration of job waiting time. These platforms orchestrate resource allocation and task execution, affecting overall system utilization and job turnaround time. Zhao et al. [3] demonstrate that job scheduling effectiveness determines QoS quality and system performance thus requiring optimal job allocation strategies to reach high efficiency levels. The selection of job scheduling policies through prioritization affects both user experience and resource utilization efficiency, demonstrating direct link between scheduling methods and system performance [4].

High-performance computing (HPC) environments excel in processing power through optimized hardware configurations, low-latency interconnects, and system architectures specifically designed for intensive scientific workloads. In contrast, cloud computing platforms offer superior portability and scalability for variable workloads. This creates a fundamental trade-off where HPC systems maximize processing power at the expense of flexibility, whereas virtualized systems prioritize accessibility and resource elasticity at the potential cost of peak performance.

Our primary research objective was to create a computing platform for job scheduling that combines the advantages of both HPC and cloud technologies. We aimed to develop a solution that enables a more dynamic allocation of computational resources in multi-user cloud environments, effectively minimizing idle time (e.g. CPU, GPU) while maintaining system stability and providing access across different user groups and research teams. To achieve this integration, we implemented SLURM [5] (Simple Linux Utility for Resource Management) as a reference architecture (see the specification in next section), providing a robust on-demand deployment management layer that bridges traditional HPC workload management with cloud elasticity. According to current studies, SLURM remains an ideal solution for HPC environments, based on its workload management efficiency and parallel execution criteria [6].

To enhance automation, scalability, provide configuration consistency and maintain deployment reproducibility across our computing platform, we adopted the Infrastructure as Code (IaC) paradigm. This methodology allowed us to define and manage our entire infrastructure through machine-readable definition files rather than physical hardware configuration or interactive configuration tools.

By codifying our system specifications, we achieved greater consistency between deployments, so the SLURM platform could be further developed, tested and maintained more efficiently, and rolled back when necessary.

The rest of the paper is organized as follows: Section 2 presents the bridging of traditional HPC and cloud environments, followed by the reference architecture implementation. Section 3 describes our measurement methodology and experimental results using AlphaFold protein structure prediction as a benchmark across varying node configurations. Section 4 reviews related work in container orchestration and HPC workload management, comparing our approach with existing Kubernetes and SLURM implementations.

2 SLURM Architecture and Cloud Integration

2.1 Bridging Traditional HPC and Cloud Environments

High-performance computing (HPC) infrastructures are widely used in scientific communities. Traditionally, these systems have been deployed to execute calculation-heavy tasks, such as genome sequencing [7], neural network pre-training [8], and quantum simulations [9] in multi-user environments where performance and execution speed are crucial. In these settings, programs run as close to the hardware as possible, avoiding virtualization layers that would otherwise compromise performance.

Cloud computing represents a paradigm shift in how computational assets are utilized. It offers on-demand access to a shared pool of configurable computing resources that can be rapidly provisioned with minimal management effort. The evolution of processing environments has transformed HPC from isolated computing clusters to more distributed and accessible systems, creating new opportunities for scientific research. Traditional HPC environments are characterized by fixed resources, specialized hardware, and batch-oriented workloads. Users would submit jobs to a queue system and wait for resources to become available. In contrast, cloud environments offer dynamic provisioning, scalability, and better availability, which has fundamentally changed the way processing assets are used. These platforms enable researchers to dynamically adjust processing capacities based on workload requirements [10]. This elasticity allows efficient handling of burst computing needs without the constraints of physical system limitations. These infrastructures allow researchers to access computational resources regardless of geographical location, providing access to high-performance computing capabilities that were previously limited by several factors.

Despite the potential benefits, several challenges persist in using cloud resources for high-performance computing workloads. Configuration complexity represents a significant barrier, as cloud environments offer numerous configuration options that can dramatically affect performance and cost. Navigating these options requires specialized knowledge that may not be common among traditional HPC users. The abstraction layers inherent in cloud computing introduce performance penalties that can be particularly problematic for communication-intensive applications. Virtualization overhead, shared resources, and network virtualization can all contribute to decreased performance compared to bare-metal deployments. These layers also introduce variability in performance that complicates benchmarking and optimization efforts.

2.2 Slurm Reference Architecture

Reference Architectures (RAs), also known as Blueprints[1], are recurring patterns and best practices that can be reused across different contexts with minimal configuration but customization ability. They simplify the deployment of complex architectures and offer user-friendly customization options to enhance flexibility and ease of use. One of the main goals of the RA is to provide a scalable and flexible approach to managing on-demand computational resources for high-performance computing workloads, combining the cloud versatility with the higher resource utilization that job scheduling might offer. Figure 1 illustrates the high-level architecture of the workload manager deployed within a virtualized infrastructure. A central SLURM controller node contains the main components of SLURM: slurmdbd, slurmd, and slurctld daemons, and manages one or more compute nodes. (i) Slurmdbd is responsible for managing the SLURM database. This database stores historical and current information about job accounting, resource usage, and cluster status. (ii) Slurmd runs on each compute node in the SLURM cluster. Its primary role is to manage the resources (CPU, memory, GPUs, etc.) on its local node and execute the tasks assigned to it by the SLURM controller. (iii) Slurmctld is the central management process for the entire cluster. It receives job submission requests, makes scheduling decisions, allocates resources, and monitors the overall health and status of the cluster. Typically, there are one primary slurmctld and optionally one backup for high availability.

The deployment steps were the following: First, we defined the required cloud resources in Terraform files and built the infrastructure accordingly. Once the deployment was complete, Terraform automatically updated the hosts file inventory used by Ansible. As a second step, Ansible then executed the installation and configuration tasks needed by SLURM. During the design phase, several critical challenges had to be addressed to ensure a robust and functional architecture. One major concern was data synchronization between nodes, which is essential to

[1] https://doc.slices-sc.eu/blueprint/

maintain consistency and enable coordinated processing in distributed environments. Another key challenge we faced is the synchronization of the configuration, that is, ensuring that all nodes share identical system settings, environment variables, and software configurations to avoid runtime issues, as described in our earlier work [11]. Establishing a reliable mechanism for service synchronization was also necessary, particularly to start, stop, and monitor services in a coordinated manner across the cluster. Furthermore, implementing secure network firewalling between nodes was crucial to restrict unauthorized access and protect internal communication channels. The system also required installation of global components, such as container runtime environments, which needed to be deployed and accessible from all nodes. Finally, service-level authorization had to be established to enforce proper access controls and ensure that only trusted components could interact within the system.

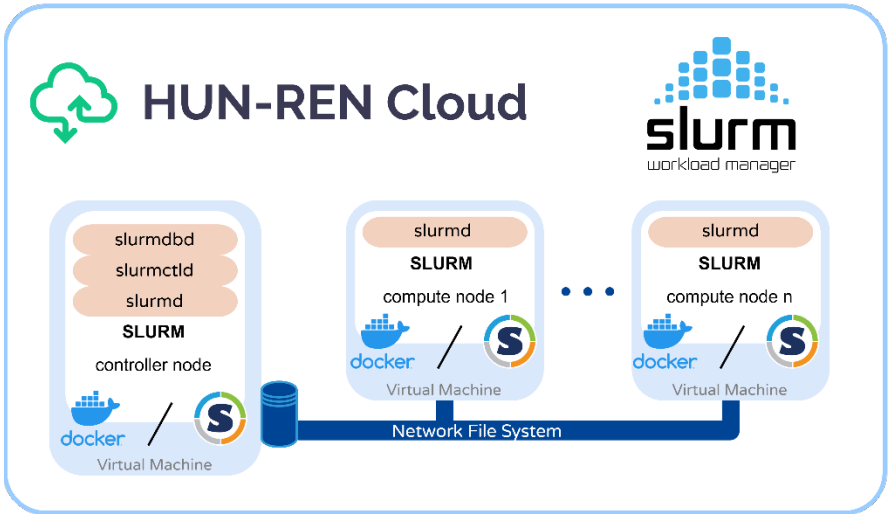

Figure 1
SLURM reference architecture

To resolve the issues related to data synchronization, we configured a central NFS server for data storage purposes. To resolve configuration synchronization, we used the Infrastructure as Code paradigm. IaC transforms traditional manual provisioning into programmable, version-controlled processes. It provides automation of deployment workflows, on-demand resource utilization, provisioning, scalability, reproducibility and the mitigation of human errors. When selecting appropriate IaC tools for the deployment and configuration of the workload manager, many factors were considered. We aimed for cloud agnosticism to prevent vendor lock-in and open-source solutions to benefit the academic and scientific communities. The efficiency and performance of deployment were also considered. Based on these criteria, our implementation leverages Terraform for infrastructure provisioning and Ansible for configuration management. Terraform

provides declarative configuration for creating and managing the core infrastructure components, including virtual machine instances as computation nodes, persistent storage devices for data management and security groups for defining network policies. A significant challenge in traditional SLURM deployments is the static nature of configuration files. Our solution implements template-based configuration files and automated service restart mechanisms managed through Ansible. This dynamic approach allows administrators to modify SLURM configurations without manual intervention across multiple nodes, significantly reducing management overhead while improving reliability. Continuous Integration and Continuous Deployment (CI-CD) pipelines play a crucial role in modern infrastructure management by automating testing, validation, and deployment processes, thereby reducing human error and ensuring consistent system behaviour across different environments. Our CI-CD pipeline uses GitLab runners to perform periodic, weekly automated tests for streamlined SLURM infrastructure deployment, configuration, and deletion with the help of the Infrastructure as Code repository.

Our workload-management implementation incorporates several complementary technologies to extend its functionality. While it runs natively on the provisioned virtual machines, Singularity [12] [13] container integration provides user-level dependency management, application isolation and the portability of computational workloads across different infrastructures. In comparison to Docker, Singularity is more suitable for HPC workloads due to its seamless integration with existing HPC schedulers and resource managers such as SLURM. In addition, the rootless execution model eliminates security vulnerabilities associated with privileged container operations.

To support interactive computing workflows, we also integrated JupyterLab environments, packaged as Singularity containers. These workflows serve as web-based development interfaces accessible through the SLURM scheduling system. The solution also incorporates robust support for parallel computing paradigms through OpenMP integration for shared-memory parallelism and OpenMPI configuration for distributed-memory parallel computing. The integration of MPI with SLURM is facilitated through the Process Management Interface for Exascale (PMIx) framework, which enables direct communication between the SLURM scheduler and MPI runtime environments, allowing for interprocess communication management across compute nodes. Furthermore, Singularity containers support MPI workloads by maintaining compatibility with the host system's MPI implementation. Through bind mounting of necessary libraries and communication endpoints, MPI processes can execute within isolated container environments while preserving the low-latency, high-bandwidth communication essential for scalable distributed computing applications.

2.3 Operational Management and Monitoring

Cloud-based HPC systems present unique security challenges at both system and user levels that require comprehensive management approaches. Our SLURM reference architecture addresses these challenges through integrated security, resource management, and monitoring frameworks. Data privacy is enforced through strict file system permissions, ensuring that each user can only access designated folders and project directories. This isolation prevents unauthorized access to sensitive research data and maintains confidentiality across different research groups. The Infrastructure as Code (IaC) paradigm enhances security management by providing consistent and auditable infrastructure configurations. Security patches and system updates are systematically deployed across the cluster through automated Ansible playbooks, ensuring that all nodes maintain the same security baseline. This approach eliminates configuration drift and reduces potential security vulnerabilities caused by inconsistent manual updates.

The architecture also incorporates Quality of Service policies to ensure fair resource allocation among users and research groups. QoS configurations define priority levels, resource limits, and scheduling preferences that govern job execution order and resource consumption, preventing individual users or projects from monopolizing cluster resources while guaranteeing minimum service levels for critical research activities. Storage quota management on the NFS server enforces disk usage limits for individual users and projects, preventing storage exhaustion and ensuring equitable access to shared storage resources. These quotas are dynamically configurable based on project requirements and available storage capacity. Partition visibility configuration controls which computational resources are accessible to different user groups, enabling administrators to reserve specialized hardware for specific research domains or priority projects.

To ensure optimal resource utilization and system health, our solution implements a containerized monitoring framework. Due to Docker Compose qualities, the management and version control of monitoring services became more streamlined and transparent. This approach significantly improves service scalability and maintainability compared to traditional monitoring deployments. The monitoring architecture implements Dockerized node-exporter and slurm-exporter agents across computing nodes, Prometheus time-series database for metrics collection and Grafana dashboards for visualization of resource utilization patterns. The Docker files were also deployed and managed through Ansible template modules to enhance automation and to reduce manual administration overhead. The SLURM exporter metrics provide detailed insights into job scheduling efficiency, queue states, and node utilization patterns specific to the workload manager. These metrics are essential for understanding cluster performance and identifying potential bottlenecks in the scheduling process. Customized Grafana dashboards were developed to present comprehensive visualizations of processing power utilization across the entire cluster. These dashboards display real-time metrics including GPU, CPU and memory usage, job completion rates, and queue wait times, enabling

administrators to make informed decisions about infrastructure scaling and system configuration. The containerized monitoring approach ensures that the monitoring infrastructure can scale alongside the computational resources while maintaining consistent configuration management through Infrastructure as Code principles.

3 Measurement

As an experiment, the aim was to create a parallel execution platform for AlphaFold [14]. We investigated the runtime speedup and node resource utilization efficiency using node level parallelism across the SLURM computing cluster. The experiment examined CPU utilization and scaling efficiency by doubling the number of compute nodes from 1 to 8 (1n, 2n, 4n, and 8n configurations) while using Round-Robin scheduling logic. During the measurement process, we had a maximum of 8 nodes to run the experiment. Each experimental configuration was executed five times, and the results were averaged to ensure statistical reliability. Computational resources were standardized with each node allocated 4 vCPUs, and the SLURM parameter *--cpus-per-task* set to 4 to maintain thread level parallelism across all nodes consistently. The workload consisted of 403 separate Protein Data Bank (PDB) files, distributed between all nodes. In other words, 403 individual job steps were scheduled against the computational cluster. Each prediction task included the extraction and analysis of predicted Local Distance Difference Test (pLDDT) confidence scores to assess prediction quality. These scores were visualized and statistically analysed to assess the structural reliability of the predicted models. For our parallel execution experiments, we used organism proteomes, specifically the Homo sapiens proteome dataset, which contains 23,391 predicted protein structures [14].

AlphaFold, developed by DeepMind, has revolutionized the field of three-dimensional protein structure analysis by achieving unprecedented accuracy in predicting protein structures from amino acid sequences [15]. However, the processing demands of AlphaFold are substantial, particularly when processing large datasets of protein sequences. PDB files contain the structural information of proteins, including 3D atomic coordinates. In the context of AlphaFold, these files store predicted protein structures along with confidence metrics such as pLDDT scores. These confidence scores are crucial for researchers to evaluate the reliability of predictions before applying them in downstream analyses or experimental validation. Since the computational demands of such operations are intensive, optimizing execution efficiency through parallel computing strategies is essential to make AlphaFold more accessible and practical for biological research.

The data presented in Table 2 provides a quantitative analysis of the performance scaling of a benchmark task, executed using a SLURM scheduler across varying numbers of worker nodes.

Table 2
PERFORMANCE METRICS WITH VARYING NUMBER OF NODES

Number of nodes	Average execution time	Average CPU utilization	Median CPU utilization	CPU *P90*
1	3154.75 s	74.45 %	78.90 %	81.50 %
2	2186 s	72.25 %	81.50 %	86.80 %
4	1144.50 s	56.51 %	57.20 %	82.10 %
8	563.75 s	49.86 %	51.10 %	76.97 %

The table details the average execution time, average CPU utilization, median CPU utilization, and the 90th percentile CPU utilization for node configurations of 1, 2, 4, and 8. Examining the CPU utilization metrics offers further insight into the resource usage efficiency across different node configurations. The average CPU utilization generally shows a decreasing trend as the number of nodes increases. The single-node configuration exhibited the highest average CPU utilization at 74.45%, suggesting a more intensive use of the available resources on that single machine. The median CPU utilization provides a measure of the central tendency of CPU usage, which is less susceptible to outliers. The 90th percentile CPU utilization offers a view of the peak resource consumption. The values suggest that while the average utilization decreases with more nodes, the peak utilization can still be relatively high, particularly for the two and four-node configurations. This information is crucial for understanding potential bottlenecks and ensuring sufficient resource provisioning.

Figure 2 presents the mean computational runtime as a function of the number of active worker nodes (1, 2, 4, and 8) used by a SLURM scheduler for the benchmark task. The y-axis indicates the average runtime in seconds, while the x-axis displays the node configurations. Each bar represents the mean runtime derived from multiple measurements for that specific node count. Error bars surmounting each bar denote the standard deviation of the runtimes, illustrating the variability across trials. Data labels are provided above each error bar, indicating the precise mean runtime value.

An inverse relationship between the number of active worker nodes and the mean runtime is noticeable. The single-node configuration (Node 1) exhibited the longest mean runtime at 3154.75±68.15 seconds. Increasing the worker count to two nodes (Node 2) reduced the mean runtime to 2186.00±60.08 seconds. A further increase to four nodes (Node 4) resulted in a mean runtime of 1144.50±43.08 seconds. The shortest mean runtime was achieved with eight nodes (Node 8), recording 563.75±28.18 seconds. The absolute magnitude of the standard deviation also decreased with an increasing number of nodes, suggesting a reduction in the absolute spread of execution times as parallelism increased.

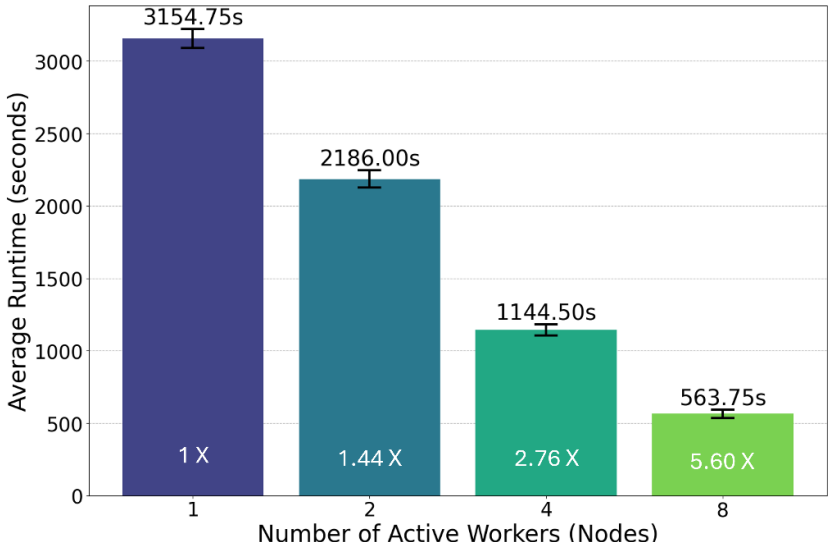

Figure 2

Relationship between the number of worker nodes and average computational runtime. The average speedup value, relative to the baseline (1 node) runtime, is indicated by an 'X' sign within each bar

When distributing calculation-heavy tasks across multiple worker nodes, several factors can lead to diminishing returns in parallelization efficiency. As the number of nodes increases, the certain bottlenecks often emerge. For example, when multiple nodes simultaneously attempt to access shared storage resources, contention for read/write operations can severely limit performance. This bottleneck is particularly pronounced in data-intensive applications where file access becomes the limiting factor rather than computational capacity. Potential solutions include implementing distributed file systems such as GlusterFS or Lustre, deploying multiple NFS servers, or using parallel file systems to distribute I/O load across multiple storage nodes. As the node count increases, communication overhead grows significantly. The network fabric connecting nodes can also become saturated with inter-node messages, leading to increased latency and reduced throughput. As the node count increases, communication overhead grows significantly. The network fabric connecting nodes can also become saturated with inter-node messages, leading to increased latency and reduced throughput. This is especially problematic in tightly coupled parallel applications that require frequent synchronization. Furthermore, when individual tasks are too small, the administrative overhead of task management can exceed the processing benefits of parallelization. Heterogeneous task durations can lead to substantial idle time on some nodes while others continue processing. For example, if some PDB files require significantly more analysis time than others, nodes that finish early must wait for the slowest tasks to complete before the entire job can finalize effectively limiting parallelization benefits.

4 Related Work

The integration of container technologies with HPC environments represents a significant technological advancement in scientific computing. Multiple research teams have explored various orchestration solutions for containerized HPC workloads, each with their unique advantages and limitations.

Milroy et al. [16] made substantial progress in addressing Kubernetes scalability for HPC applications. The team enhanced the MPI Operator to achieve an impressive 3,000-rank scale, representing a 100× improvement over previous implementations. This advancement is particularly significant as MPI remains the dominant programming model for distributed memory parallel computing in scientific domains. Their Fluence scheduler plugin demonstrated up to 3× better performance for scientific workflows compared to standard Kubernetes scheduling, optimizing placement based on node proximity, network topology, and specialized handling of GPU resources and Infiniband networking. Despite these improvements, Kubernetes in HPC environments still faces challenges. Milroy's work revealed a persistent scheduling overhead of 10-15%, increased configuration complexity, and integration difficulties with existing HPC infrastructure. Kubernetes provides superior high availability and redundancy through self-healing and replication mechanisms but introduces these operational complexities that must be carefully managed.

Aydin et al. [6] offered valuable insights into the fault-tolerance mechanisms of both Kubernetes and SLURM systems. Their analysis highlighted a critical trade-off: Kubernetes demonstrated 40-60% faster recovery times for node failures through its automated mechanisms but consistently imposed higher operational overhead (10-15%) compared to SLURM's 5-10%. Their error distribution analysis revealed that SLURM faced challenges primarily with node errors and job initialization failures, while Kubernetes struggled most with networking issues and resource contention. SLURM operates closer to the hardware with lower overhead, making it more efficient for compute-intensive workloads. However, it required significantly more manual intervention for fault recovery, with an average of 7.4 administrator actions per major failure compared to Kubernetes' 2.1 actions. SLURM provides efficient resource management but faces scalability issues compared to Kubernetes' more flexible architecture.

The proposed solution bridges the gap between Kubernetes and SLURM by implementing SLURM on native VMs with Singularity support and Infrastructure as Code (IaC) deployment. This approach maintains SLURM's performance efficiency while addressing its scalability limitations. The IaC methodology automates configuration management, reducing the manual intervention that Aydin et al. [6] identified as a significant limitation in traditional SLURM deployments. This hybrid approach represents a promising direction for HPC container orchestration, combining SLURM's resource efficiency with improved automation and scalability features inspired by Kubernetes. By addressing the limitations of

both systems, this solution offers a balanced approach for modern scientific computing workloads that demand both performance and flexibility.

Lupión et al. [17] addressed accessibility challenges through S-TFManager, a lightweight open-source web manager that simplifies the training of TensorFlow neural network models in SLURM-based HPC environments. This tool offers researchers without extensive system administration expertise a user-friendly interface with built-in visualization and hyperparameter exploration capabilities, streamlining the management of multiple training jobs on shared computing clusters.

The reference architecture further enhances S-TFManager by integrating JupyterLab environments through Singularity containers directly within the SLURM scheduling system. This integration enables researchers to deploy customized and more robust development environments with their specific dependencies while leveraging SLURM's resource allocation and scheduling capabilities.

Conclusion

In this study, we demonstrated the effectiveness of deploying SLURM in a cloud environment to manage HPC workloads efficiently. The utilization of SLURM, automated through the Infrastructure as Code paradigm, enables scalable and reproducible resource allocation. To promote transparency and reproducibility, the source code developed for this work has been released as open-source software and is freely accessible[2]. Our results show significant performance improvements with the increased utilization of worker nodes, highlighting SLURM's capability to handle parallel computing tasks. Future work will focus on analysing the scaling limitations and overheads to further optimize performance.

The results shown in Figure 2. demonstrate that increasing the number of active worker nodes leads to a substantial reduction in the mean computational runtime for the benchmarked task under the SLURM scheduler. Specifically, scaling from a single node to eight nodes decreased the average execution time by approximately 82.1%, from 3154.75 seconds to 563.75 seconds.

Although the experiment results are promising, we observed that as the bottlenecks (I/O, network contention and workload imbalance) compound, the theoretical linear speedup promised by parallel computing gives way to sublinear scaling, where adding more nodes yields progressively smaller performance improvements, eventually reaching a point of diminishing returns. As emerging heterogeneous computing architectures, smart workload distribution algorithms, and specialized interconnect technologies push the boundaries of efficient parallelization, the analysis of these bottlenecks represent potent ground for future research.

[2] https://git.sztaki.hu/science-cloud/reference-architectures

In conclusion, we deployed a SLURM scheduler and workload manager platform in a cloud environment, using Infrastructure as Code. The utilization of such a system provides computational task execution benefits for scientific projects and communities. The observed scaling behaviour suggests that the workload is amenable to parallel processing up to at least 8 nodes, though with inherent overheads preventing ideal linear speedup. Further research might explore the specific sources of this overhead and assess automated scaling in relation to the utilization of computational cluster resources. The ongoing research in this field continues to drive innovation at the intersection of HPC and cloud-native technologies with each advancement bringing us closer to converged computing platforms that serve diverse scientific workloads efficiently. As container technologies and orchestration solutions evolve, we can expect further improvements in scalability, fault tolerance, and usability for containerized HPC environments.

As future work, the aim is to conduct a similar experiment on both a traditional HPC environment and a cloud computing platform for comparative analysis. This comparative study will assess MLOps integration capabilities across different infrastructures and evaluate the effectiveness of both container orchestration and traditional workload management systems in that regard. The analysis will focus on processing efficiency, application portability, and I/O performance penalties inherent to containerized workloads. During the measurement process, energy and cost analysis might also be assessed as it is always a critical parameter in the HPC and cloud hardware system analysis. This study will provide empirical evidence to guide optimal deployment strategies for scientific computing applications across heterogeneous computing environments.

Acknowledgement

The authors thankfully acknowledge the support of the Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University. On behalf of the "Modelling of orchestration methods with machine learning" project, we are grateful for the possibility of using HUN-REN Cloud (see https://science-cloud.hu/), which helped us achieve the results published in this paper. This work was partially funded by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program.

References

[1] M. Héder *et al.*, 'The Past, Present and Future of the ELKH Cloud', *Inf. Társad.*, Vol. 22, No. 2, p. 128, Aug. 2022, doi: 10.22503/inftars.XXII.2022.2.8

[2] I. Ullah, M. S. Khan, M. Amir, J. Kim, and S. M. Kim, 'LSTPD: Least Slack Time-Based Preemptive Deadline Constraint Scheduler for Hadoop Clusters', *IEEE Access*, Vol. 8, pp. 111751-111762, 2020, doi: 10.1109/ACCESS.2020.3002565

[3] '(PDF) A Novel Optimization Strategy for Job Scheduling based on Double Hierarchy', *ResearchGate*, doi: 10.25103/jestr.101.10

[4] Q. Li, W. Wu, X. Zhou, Z. Sun, and J. Huang, 'R-FirstFit: A Reservation Based First Fit Priority Job Scheduling Strategy and Its Application for Rendering', in *2014 IEEE 17th International Conference on Computational Science and Engineering*, Dec. 2014, pp. 1078-1085, doi: 10.1109/CSE.2014.213

[5] A. B. Yoo, M. A. Jette, and M. Grondona, 'SLURM: Simple Linux Utility for Resource Management', in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., Berlin, Heidelberg: Springer, 2003, pp. 44-60, doi: 10.1007/10968987_3

[6] '(PDF) Comparing Fault-tolerance in Kubernetes and Slurm in HPC Infrastructure', in *ResearchGate*, Jun. 2025, Accessed: Aug. 07, 2025, [Online] Available: https://www.researchgate.net/publication/384455176_Comparing_Fault-tolerance_in_Kubernetes_and_Slurm_in_HPC_Infrastructure

[7] B. Schmidt and A. Hildebrandt, 'Next-generation sequencing: big data meets high performance computing', *Drug Discov. Today*, Vol. 22, No. 4, pp. 712-717, Apr. 2017, doi: 10.1016/j.drudis.2017.01.014

[8] B. Van Essen, H. Kim, R. Pearce, K. Boakye, and B. Chen, 'LBANN: livermore big artificial neural network HPC toolkit', in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, in MLHPC '15. New York, NY, USA: Association for Computing Machinery, 0 2015, pp. 1-6, doi: 10.1145/2834892.2834897

[9] J. Doi, H. Takahashi, R. Raymond, T. Imamichi, and H. Horii, 'Quantum computing simulator on a heterogenous HPC system', in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, in CF '19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 85-93, doi: 10.1145/3310273.3323053

[10] E. Roloff, M. Diener, A. Carissimi, and P. O. A. Navaux, 'High Performance Computing in the cloud: Deployment, performance and cost efficiency', in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, Dec. 2012, pp. 371-378, doi: 10.1109/CloudCom.2012.6427549

[11] J. Kovács, B. Ligetfalvi, and R. Lovas, 'Automated Debugging Mechanisms for Orchestrated Cloud Infrastructures with Active Control and Global Evaluation', *IEEE Access*, Vol. 12, pp. 143193-143214, 2024, doi: 10.1109/ACCESS.2024.3467228

[12] G. M. Kurtzer, V. Sochat, and M. W. Bauer, 'Singularity: Scientific containers for mobility of compute', *PLOS ONE*, Vol. 12, No. 5, p. e0177459, May 2017, doi: 10.1371/journal.pone.0177459

[13] G. Hu, Y. Zhang, and W. Chen, 'Exploring the Performance of Singularity for High Performance Computing Scenarios', in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 2587-2593, doi: 10.1109/HPCC/SmartCity/DSS.2019.00362

[14] 'UniProt', UniProt. Accessed: Aug. 07, 2025 [Online] Available: https://www.uniprot.org/proteomes/UP000005640

[15] J. Jumper *et al.*, 'Highly accurate protein structure prediction with AlphaFold', *Nature*, Vol. 596, No. 7873, pp. 583-589, Aug. 2021, doi: 10.1038/s41586-021-03819-2

[16] D. J. Milroy *et al.*, 'One Step Closer to Converged Computing: Achieving Scalability with Cloud-Native HPC', in *2022 IEEE/ACM 4th International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, Nov. 2022, pp. 57-70, doi: 10.1109/CANOPIE-HPC56864.2022.00011

[17] M. Lupión, N. C. Cruz, F. Romero, J. F. Sanjuan, and P. M. Ortigosa, 'A Lightweight Execution Manager for Training TensorFlow Models under the Slurm Queuing System', *Acta Polytech. Hung.*, Vol. 22, No. 3, pp. 63-78, 2025, doi: 10.12700/APH.22.3.2025.3.4