Enhancing Supercomputing Education through a Low-Cost Cluster: A Case Study at Insper

Lícia S. C. Lima[1], Tiago A O Demay[2], Leonardo N Rosa[3], André Filipe M. Batista[4], Luciano Silva[5]
dept. of Computer Engineering.[1,2,4,5], dept. Fab.Lab.[3]
Emails: (liciascl[1], tiagoaodc[2], leonardonr[3], andrefmb[4], lucianos4[5])@insper.edu.br
Insper, São Paulo, Brazil.

***Abstract*—High-Performance Computing (HPC) and parallel programming presents intricate challenges due to the sophisticated interplay between advanced hardware and software components. This paper delineates a case study of a cost-effective cluster comprising 24 Upboards engineered to bolster a project-based Supercomputing course. The project received the name of UpCluster, and it serves as a pragmatic, cost-efficient solution for experiential learning, mitigating the abstraction often associated with theoretical constructs. The curriculum encompasses various topics, including distributed computing, parallel computing, algorithm analysis, and the Message Passing Interface (MPI). The team meticulously documented the cluster infrastructure, providing a comprehensive guide for the configuration and utilization of the Single Board Computer cluster with Kubernetes and MPI operators. Students engaged in practical experimentation, developing scalable algorithms, and gaining valuable insights into the challenges and opportunities associated with distributed computing. These experiences fostered a deeper appreciation for the complexities and potential of distributed computing. The primary objective of this study is to demonstrate the efficacy of the cost-effective cluster in augmenting high-performance computing education. By providing a practical learning environment, the UpCluster complements theoretical instruction and empowers students to acquire practical skills in the design of large-scale distributed systems with multi-core nodes. Furthermore, the paper discusses this low-cost cluster's potential impact and applications in HPC education. The insights from the study may benefit academic departments and institutions seeking to develop analogous project-based courses focused on high-performance computing for graduate students.**

I. Introduction

HPC has emerged as an indispensable instrument for propelling research and innovation, spanning from academia to industry. These sectors utilize HPC in both scientific and engineering computing as seen in the work of Nengzhi et al. [1]. The pedagogical process of HPC and parallel programming presents challenges, primarily due to the complex interaction between advanced hardware and software components. This situation highlights the growing need for educational methods that provide students with hands-on experience in designing, developing, and operating HPC systems.

HPC is centered around addressing computational problems that surpass the capabilities of standard desktop computing. To educate graduate students in this domain, it's essential to effectively convey the principles of parallel computing. These principles enable the resolution of intricate issues using machine clusters as in the work of Shamsi et al. [2] and Weems et al. [3].

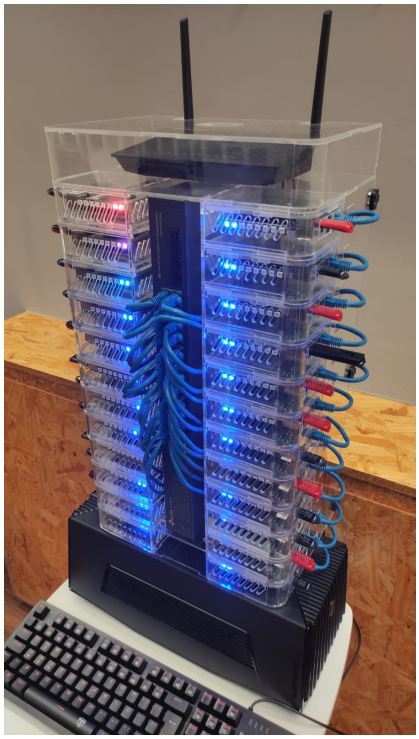

Fig. 1. UP Cluster at Insper. shows the operational Up cluster in the networking and HPC laboratory. Documentation available at https://insper.github.io/UpCluster/

In light of these needs, our study introduces a cost-effective cluster tailored specifically for a project-centric Supercomputing course. As depicted in Figure 1, this cluster, consisting of 24 Upboards, offers an affordable solution for experiential learning. It bridges the divide between theoretical knowledge and practical application. Our primary aim is to showcase how this cost-effective cluster can enhance high-performance computing education. By establishing a tangible learning environment, the cluster not only complements theoretical teachings but also equips students with hands-on skills in HPC system design and operation.

This paper offers a detailed overview of the cluster's infrastructure, the course curriculum, and the techniques used to promote hands-on learning. We also delve into the potential applications and impact of this budget-friendly cluster in HPC education. The findings from our research can aid academic

departments and institutions aiming to craft similar project-based HPC courses for graduate students.

The subsequent sections of this paper are organized as follows: Section 2 reviews related work in HPC education. Section 3 describes the cluster's hardware and software setup, emphasizing its cost-effectiveness and scalability. Section 4 details the Supercomputing course content. Section 5 showcases an accelerated Monte Carlo-based pi calculation using MPI on the affordable UpCluster. Section 6 evaluates the cluster's role in advancing HPC education, focusing on its practicality and influence on student learning. Section 7 concludes the paper, summarizing our study's contributions.

II. Related work in HPC education

HPC and MPI have been active research areas, with multiple implementations and evaluations discussed in the literature. The application of HPC in scientific advancement is essential for performing high-performance simulations, analyses, and modeling in fields such as physics, chemistry, biology, environmental sciences, and engineering as observed in the works [4]–[10]. The teaching of HPC holds promise due to its wide application and real-world relevance. HPC involves the use of advanced computing techniques to solve complex problems and perform computationally intensive calculations in a reasonable amount of time as in the articles [11]–[13].

This section divides the content into three main topics: Implementations and Evaluations of MPI and HPC, Parallel Programming Models and Tools, and Education and Training in HPC.

A. Implementations and Evaluations of MPI and HPC

Several works have explored different implementations of MPI and HPC and their performance evaluations. For instance, in work of Huang et al. [14] proposes a message-balanced forwarding system along with middleware-level throttling to reduce I/O variability in QoS-less HPC storage systems. They conducted a series of experiments to evaluate the effectiveness of their proposal and found that it could reduce I/O variability.

Similarly, the article of Li et al. [15] presents solutions for the efficient execution of programs written in the hybrid MPI/OpenMP programming model targeting large-scale distributed systems with multicore nodes. The proposed algorithm achieves substantial energy savings with either negligible performance loss or even performance gain. Among the contributions, the article proposes solutions to make parallel programs more energy-efficient in large-scale distributed systems with multicore nodes. The authors introduce a novel algorithm that can save significant energy while maintaining good performance. They also provide a study on energy-saving opportunities in strong and weak scaling of hybrid applications at unprecedented system scales of up to 1024 cores.

The significant power footprint of edge computing systems motivates holistic approaches to energy management that integrate hardware and software solutions.

Container utilization is increasingly prevalent in the cloud computing era due to its lightweight and portable nature, enabling efficient resource utilization and rapid application deployment, the work of Zhou et al. [10] discuss the use of containers in HPC environments. The article examines the utilization of containers in HPC environments. Containers encapsulate complex applications and their dependencies within isolated environments, enhancing compatibility and portability. The article emphasizes the distinctions between containerization and its orchestration strategies in HPC systems compared to cloud systems. They propose a novel container implementation approach that can reduce the performance overhead associated with traditional containers. They evaluated their approach on a computer cluster and found it could improve application performance.

B. Parallel Programming Models and Tools

Parallel programming models and tools, such as the MPI and OpenMP, are vital for effectively harnessing the computational power of modern systems. MPI facilitates communication and synchronization among distributed processes, while OpenMP simplifies shared memory parallelization through compiler directives and runtime libraries. These models, along with various tools and libraries, provide abstractions and frameworks that enable developers to express and exploit parallelism efficiently, enabling scalable and efficient parallelization of applications across distributed and shared memory architectures. Tools like Kubernetes are an advanced container orchestration platform that excels at managing clusters and distributed computing environments with its robust autoscaling, load balancing, and service discovery features. By abstracting away the underlying complexities of the infrastructure, it enables seamless scalability and distribution of workloads, improving resource utilization and fault tolerance. Overall, parallel programming models and tools are essential for harnessing the power of parallel computing systems. They enable developers to express parallelism effectively, abstract away system complexities, and optimize performance. The literature also explores several parallel programming models and tools. For example, Hui et al. [16] discusses the implementation of a microservices-based distributed cluster scheduler for Kubernetes called Epsilon.

In the article of Mart et al. [17], discussed the automatic detection of anomalies in Kubernetes clusters using Prometheus. Kubernetes is an open-source container management platform with features like automatic scaling, service discovery, load balancing, and fault tolerance. The authors propose a new solution to capture and predict anomalies earlier, leveraging the monitoring system's metrics. The concept of observability, originating from control theory, refers to the ability to infer the internal states of a system based on its external outputs. In distributed systems, observability is often composed of three main components: logs, metrics, and traces.

The work by Zhou et al. [10] proposes a floating-point (FP) processing element as the basic unit of accelerators necessary to meet the requirements of multiple precisions with energy-

efficient operations. They implemented their processing element on an FPGA chip and found it could improve application performance.

Additionally, the work of Mullen et al. [12] discusses the application of high-performance computing in radar modeling. They implemented a radar model on a computer cluster and found it could improve radar prediction accuracy.

The article conducted by Hasan et al. [18] introduces the Kube-Monitor framework, designed for real-time resource monitoring in heterogeneous Kubernetes clusters. This framework enables monitoring resources in a diverse range of devices, including traditional data center infrastructure and Raspberry Pi devices. Kube-Monitor collects and updates 15 different resource metrics at a five-second interval, providing Kubernetes operators with up-to-date and comprehensive insights into resource utilization. The article also emphasizes the importance of enhancing computer science and engineering undergraduate education by incorporating HPC courses into the curriculum. It discusses the implementation of MPI-enabled grids using widely adopted software systems such as MPICH and Globus. Furthermore, the article addresses the evolution of HPC courses and the challenges faced. It highlights the research achievements of students in HPC, providing valuable insights for academic institutions and educators in the field.

C. Education and Training in HPC

Education and training in HPC have been a focus area in the literature. The work by Mullen et al. [12] proposes using severe games in informal HPC courses to provide students with tangible, hands-on experiences with HPC concepts. They implemented their game in an HPC course and found that it could improve students' understanding of HPC concepts.

Similarly, the work by Catalán et al. [13] discusses the need to teach HPC to undergraduate students and proposes a non-mandatory course titled "Build Your Own Cluster Using Raspberry Pi" to provide students with HPC skills. They implemented their course at a university and found that it could improve students' understanding of HPC concepts.

The research conducted by Gschwandtner et al. [11] discusses the challenges of teaching HPC due to the complex hardware and software components involved. It explores the difficulties in understanding and managing different levels of the HPC hardware architecture hierarchy and highlights the various software tools available for interacting with these aspects. The article presents Cluster Coffer, a miniature cluster computer, as a solution to make HPC more accessible in education and public outreach. It provides insights into its design goals, implementation, and use cases. Finally, the article presents two use cases where Cluster Coffer has been used multiple times and will continue to be used in the coming years.

In the work of B. Neelima [19], the author address the successful introduction of HPC courses to undergraduate students at a private engineering institution in India. It highlights the institution's background, curriculum enhancements, infrastructure development, academic achievements, and challenges. The article is a reference for educators and students interested in implementing similar HPC courses based on the institution's best practices.

In this paper, we shall delve into the concepts of the articles mentioned above, explicitly emphasizing tackling the challenges associated with instructing HPC and Distributed Computing. However, our main contribution is to demonstrate how we successfully abstracted the complexity of these issues using an efficient and low-cost cluster we designed. This cluster has proven effective in enhancing classroom dynamics and providing students with hands-on experiences in HPC and distributed computing.

III. Cluster Characteristics

The team established the conception of the UpCluster with a specific challenge in mind. They aimed to enhance the student experience in HPC and parallel programming classes using a project-based learning method.

Additionally, there was a scenario where materials were readily available for the project's construction. These materials were initially acquired for other projects. However, they were eventually replaced with more up-to-date hardware. This situation allowed for the system's development at a minimal cost.

The UpCluster's quality underscores the feasibility of creating a functional and cost-effective computing environment by repurposing existing resources.

This section will describe the hardware, physical structure, and software used to build the UpCluster, along with the design choices made since the team intends to use this project in an educational environment to facilitate the student's understanding and bring physicality to the concepts of the subject. Details about the design, building, installation guides, configurations, and applications are available in the project documentation[1].

A. Hardware

The system consists of 24 UpBoards as shown in Figure 2, each equipped with an Intel Atom X5-Z8350 "Cherry Trail" quad-core processor running at 1.44GHz (with a burst frequency of 1.92GHz) and integrating Intel Gen8 HD graphics. The set of boards has 21 units used as compute nodes and three unities used as head nodes.

A Tp-Link Tl-SG3428 switch is responsible for the system data transmission interconnecting the nodes, and an Asus AC750 Dual Band router establishes the external connectivity. Each board holds one Pimoroni LED strip, used as visual feedback from the operations of the system, and a bootable USB drive for storing the operating system.

[1]More details can be found on the official course website at Github Insper, "Supercomp - high-performance computing course," GitHub, https://insper.github.io/supercomp/

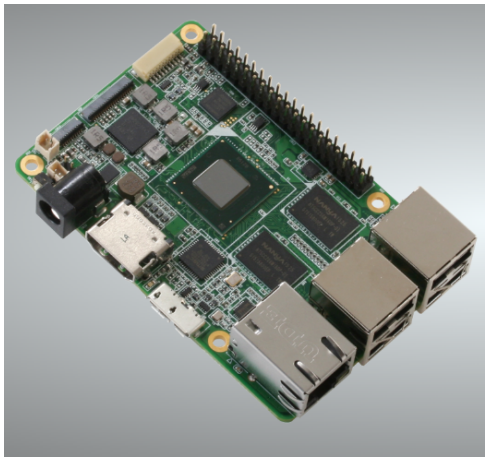

Fig. 2. UP Board Series is a credit-card-sized board with high performance and low power consumption features.

B. Physical Structure

The UpCluster structure consists of 3mm transparent acrylic sheets. The team obtained the material by reusing a protective barrier initially used in the COVID-19 pandemic in customer service areas. The material transparency allows for the observation of the internal assembly of the hardware.

The structure of the project has 24 drawers containing the UpBoards, designed with specific cutouts to enable access to connections for network cables, power cables, and bootable USB drives. The drawers' mechanism allows for isolated UpBoards repairs in the event of a defect, eliminating the need to disassemble the entire system. This design expedites maintenance processes while reducing the possibility of disruption to the system.

Two stacks of 12 drawers arrange the compartments, with a space between them that accommodates the switch device. A compartment is dedicated to house the router, positioned on the top of the structure. Finally, the bottom of the UpCluster holds the power supply of the system.

C. Software

The UpCluster uses Ubuntu 20.04.04 as an operating system (OS). The team chose this specific OS due to the compatibility with the hardware from Up, mainly for making available the use of the I/O ports from the board. The current versions of this distribution were not compatible with the hardware. The system also uses Kubernetes, an open-source platform that automates, scales, and manages containerized applications.

In order to ensure continuous service operation, even during server failures or network problems, the system uses KeepAlived and HAProxy. KeepAlived is an open-source implementation of high availability for network services that run in the operating system. The implementation allows the three head nodes of the cluster to share the same virtual IP address, establishing communication with each other to monitor the services they maintain. If the system detects a failure in a head node, another takes over the virtual IP address to maintain service continuity. HAProxy acts as a load balancer running in the Kubernetes environment. It will guarantee high availability when running critical workloads avoiding downtime, allowing scalability through workload distribution, and providing features like replication and fault tolerances ensuring that applications and services keep functioning even if some node or component fails. While KeepAlived maintains the availability of the head node, HAProxy will optimize the workload distribution and maintain the high availability of the applications.

To allow students in the course to visualize real-time activities on the UpCluster, we also implemented a monitoring stack using the open-source software Prometheus and Grafana, open-source tools for collecting, storing, and visualizing time-series data. Prometheus adopts a pull-based approach to periodically scrape metrics from target endpoints, while Grafana, as shown in 3, offers a user-friendly interface to create customizable dashboards and graphs. This combination empowers students to gain real-time insights into the cluster's performance and behavior, enhancing their understanding of distributed computing and practical challenges in managing high-performance systems.

Fig. 3. Global CPU usage, global Memory usage, and Kubernetes Resource count - Grafana graph showing cluster utilization over time.

IV. Project-based HPC course design

The course takes place over six months, with two classes per week, each lasting 2 hours, for a group of approximately 30 students from the 7th semester of the undergraduate computer engineering program. The course design prioritized a practical and hands-on approach to overcome the abstraction of theoretical concepts. The curriculum integrates theoretical understanding with real-world application. Evaluation includes practical exams and a multifaceted project covering different high-performance computing topics. The project provides

students with hands-on experience, reinforcing the practical application of learned concepts.

A. Course Details:

- **An introductory definition of HPC:** An overview of basic and historical concepts and how the high-performance computing area is today in the world.
- **C++ contextualization:** Addressing important language topics with hardware resources for program optimization, heuristics, and computationally complex problems.
- **Profiling:** In this module, students will learn about profiling techniques to analyze the performance of their programs. They will explore tools and methodologies to identify bottlenecks and optimize code efficiency, gaining insights into program behavior and resource utilization.
- **Heuristics:** This module introduces students to heuristic algorithms, which are problem-solving techniques that prioritize finding good solutions rather than guaranteed optimal solutions. Students will study heuristic algorithms, such as greedy algorithms and metaheuristics, and apply them to solve real-world computational problems.
- **Randomization:** Students will delve into the power of randomization in algorithm design. They will learn to use randomization techniques to enhance problem-solving approaches, including randomized algorithms, Monte Carlo simulations, and optimization methods.
- **Local and Exhaustive Search:** This module covers both local search and exhaustive search algorithms. Local search algorithms aim to improve solution quality by iteratively exploring neighboring solutions. On the other hand, an exhaustive search involves exploring all possible solutions to find the optimal solution. Students will study algorithms like brute-force search and backtracking, understanding their limitations and knowing when to apply them effectively.
- **Performance Comparison:** This module emphasizes the importance of performance comparison in evaluating and selecting appropriate algorithms or implementations. Students will learn techniques for benchmarking and analyzing the efficiency of different approaches, considering factors like runtime, memory usage, and scalability.
- **Parallelism and GPU Computing:** This module explores parallel computing concepts and GPU programming. Students learn the significance of parallelism in high-performance computing and acquire practical skills in parallel programming models, including shared memory (OpenMP) and distributed memory (MPI). They tackle side effects, synchronization, and load-balancing challenges in parallel applications. The module also introduces Graphics Processing Units (GPUs) and their role in accelerating computations. Students gain hands-on experience in GPU programming, optimizing memory access patterns, and leveraging data parallelism to process large datasets efficiently. Advanced techniques, such as memory management, thread synchronization, and kernel optimization, are covered to achieve optimal performance in GPU-based computations. This module equips students with the necessary skills to harness parallelism and GPU resources, enhancing computational speed and efficiency.
- **MPI (Message Passing Interface) in Distributed Computing:** The module begins with an introduction to MPI, a widely-used standard for interprocess communication in distributed computing. Students learn to develop distributed programs using MPI, focusing on message passing, process coordination, and scalability. The module integrates the UpCluster into the course to enhance practical learning. This cluster allows students to design, develop, and test their distributed programs in a real-world, commodity-based supercomputing system. By integrating the UpCluster and covering MPI introduction, reductions, and broadcasting, students acquire essential skills in developing distributed-memory distributed programs. They leverage MPI operations for efficient communication and data distribution in distributed computing environments. This practical approach equips them to tackle complex computational challenges and achieve optimal performance.

V. Accelerating Monte Carlo-based Pi Calculation Using MPI on a Low-Cost UpCluster

The activity involved the development of a distributed algorithm using MPI for calculating Pi based on the Monte Carlo method. Students began by implementing a sequential version of the algorithm, generating random points within a square and estimating Pi based on the number of points falling within the inscribed circle. This sequential implementation served as a reference for performance evaluation. To enhance the computational speed and explore the benefits of distributed computing, the MPI distributes the algorithm on the UpCluster. Figure 4 represents the calculation of Pi using the Monte Carlo approximation, which consists of a random sampling of points within a square of side 2(in blue). If the distance from the sampled point to the center of the square is less than or equal to 1, the point falls inside the inscribed circle (in red). The number of points falling inside the circle is proportional to Pi. By repeating this process with many points, the estimated value of Pi approaches the actual value. Simulations and numerical optimizations widely use the technique in science and engineering.

Students ventured into distributed computation using the cluster after initial implementation on individual machines. The workload was apportioned among multiple workers, each executing simulations with distinct point sets, enabling students to observe the impact of task distribution on the precision of the calculated Pi value. Each worker conducted a probabilistic calculation with N points, yielding distinct values post-N iterations. The mean of these outcomes across all workers provided a more accurate approximation of Pi than achievable with a single node.

Employment of the cluster enabled students to dispel the abstraction inherent in distributed computing, significantly augmenting comprehension of the practical task. They grappled

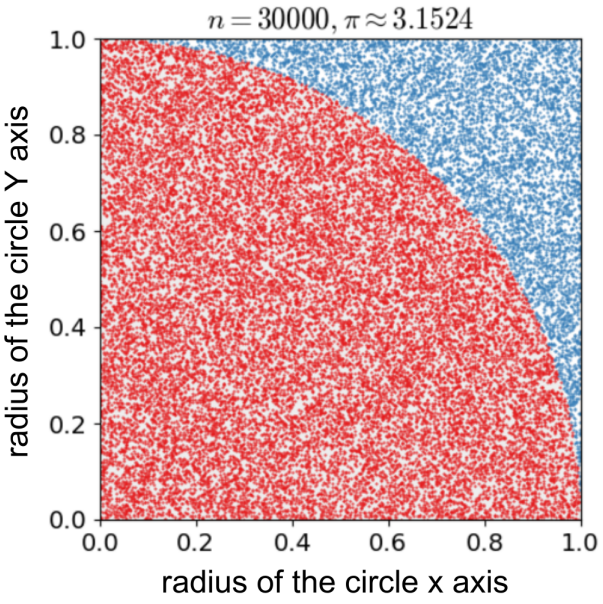

Fig. 4. Process of calculating the value of Pi using the Monte Carlo method. Random points are generated within a square (in blue), and the number of points falling within the inscribed circle (in red) is used to estimate the value of Pi.

N. of iterations	N. Works	network communication time	calculation time	value of the calculated pi constant	Error
500	8	0:01:00	0:00:20	3.11300000	-0.02859265
5000		0:01:00	0:00:40	3.12770000	-0.01389265
50000		0:01:00	0:01:00	3.14220000	0.00060735
500000		0:01:00	0:01:30	3.14241000	0.00081735
5000000		0:01:00	0:02:00	3.14162000	0.00002735
50000000		0:01:00	0:03:00	3.14148283	-0.00010982
500000000		0:01:00	0:15:00	3.14157951	-0.00001315
5000000000		0:01:00	0:22:00	3.14158514	-0.00000751
500	16	0:01:20	0:00:20	3.12600000	-0.01559265
5000		0:01:20	0:00:40	3.13295000	-0.00864265
50000		0:01:20	0:00:50	3.14401500	0.00242235
500000		0:01:20	0:01:00	3.14249900	0.00090635
5000000		0:01:20	0:01:00	3.14151120	-0.00008145
50000000		0:01:20	0:02:00	3.14148885	-0.00010380
500000000		0:01:20	0:15:00	3.14158812	-0.00000454
5000000000		0:01:20	0:22:00	3.14158913	-0.00000352
500	32	0:03:00	0:00:20	3.13400000	-0.00759265
5000		0:03:00	0:00:30	3.13220000	-0.00939265
50000		0:03:00	0:00::40	3.14312000	0.00152735
500000		0:03:00	0:00:50	3.14195550	0.00036285
5000000		0:03:00	0:01:40	3.14157120	-0.00002145
50000000		0:03:00	0:02:00	3.14155814	-0.00003451
500000000		0:03:00	0:15:00	3.14158924	-0.00000342
5000000000		0:03:00	0:22:00	3.14159475	0.00000210

Fig. 5. Results of Monte Carlo-based Pi Calculation Using MPI on a Low-Cost UpCluster

with challenges associated with efficient communication and synchronization among workers. Reflecting upon the outcomes of diverse simulations with varying numbers of workers and point sets, students gained insights into the trade-offs between computational precision and performance. This practical experience facilitated a deeper appreciation of the complexities and potentialities inherent in distributed computing.

Figure 5 presents the simulation results of 20 runs for each configuration. The values presented in the table correspond to the average of these runs. The columns of the table represent the number of iterations used for Pi calculation, the number of allocated workers for performing the calculation, the time taken for the network to stabilize and allow communication between the main MPI node and each worker, the total time for the calculation to complete across all nodes and the result to be presented by the main node, the calculated value of the Pi constant, and the absolute error with the ideal value of Pi. The students conducted insightful analyses during the study, identifying relevant findings regarding the performance and efficiency of distributed Pi calculation. The students identified that the additional overhead of network communication time did not justify task distribution for small values of N, considering only the time and the marginal difference in the obtained results. Furthermore, the wide availability of observability and monitoring software, as shown in Figure 6, for clusters had a detrimental impact on network communication, memory space, and CPU utilization for Pi calculation. Additional analysis revealed that implementing a hybrid program combining OpenMP and MPI could significantly reduce the calculation time by leveraging the ability to divide and parallelize the task simultaneously, as shown by Rabenseifner [20]. These student analyses provided valuable insights and contributed to a comprehensive understanding of the challenges and opportunities associated with distributed Pi calculation.

As depicted in Figure 6, the most significant memory resource allocation of UpCluster is dedicated to its operation. Specifically the Kube-system namespace, consuming approximately 5% of the total cluster capacity. Additionally, the Grafana and Prometheus namespaces account consumes 2.5%. In contrast, the memory utilized for Pi calculation constitutes less than 1% of the cluster's memory potential. This observation highlights the efficient allocation of resources, ensuring that the Pi calculation task does not overly burden the overall cluster performance, leaving substantial memory headroom for other critical components.

This practical approach allowed students to engage in practical experimentation actively, developing scalable algorithms and acquiring a deeper understanding of the benefits and challenges of utilizing distributed computing resources. Through hands-on experience and reflective analysis, they honed their skills in analyzing and interpreting results, enabling informed decisions about algorithm design and resource allocation. These experiences fostered a deeper appreciation for the complexities and potential of distributed computing, equipping students for future challenges and opportunities in this field.

VI. Results

Project-Based Learning (PBL) is an educational approach that promotes active and hands-on learning through real-world

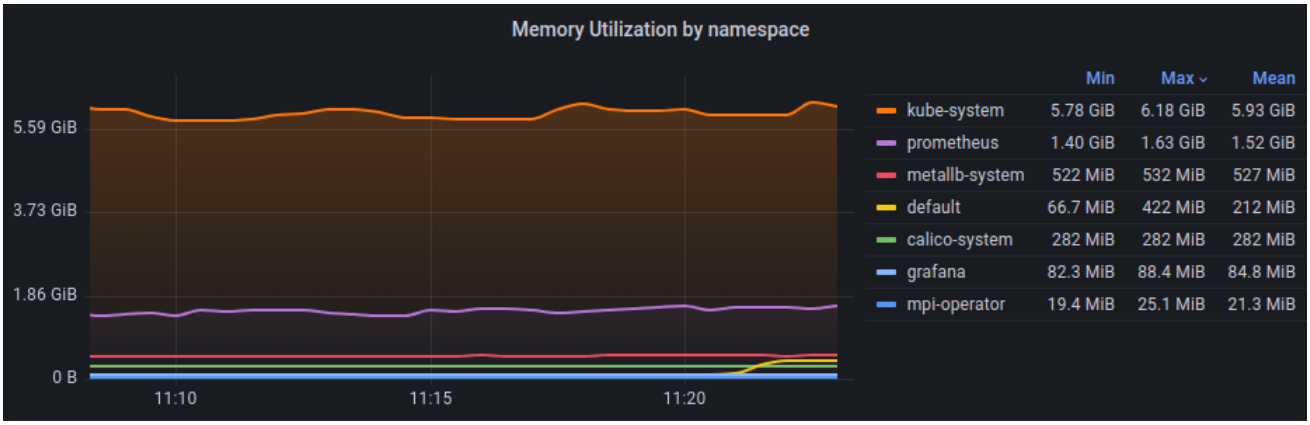

Fig. 6. Memory utilization by namespace - Grafana graph showing cluster utilization over time.

projects. It encourages students to actively engage in problem-solving, critical thinking, collaboration, and communication skills. In HPC education, using the UpCluster as a practical tool has proven highly effective in enhancing the learning experience. The cluster's practicality and impact on student learning are evident in several ways.

Firstly, the hands-on nature of working with the cluster allows students to apply theoretical concepts in a real-world context directly. By designing, developing, and testing commodity-based supercomputing systems on the cluster, students gain practical experience in HPC system architecture, deployment, and operation. Additionally, the UpCluster is an accessible and cost-effective platform for student experimentation and exploration. It provides a tangible and realistic environment for students to explore the complexities of distributed computing, MPI programming, and optimization techniques. Through active engagement with the cluster, students understand the challenges and trade-offs of utilizing distributed computing resources effectively.

Figure 7 illustrates how each namespace utilizes memory resources in UpCluster. The machine with IP 192.168.50.102, represented by the yellow curve with a peak consumption of 2.05GB, allocates the infrastructure service of the Kubernetes cluster. The graph displays the applications responsible for solving the Pi problem at the bottom, with a maximum memory consumption of 875MB. These data indicate that the cluster's operating infrastructure consumes more memory resources than the distributed application. Globally, the "kube-system" namespace accounts for 5% of the total Upcluster capacity, whereas the application amounts to less than 1%. Regarding CPU consumption by the application responsible for the Pi problem, the average is 22%, with a maximum consumption of 40% for workers and 50% for the leader. As for the "kube-system" namespace, the peak CPU consumption reached 71%, with an average of 44%. Thus, the UpCluster's operating infrastructure has a more significant impact on the system than the distributed application. These results present students with a scenario where it is crucial to carefully evaluate when it is genuinely appropriate to use distributed computing resources. The problem must be computationally complex to justify the trade-off.

Furthermore, the cluster's use in HPC education fosters collaborative learning and teamwork. Students have the opportunity to work together, sharing knowledge and insights as they tackle complex computational challenges. Collaborative projects on the cluster promote effective communication and problem-solving skills, preparing students for real-world scenarios where teamwork and collaboration are essential. According to research conducted by Chaudhury [21], that introduces a web-based platform named as Let's HPC. The platform serves as a valuable resource for enhancing pedagogical practices in the areas of High-Performance Computing (HPC) and Parallel and Distributed Computing (PDC). Let's HPC allows students to gain a comprehensive understanding of parallel algorithms and provides opportunities for assessment, instruction, and system-level analysis. By harnessing the dynamic capabilities of the platform, educators and students can achieve a more profound comprehension of HPC and PDC concepts with greater ease and efficiency. This study corroborates with the results we analyzed from our work, showing that students acquire practical skills by engaging with real-world scenarios.

The practicality and impact of the UpCluster in HPC education extend beyond the classroom. By gaining hands-on experience with a low-cost cluster, students develop valuable, highly relevant skills in industry and research settings. The ability to design, deploy, and optimize HPC systems using commodity hardware and open-source software becomes a valuable asset for students pursuing careers in scientific research, engineering, and data-intensive fields. Using the UpCluster as a practical tool in HPC education demonstrates the effectiveness of project-based learning. It enables students to bridge the gap between theoretical knowledge and real-world application, enhancing their understanding of parallel computing concepts and techniques. The hands-on experience with the cluster cultivates critical thinking, problem-solving, collaboration, and communication skills, preparing students for the challenges and opportunities in high-performance com-

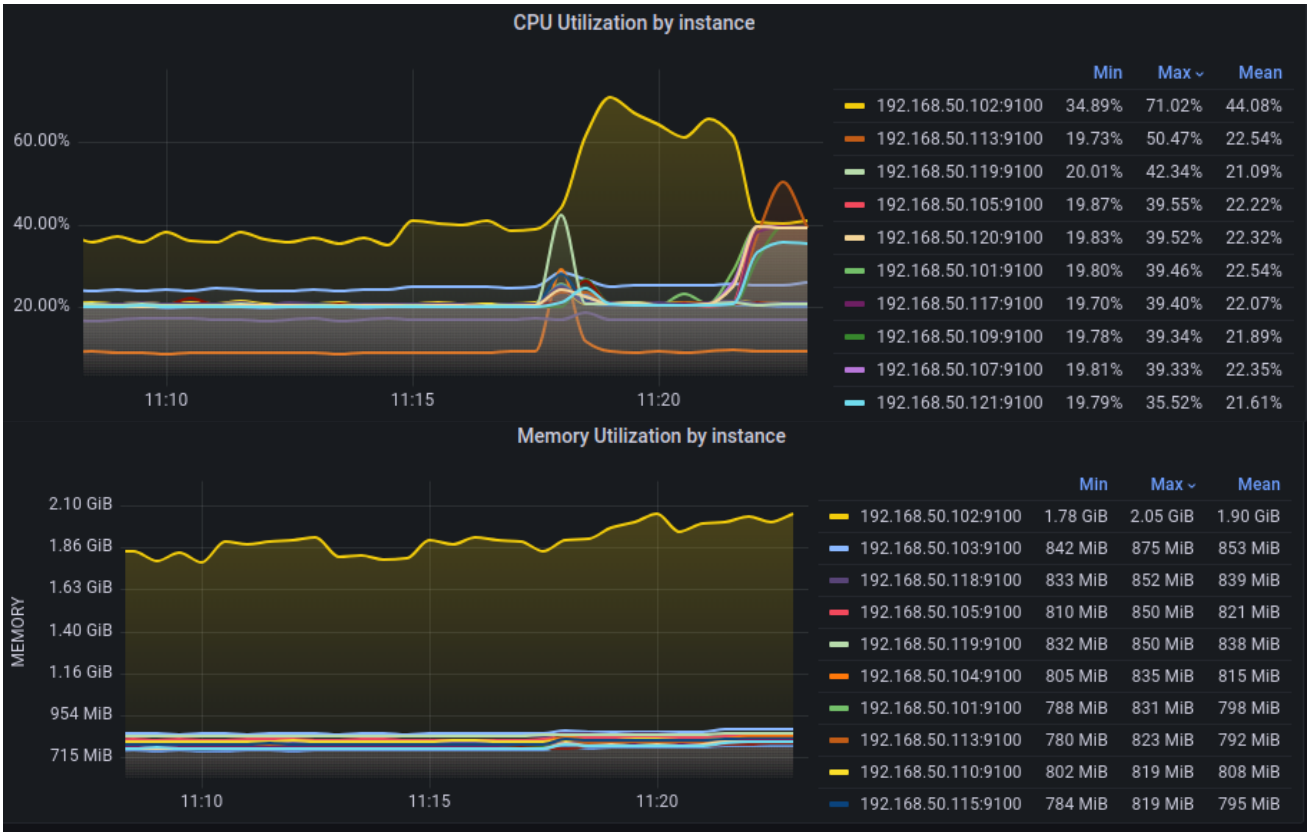

Fig. 7. CPU usage by instance, Memory usage by instance, and Memory utilization by namespace - Grafana graph showing cluster utilization over time.

puting.

VII. Conclusion

This study highlights the significant contributions of incorporating the UpCluster as a practical tool in HPC education. By providing students with hands-on experience, the cluster enhances their understanding of HPC concepts and techniques. The practicality and impact of the cluster in student learning are in the practical application of parallel, MPI programming, and optimization techniques. The key contributions of this study include:

- **Practical Learning Experience:** The UpCluster allows students to apply theoretical concepts in a real-world context directly. They gain practical skills in HPC system design, deployment, and operation, preparing them for real-world applications in industry and research.
- **Bridging Theory and Practice:** The cluster is a tangible platform that bridges the gap between abstract HPC concepts and their practical implementation. Students develop a deeper understanding of the challenges and trade-offs involved in utilizing distributed computing resources effectively.
- **Collaboration and Communication Skills:** Collaborative projects on the cluster promote effective teamwork, communication, and problem-solving skills. Students learn to work together, share knowledge, and tackle complex computational challenges, mirroring real-world scenarios.
- **Cost-Effective Solution:** The UpCluster provides a cost-effective alternative to traditional supercomputing systems. Its accessibility allows more students to engage in hands-on HPC experiences, providing a practical approach to HPC education.
- **Next Steps:** Establish a queuing system (CI/CD) that empowers students to develop code, rigorously test algorithm performance through automated validation, and seamlessly submit their code to the cluster via the integrated queuing system. Provide the cluster for scientific and technological initiation activities and as a computational asset for academic research in the field of Astroinformatics, spanning areas such as exoplanet classification, neutron stars, and supernovae analysis, and create analogous cluster setups with alternative hardware configurations to facilitate comparative performance assessments.

Incorporating the UpCluster as a practical tool in HPC education has significantly contributed to practical learning experiences, bridging theory and practice, fostering collaboration and communication skills, and providing a cost-effective solution. It not only enhances the educational experience but also has the potential to further expand its impact by enabling students to work on scientific and technological initiation activities and academic research in the field of

Astroinformatics. Additionally, looking ahead, there is the aspiration to create analogous cluster setups with alternative hardware configurations to facilitate comparative performance assessments.

REFERENCES

[1] J. Nengzhi, Z. Jianwu, X. Haili, W. Xiaoning, and S. Yulin, "Comparative research on high-speed networks of high performance computing cluster based on mpigraph," pp. 579–583, 2020.

[2] J. A. Shamsi, N. M. Durrani, and N. Kafi, "Novelties in teaching high performance computing," pp. 772–778, 2015.

[3] C. Weems, "Eduhipc 2021 keynote talk early parallel and distributed computing education: Canary in the coal mine," pp. 3–3, 2021.

[4] S. Diesburg, P. Gray, and D. Joiner, "High performance computing environments without the fuss: the bootable cluster cd," pp. 8 pp.–, 2005.

[5] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster," 2005.

[6] N. Regola and J. C. Ducom, "Recommendations for virtualization technologies in high performance computing," 2010, pp. 409–416.

[7] W. Mao, K. Li, Q. Cheng, L. Dai, B. Li, X. Xie, H. Li, L. Lin, and H. Yu, "A configurable floating-point multiple-precision processing element for hpc and ai converged computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, pp. 213–226, 2 2022.

[8] C.-C. Chen, K. S. Khorassani, G. K. R. Kuncham, R. Vaidya, M. Abduljabbar, A. Shafi, H. Subramoni, and D. K. Panda, "Implementing and optimizing a gpu-aware mpi library for intel gpus: Early experiences." IEEE, 5 2023, pp. 131–140. [Online]. Available: https://ieeexplore.ieee.org/document/10171511/

[9] M. S. Beni, L. Crisci, and B. Cosenza, "Empi: Enhanced message passing interface in modern c++." IEEE, 5 2023, pp. 141–153. [Online]. Available: https://ieeexplore.ieee.org/document/10171546/

[10] N. Zhou, H. Zhou, and D. Hoppe, "Containerization for high performance computing systems: Survey and prospects," *IEEE Transactions on Software Engineering*, vol. 49, pp. 2722–2740, 4 2023.

[11] P. Gschwandtner, A. Hirsch, P. Thoman, P. Zangerl, H. Jordan, and T. Fahringer, "The cluster coffer: Teaching hpc on the road," *Journal of Parallel and Distributed Computing*, vol. 155, pp. 50–62, 9 2021.

[12] J. Mullen, L. Milechin, and D. Milechin, "Teaching and learning hpc through serious games," *Journal of Parallel and Distributed Computing*, vol. 158, pp. 115–125, 12 2021.

[13] S. Catalán, R. Carratalá-Sáez, and S. Iserte, "Leveraging teaching on demand: Approaching hpc to undergrads," *Journal of Parallel and Distributed Computing*, vol. 156, pp. 148–162, 10 2021.

[14] D. Huang, Q. Liu, J. Choi, N. Podhorszki, S. Klasky, J. Logan, G. Ostrouchov, X. He, and M. Wolf, "Can i/o variability be reduced on qos-less hpc storage systems?" *IEEE Transactions on Computers*, vol. 68, pp. 631–645, 5 2019.

[15] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, "Hybrid mpi/openmp power-aware computing," pp. 1–12, 2010.

[16] A. N. Jing Hui and B. S. Lee, "Epsilon: A microservices based distributed scheduler for kubernetes cluster," pp. 1–6, 2021.

[17] O. Mart, C. Negru, F. Pop, and A. Castiglione, "Observability in kubernetes cluster: Automatic anomalies detection using prometheus." Institute of Electrical and Electronics Engineers Inc., 12 2020, pp. 565–570.

[18] B. T. Hasan and D. B. Abdullah, "Real-time resource monitoring framework in a heterogeneous kubernetes cluster." Institute of Electrical and Electronics Engineers Inc., 2022, pp. 184–189.

[19] B. Neelima, "High performance computing education in an indian engineering institute," *Journal of Parallel and Distributed Computing*, vol. 105, pp. 73–82, 7 2017.

[20] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes parallel, distributed and network-based processing," 2009.

[21] B. Chaudhury, A. Varma, Y. Keswani, Y. Bhatnagar, and S. Parikh, "Let's hpc: A web-based platform to aid parallel, distributed and high performance computing education," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 213–232, 8 2018.