future internet

Article

CommC: A Multi-Purpose COMModity Hardware Cluster

Agorakis Bompotas *, Nikitas-Rigas Kalogeropoulos and Christos Makris *

Department of Computer Engineering and Informatics, School of Engineering, University of Patras, 26504 Rio, Greece; kalogeropo@ceid.upatras.gr

* Correspondence: mpompotas@ceid.upatras.gr (A.B.); makri@ceid.upatras.gr (C.M.)

Abstract: The high costs of acquiring and maintaining high-performance computing (HPC) resources pose significant barriers for medium-sized enterprises and educational institutions, often forcing them to rely on expensive cloud-based solutions with recurring costs. This paper introduces CommC, a multi-purpose commodity hardware cluster designed to reduce operational expenses and extend hardware lifespan by repurposing underutilized computing resources. By integrating virtualization (KVM and Proxmox) and containerization (Kubernetes and Docker), CommC creates a scalable, secure, and cost-efficient computing environment. The proposed system enables seamless resource sharing, ensuring high availability and fault tolerance for both containerized and virtualized workloads. To demonstrate its versatility, we deploy big data engines like Apache Spark alongside traditional web services, showcasing CommC's ability to support diverse workloads efficiently. Our cost analysis reveals that CommC reduces computing expenses by up to 77.93% compared to cloud-based alternatives while also mitigating e-waste accumulation by extending the lifespan of existing hardware. This significantly improves environmental sustainability compared to cloud providers, where frequent hardware turnover contributes to rising carbon emissions. This research contributes to the fields of cloud computing, resource management, and sustainable IT infrastructure by providing a replicable, adaptable, and financially viable alternative to traditional cloud-based solutions. Future work will focus on automating resource allocation, enhancing real-time monitoring, and integrating advanced security mechanisms to further optimize performance and usability.

Keywords: cloud computing; big data; commodity hardware; cost-efficient processing; Apache Spark; Kubernetes; proxmoxve; environmentally conscious infrastructure

Academic Editors: Antonis Litke, Rodger Lea and Takuro Yonezawa

Received: 5 February 2025
Revised: 4 March 2025
Accepted: 7 March 2025
Published: 11 March 2025

Citation: Bompotas, A.; Kalogeropoulos, N.-R.; Makris, C. CommC: A Multi-Purpose COMModity Hardware Cluster. *Future Internet* **2025**, *17*, 121. https://doi.org/10.3390/fi17030121

1. Introduction

The cost of owning and maintaining high-end supercomputers capable of dealing with large-scale problems is usually too high for medium-sized companies or educational institutions. This problem is addressed by modern big data engines such as Apache Spark [1] that are designed to run on commodity hardware, which is efficiently combined to create computing clusters. The common practice is to lease the hardware that is necessary for executing analyses in the form of virtual machines that reside in the cloud (IaaS). In this way, an organization can greatly reduce its running costs, as it pays for hardware only when it uses it. However, the majority of organizations still have to purchase and maintain personal computers for their employees. This hardware remains largely underutilized as it is only used during working hours, and even when it is in use, most of the available resources (CPUs, RAM, etc.) are in an idle state. The work of Abdulghafour Mohammad and Yasir Abbas [2] identifies 11 major barriers that hinder small and medium enterprises from effectively utilizing cloud computing resources.

Such barriers can be categorized into technological, organizational, and environmental barriers. Technological issues include security and privacy concerns, integration difficulties with existing infrastructure, and unreliable cloud service performance. Organizational challenges stem from limited IT expertise, financial constraints, and resistance to change, which hinder smooth adoption. Environmental barriers involve regulatory compliance, vendor lock-in, market pressure for rapid adoption, unreliable internet connectivity, and lack of trust in service providers. To overcome these obstacles, organizations must adopt strategic planning, invest in training, and carefully select cloud providers to maximize the benefits of cloud computing.

Moreover, the current trend of research in this area is concentrated on high-performance systems, focusing on optimizing resource allocation, minimizing latency, and improving scalability in dynamic and heterogeneous environments. Advanced frameworks like EVRM [3] leverage cutting-edge technologies such as deep reinforcement learning to manage multiple virtual resources, ensuring efficient utilization of CPUs, memory, and bandwidth while adapting to fluctuating workloads. These innovations aim to address the limitations of traditional resource management approaches, enhancing system performance and reliability, especially in the context of virtualization and cloud computing.

The Elastic Virtual Resource Management (EVRM) framework, which is the work of Wang et al. (2025) [3], incorporates four modules—monitor, analyzer, planner, and executor—to dynamically manage virtual instances. The framework employs a deep reinforcement learning-based resource allocation method, Deep Deterministic Policy Gradient-based Resource Allocation (DDPG-RA), to optimize the allocation of CPUs, memory, and bandwidth. This approach learns the complex dependencies between resources and application performance, enabling better adaptation to fluctuating demands.

The demand for high-performance computing has increased dramatically, fuelled by advancements in big data analytics, artificial intelligence, and other computation-intensive workloads [4]. However, acquiring and maintaining specialized hardware, such as supercomputers, remains financially challenging for many medium-sized organizations and educational institutions [5]. This limitation has led to a growing interest in utilizing commodity hardware and virtualization technologies to achieve scalable and cost-effective computing solutions.

We developed a system that integrates cutting-edge open-source virtualization and cluster management technologies to address this need. Our approach involves deploying a hypervisor stack, virtualization management software, and a desktop environment on standard personal computers. This setup enables the creation of virtual machines (VMs) on each physical host, which users can seamlessly access through a VNC-based application. This ensures that users can perform their daily tasks without realizing they are interacting with a virtualized system rather than one running on physical hardware. This configuration also empowers system administrators with precise control over resource allocation for VMs on each host, allowing them to run additional tasks on the same machines.

The use of virtual machines instead of physical machines provides significant security enhancements by isolating workloads effectively [6]. Each VM operates within its own sandboxed environment, ensuring that a compromise in one VM does not impact others or the host system. This isolation reduces the risk of malware propagation or unauthorized access. Additionally, VMs enable secure snapshots and rollbacks, allowing administrators to quickly restore systems to a known safe state in the event of a security breach. Virtualized environments also support advanced monitoring and access control mechanisms, simplifying real-time threat detection and response. Features like virtual network segmentation and controlled resource sharing further enhance the security and manageability of the infrastructure compared to traditional physical systems.

For enhanced flexibility and easier maintenance, the physical machines are interconnected to form a cluster. This cluster setup provides numerous benefits for managing virtualized environments efficiently. By pooling resources such as CPU, RAM, and storage across multiple nodes, the cluster enhances scalability and ensures optimal utilization [7]. It facilitates seamless VM migration between nodes with minimal downtime, ensuring high availability and robust disaster recovery capabilities to maintain service continuity even during hardware failures [8]. Centralized cluster management simplifies administration by offering a unified interface to monitor and control the entire infrastructure. Moreover, shared storage solutions improve data redundancy and accessibility. With integrated tools for backup, restoration, and resource allocation, clustering enables organizations to maintain highly efficient, flexible, and reliable virtualized infrastructures.

This cluster, as mentioned before, can also be exploited to run big data engines like Apache Spark. However, this cannot be achieved without a specialized cluster manager for these kinds of jobs. Our solution is to host (in the same physical machines) some additional virtual machines that form a Kubernetes [9] cluster. Using Apache Spark with Kubernetes (K8s) offers powerful advantages for managing big data workloads. Kubernetes provides a flexible, scalable, and containerized environment that simplifies deploying and orchestrating Spark applications. Unlike Spark's Standalone mode, which lacks the breadth of ecosystem integrations and advanced scheduling features, Kubernetes provides fine-grained control over resource allocation and integrates natively with DevOps workflows, CI/CD pipelines, and monitoring tools. By leveraging Kubernetes' built-in resource management, Spark clusters can scale dynamically to handle varying workloads efficiently. The integration allows seamless containerization, ensuring consistency across development, testing, and production environments. Additionally, Kubernetes enhances fault tolerance by automatically rescheduling Spark executors in case of failures. Its native support for multi-cloud and on-premises environments ensures flexibility in infrastructure choices, while unified logging and monitoring tools streamline performance tracking and debugging. This container-centric approach reduces overhead, ensures portability, and simplifies management compared to provisioning and maintaining dedicated Spark Standalone clusters. Together, K8s and Spark enable robust, scalable, and efficient big data processing pipelines.

Compared to Mesos or YARN, Kubernetes benefits from a wider adoption and more active development community, which translates into better support, frequent updates, and rich plugins for networking, storage, and security. While YARN is tailored to Hadoop ecosystems and Mesos is known for generalized cluster management, Kubernetes embraces a broader cloud-native philosophy that lets Spark coexist seamlessly with other containerized workloads, ultimately delivering greater flexibility and efficiency. In CommC's case, this setup allows for running a series of services in the same Kubernetes environment. Running web services on Kubernetes enhances scalability, reliability, and operational efficiency by automating scaling to handle traffic fluctuations, ensuring consistent performance, and optimizing resource use. Its self-healing capabilities replace failed containers and restart unresponsive services, boosting uptime. Kubernetes simplifies deployments with declarative configurations, enabling seamless updates and rollbacks while minimizing disruptions. Moreover, Kubernetes natively provides both Layer 4 (TCP/UDP) load-balancing through its Service abstraction (e.g., ClusterIP, NodePort, and LoadBalancer Services) and Layer 7 (HTTP/HTTPS) load-balancing via the Ingress resource, which ensures even traffic distribution for better performance and resilience. Its platform-agnostic design supports deployments across on-premises, cloud, or hybrid environments, avoiding vendor lock-in. Robust networking features, including service discovery and secure communication, combined with integrated monitoring and logging, make Kubernetes an ideal platform for deploying, managing, and scaling modern web applications.

Nonetheless, the architecture that was described in the previous paragraphs comes with some drawbacks, too. Some jobs are inherently difficult to parallelize, and the machines used in this scenario are very limited, so it is of great importance to extract every bit of available performance without sacrificing the ease of managing the system. Our proposed solution for serial performance-oriented tasks is to run Linux containers alongside the full virtual machines. This enables us to have a high-level production orchestration system for managing our cluster and, at the same time, maintain full control over the resources allocated by the nodes participating in it.

However, apart from the technological advantages, this approach challenges environmental issues. It is common knowledge that the rapid growth of electronic waste (E-waste) [10] has been noted as an issue by various resources and organizations. The work of Jain et al. (2023) [10] highlights that E-waste generated by the disposal of electronics like laptops, mobile phones, and televisions contains hazardous materials posing significant risks to human health and the environment. The study explores global and Indian E-waste production trends, identifying factors such as industrialization, technological advancements, and poor recycling practices as key contributors. E-waste arises not only from the end of a product's lifespan but also from consumer behavior and technological obsolescence. Despite its rapid growth (3–4% annually), only 15% of E-waste is recycled, highlighting a significant environmental and health concern due to toxic substances [11]. Our proposed system architecture expands the lifespan of systems deemed obsolete and will eventually deteriorate the E-waste issue that our planet suffers from.

Our contribution: The CommC system introduces a novel approach to leveraging commodity hardware for cost-effective, scalable, and multi-purpose computing. Unlike traditional cloud-based or high-performance computing (HPC) cluster solutions, CommC integrates virtualization (Proxmox VE) and container orchestration (Kubernetes and Docker) within the same physical infrastructure, optimizing resource utilization while maintaining workload isolation. This hybrid model allows organizations to repurpose underutilized hardware, providing a low-cost alternative to expensive cloud services while retaining flexibility.

One of the core contributions of CommC is its cost-efficiency. By utilizing open-source technologies, CommC offers a self-managed cloud-like environment without incurring vendor lock-in or recurring subscription fees. Our benchmarking results demonstrate significant cost reductions compared to cloud providers such as Google Cloud, AWS, and Microsoft Azure, making it an ideal solution for medium-sized enterprises, research institutions, and educational organizations. Additionally, CommC ensures scalability and adaptability, dynamically adjusting resource allocation based on workload demands. In the current system state, an administrator is responsible for the resource allocation task. Our real-world experiments validate its ability to handle machine learning, big data analytics, and web services efficiently.

Beyond cost and performance benefits, CommC contributes to environmental sustainability by extending the lifespan of older hardware. By repurposing existing systems, the platform helps mitigate electronic waste (E-waste), offering a greener alternative to cloud computing, which relies on energy-intensive data centers. Furthermore, CommC enhances security and fault tolerance by leveraging KVM-based virtualization and Kubernetes security policies, ensuring workload isolation, access control, and resilience against cyber threats.

Our novelty: The CommC system was developed as a direct response to the high cost of cloud infrastructure, which often makes large-scale computing inaccessible to medium-sized enterprises, research institutions, and educational organizations. Even though cloud platforms like Google Cloud, AWS, and Microsoft Azure offer scalable computing solutions, their recurring costs, data transfer fees, and vendor lock-in present significant financial and operational challenges. Many organizations face the dilemma of either investing in expensive cloud services or struggling with underutilized local hardware, which remains idle for large portions of the day. CommC bridges this gap by providing a scalable, cloud-like environment that repurposes existing hardware into a cost-efficient and high-performance computing cluster. Additionally, CommC enables cost savings of up to 6×–8× compared to cloud platforms by eliminating compute instance rental fees, storage costs, and network egress charges. Instead of paying for cloud infrastructure that scales with usage, organizations using CommC can leverage their own underutilized hardware, transforming idle machines into an efficient computing cluster.

Beyond cost efficiency, CommC promotes sustainability by extending the lifespan of commodity hardware, reducing electronic waste (E-waste), and lowering energy consumption compared to large cloud data centers. Unlike hyperscale cloud providers, which operate energy-intensive facilities, CommC runs on local infrastructure, offering a greener alternative without sacrificing computational power.

In summary, the proposed architecture was designed to eliminate the financial and operational barriers of cloud computing by offering an affordable, scalable, and secure alternative that enables organizations to fully utilize their existing hardware investments.

The remainder of this paper is structured as follows: Section 2 reviews related technologies and services, highlighting their roles within the proposed system. Section 3 details the system architecture, and Section 3.1 describes the main cluster components. Section 3.2 discusses the network requirement for the system. The software used for creating CommC is analyzed in Section 4 while the physical implementation of the cluster, supported applications, use cases, and cost analysis are described in Sections 5 and 5.2. Finally, conclusions, limitations, and future directions are outlined in Section 6.

2. Preliminaries

2.1. Hypervisors

A hypervisor, or Virtual Machine Monitor (VMM), is a foundational technology that enables multiple virtual machines (VMs) to run on a single physical machine by abstracting hardware resources. Hypervisors are categorized into two types: Type 1 (bare-metal), which operates directly on hardware for high performance (e.g., VMware ESXi, Hyper-V), and Type 2 (hosted), which runs on an existing operating system and is commonly used for development and testing (e.g., VirtualBox [12] and VMware Workstation [13]).

Hypervisors enable resource sharing, secure isolation, and workload flexibility, and they leverage technologies like Intel VT-x and AMD-V for hardware-assisted virtualization. They provide essential functionalities such as live migration, snapshots, and virtualized hardware components, facilitating resource optimization and high availability. However, they pose challenges such as performance overhead, security vulnerabilities, and the need for skilled management [14].

Even though the terms "hypervisor" and "VMM" are often used interchangeably, distinctions exist in specific contexts. The hypervisor oversees overall resource allocation across VMs, whereas the VMM focuses on managing individual VMs. Together, they form the backbone of modern IT, enabling cloud computing, enterprise data centers, and hybrid environments. Despite challenges like resource competition and operational costs, hypervisors remain critical for scalable, cost-effective, and flexible IT infrastructure. Recent research [15–17] focuses on improving energy efficiency and resource management in cloud data centers, addressing critical challenges such as high energy consumption, carbon footprint, and the operational costs associated with virtualized environments. One study highlights strategies for virtual machine (VM) consolidation to optimize resource usage and minimize power consumption while maintaining service-level agreements (SLAs). It explores techniques like dynamic voltage and frequency scaling (DVFS), soft scaling, and resource throttling to balance energy efficiency with performance requirements.

On the other hand, the second delves into holistic approaches to cloud resource management, leveraging advanced algorithms and predictive techniques to adapt resource allocation dynamically. This includes handling heterogeneous workloads, minimizing VM migration overhead, and enhancing system reliability. The research emphasizes reducing SLA violations and energy consumption while proposing new metrics and methodologies for assessing consolidation strategies.

Kernel-based Virtual Machine (KVM) [18] and Quick Emulator (QEMU) [19] are two widely used open-source virtualization tools that serve distinct yet complementary roles in enabling efficient and flexible virtualized environments. KVM and QEMU are complementary yet distinct tools in the virtualization ecosystem. KVM is a Type 1 hypervisor integrated into the Linux kernel, offering near-native performance by leveraging hardware virtualization extensions like Intel VT-x [20] and AMD-V [21–23]. It excels in scalability, security, and reliability, making it a preferred choice for enterprise-grade virtualization and cloud computing platforms. In contrast, QEMU is a versatile open-source tool that functions both as an emulator and a virtualization solution. While it can emulate hardware for cross-platform development, its full potential is realized when paired with KVM, where it provides device emulation and virtualized hardware, enabling high-performance virtual machines.

The key difference lies in their core focus and use cases. KVM is optimized for large-scale production environments requiring robust performance and advanced features like live migration and container integration. QEMU, on the other hand, shines in development and testing scenarios, particularly where hardware emulation or cross-architecture compatibility is needed. Together, KVM and QEMU offer a powerful combination—KVM providing the performance and scalability of hardware-assisted virtualization, and QEMU adding flexibility and comprehensive emulation capabilities, making them indispensable for hybrid and diverse workload requirements.

2.2. Containers and Their Impact on Virtualization

Containers are lightweight and portable units of software that encapsulate an application along with all its dependencies into a single entity. This packaging ensures consistent performance across diverse environments, making containers highly reliable for modern computing needs. Unlike virtual machines (VMs), which emulate an entire hardware stack and require individual operating systems, containers share the host operating system's kernel. This architectural difference makes containers faster to start and more resource-efficient, as they isolate processes at the OS level.

The primary distinction between containers and VMs lies in their design. Containers are optimized for lightweight, agile operations, making them ideal for microservices [24] and cloud-native applications. In contrast, VMs provide stronger isolation by running full operating systems, making them better suited for legacy applications, untrusted workloads, or scenarios requiring diverse OSs. Containers excel in portability and scalability, whereas VMs offer comprehensive flexibility and robust isolation.

Container technologies such as Docker and Kubernetes have transformed the management and orchestration of containerized workloads. Docker simplifies container creation and deployment, and Kubernetes automates the scaling, management, and operation of containers across clusters. Containers are integral to modern development workflows, particularly in Continuous Integration/Continuous Deployment (CI/CD) pipelines and microservices architecture. Conversely, VMs remain the preferred solution for running legacy systems, multi-tenant environments, and workloads requiring diverse operating systems.

Linux Containers (LXC)

Linux Containers (LXC) [25] is one of the earliest containerization technologies, enabling multiple isolated Linux environments to run on a shared Linux kernel. Introduced in 2008, LXC leverages kernel features like namespaces for process isolation and control groups (cgroups) for resource management. Each container operates as a process on the host system, making LXC lightweight and efficient compared to traditional virtualization. LXC has influenced the development of modern container tools and remains a foundational technology for Linux-centric environments

2.3. Reverse Proxies

A reverse proxy is a server that acts as an intermediary between client devices and backend servers. It forwards client requests to the appropriate backend server and returns the server's response to the client. Unlike a forward proxy, which accesses resources on behalf of clients, a reverse proxy operates on behalf of servers, simplifying and managing server interactions for the client. Reverse proxies are a key component in modern web architectures, offering benefits such as enhanced scalability, security, and performance.

One of the primary functions of a reverse proxy is to serve as a Layer 7 load balancer that distributes the incoming traffic across multiple backend servers. This ensures optimal resource utilization, prevents server overload, and improves overall system responsiveness. Additionally, reverse proxies can cache responses from backend servers, speeding up client interactions and reducing redundant processing. Security is another critical advantage, as reverse proxies mask the identity and IP addresses of backend servers, minimizing the attack surface and protecting against direct cyberattacks.

Reverse proxies are extensively used in content delivery networks (CDNs) to cache static content closer to users, facilitating faster delivery. They also centralize SSL/TLS termination, offloading encryption and decryption tasks from individual servers, which simplifies certificate management and enhances performance. Moreover, reverse proxies support advanced functionalities such as URL rewriting, request filtering, and authentication enforcement, making them indispensable for managing and optimizing web traffic in complex network infrastructures.

3. System Architecture

The high-level architecture of our proposed system is illustrated in Figure 1. Our cluster is composed of multiple physical nodes that are equipped with a similar set of components, and they collectively form a cohesive virtualized environment managed through a centralized cluster management layer.

The *Virtualization Cluster* is represented at the top, symbolizing the unified virtualized environment that spans across all physical nodes. This centralized layer ensures that all nodes work in harmony, managing resources and Virtual Machines (VMs) across the entire cluster. By linking multiple nodes together, the cluster provides redundancy, load-balancing, and scalability, allowing the virtualized environment to handle varying workloads and ensure high availability. In its current state, CommC operates under a hierarchical resource management structure. At a higher level, an administrator is responsible for allocating the necessary resources for each virtual machine, similar to how resource management is handled in cloud-based environments. The actual distribution of tasks and resource assignment within the system is managed by the cluster orchestrator, which, in this case, is Kubernetes.

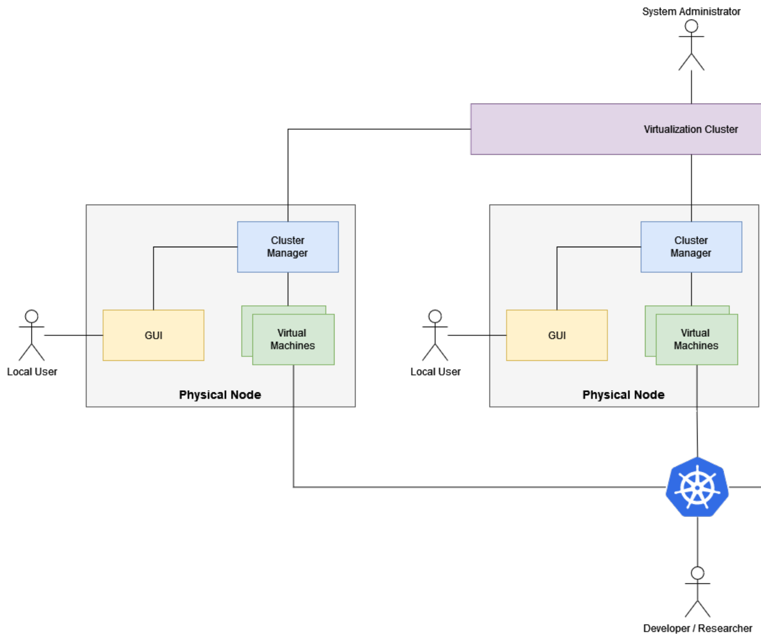

Figure 1. High-level architecture schema.

Each physical node contains three main components:

1. *GUI* (in yellow): This provides a graphical user interface for interacting with the node locally. The GUI offers a user-friendly way to interact with the virtual machines running on the cluster.
2. *Cluster Manager* (in blue): This component handles the orchestration of resources within the node and coordinates with the other nodes in the cluster. The Cluster Manager communicates with the centralized Virtualization Cluster layer, ensuring that resources are allocated effectively and that VMs can be managed across nodes.
3. *Virtual Machines* (in green): These are the actual virtualized instances running on each physical node. VMs are isolated environments that simulate separate operating systems, allowing multiple applications to run concurrently on a single physical server. The hypervisor on each node allocates resources to these VMs and ensures their isolation from each other.

At the bottom of the diagram, the *Kubernetes* icon indicates that some of the VMs within the cluster are running a separate Kubernetes cluster. This setup enables Kubernetes to orchestrate containerized applications on top of virtualized infrastructure, providing an additional layer of flexibility. By running Kubernetes within VMs, the architecture can support containerized workloads alongside traditional virtualized applications, enabling a hybrid environment within each physical node.

In summary, each node is equipped with local management and monitoring capabilities, and virtual machines provide isolated environments for running both standard applications and a Kubernetes cluster.

3.1. Cluster Node Stack

Figure 2 presents a comprehensive architecture of the main interconnected components that comprise a single node in our cluster design. The architecture is organized into three main sections: the *Local User's GUI*, the *Virtualization Stack*, and the *Operating System*, all unified under the *Cluster Node*.

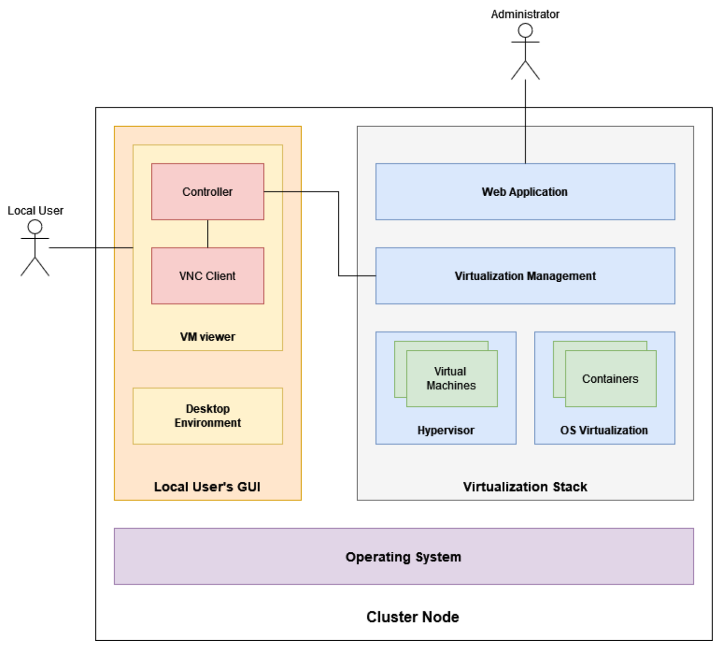

Figure 2. Architecture of cluster's single node.

On the left, the *Local User's GUI* serves as the interface through which users seamlessly interact with the virtual environment. Key components, including the *Controller* and *VNC Client* within the *VM Viewer*, provide direct control and monitoring capabilities for virtual machines, enabling local users to manage the VMs intuitively and effectively. A *Desktop Environment* is required for these components to function properly.

At the heart of the diagram, the *Virtualization Stack* drives the entire virtualization process. A *Web Application* layer at the top enables remote access and management, offering a centralized web-based interface for streamlined control. Directly beneath, the *Virtualization Management* layer orchestrates the entire virtual infrastructure, overseeing the deployment, monitoring, and administration of both virtual machines and containers. This centralized management ensures cohesive and efficient control over all virtualized resources and provides the functionality for connecting each node to the Virtualization Cluster.

Within the *Virtualization Stack*, the *Hypervisor* and *OS Virtualization* components represent the two main virtualization technologies powering the system. The *Hypervisor* facilitates the creation of *Virtual Machines* (VMs), each operating as an independent, fully isolated environment with its own dedicated resources and operating system. This design is ideal for use cases requiring high levels of isolation and security. In contrast, *OS Virtualization* supports *Containers*, which are lightweight and efficient and share the host OS kernel while remaining securely isolated from one another. Containers are well-suited for rapid scaling and microservices architectures, allowing for agile and resource-efficient deployments.

The *Operating System* layer at the base underpins the entire setup, providing essential resources such as CPU, memory, and storage for both VMs and containers. As the foundation for the Virtualization Stack, it establishes a stable and reliable environment that enables seamless operation and scalability across all virtualized workloads.

Overall, this cluster node architecture provides a robust, flexible framework that integrates user-friendly interfaces, powerful virtualization management, and efficient resource allocation. Its dual support for VMs and containers ensures that diverse workload requirements are met with precision and agility, making it an ideal solution for modern cloud, data center, and enterprise environments.

3.2. Network Schema

The network configuration requirements for the proposed setup are quite complex and intricate. The users' virtual machines should remain isolated, whereas those that form the Kubernetes cluster must communicate with each other. Every virtual machine should be able to access the internet but should not be directly accessible from it. However, a selected few services must be exposed to the World Wide Web; otherwise, the system's functionality will be rather limited. To satisfy all those needs, we came up with the architecture that is demonstrated in Figure 3.

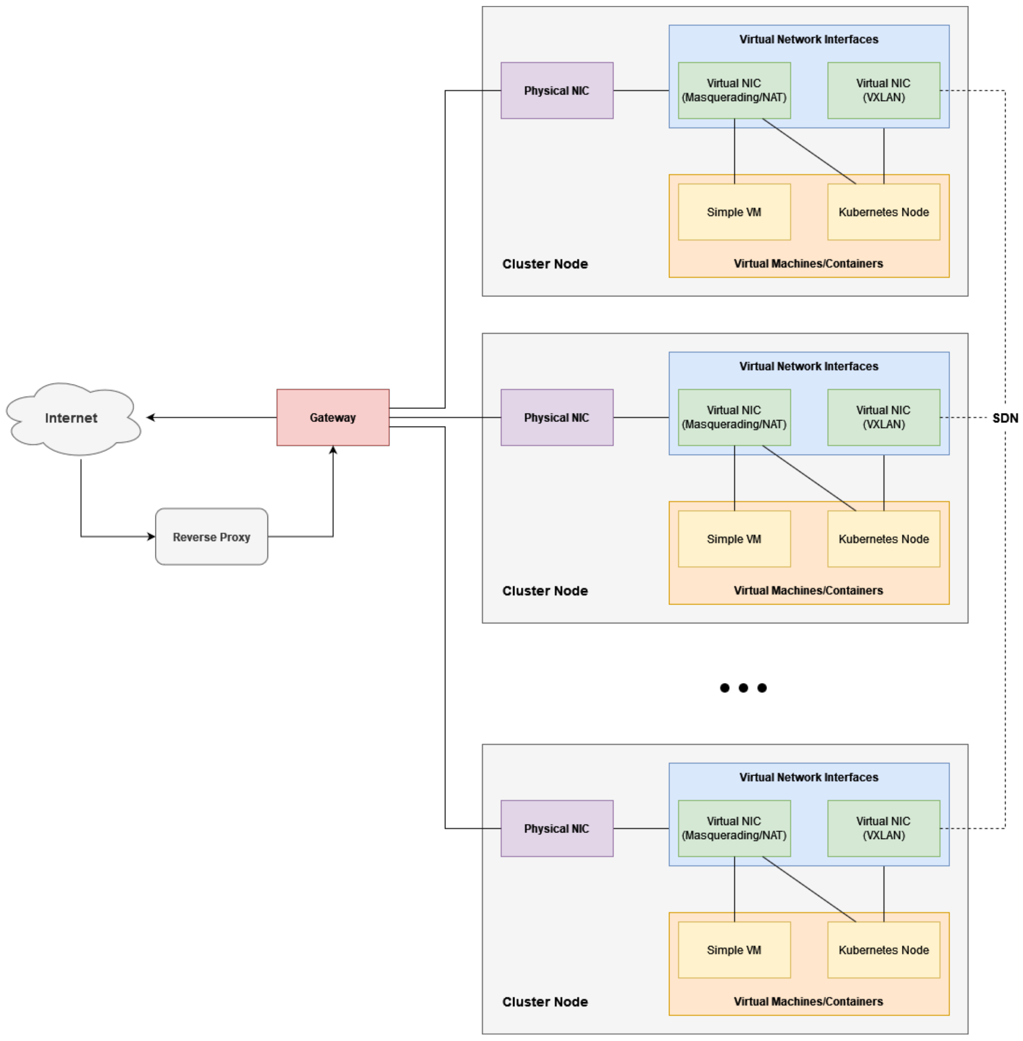

Figure 3. Network schema.

This network architecture is designed specifically to combine traditional virtual machines with containerized workloads, such as a hybrid infrastructure running both simple VMs and Kubernetes clusters.

At the edge of the network, traffic from the internet passes through a reverse proxy, which intermediates requests and forwards them to appropriate resources within the cluster. A gateway sits between the reverse proxy and the cluster, handling the routing to ensure that external traffic is appropriately directed to internal systems.

Within the cluster, each node contains both physical and virtual networking layers. The physical network interface card (NIC) provides the hardware connection between the node and the broader network, and virtual network interfaces enable communication within the node and across the cluster. Two main types of virtual interfaces are used: NAT interfaces and VXLAN interfaces. NAT interfaces allow internal workloads, such as virtual machines and containers, to access external networks while masking their internal IP addresses, providing security, simplifying routing, and saving static IPs. VXLAN interfaces create an overlay network, allowing seamless communication between workloads across different nodes as if they were on the same local network.

Inside each cluster node, a mix of virtualized environments operates, including simple virtual machines and Kubernetes-managed nodes that host containerized applications or pods. These workloads rely on the virtual network interfaces to interact both within the node and across the cluster. By combining these networking layers, the system provides a secure, flexible, and scalable environment for managing hybrid workloads that include both traditional VMs and containerized applications. This configuration enables smooth communication between distributed components while maintaining efficient and secure access to external networks.

3.3. Security

A publicly accessible cluster requires stringent measures to minimize its attack surface and protect against unauthorized access. In the configuration that we propose, the firewall is set to allow only a small number of ports, which have been intentionally randomized, rather than using the default well-known ports. SSH access via the root account is completely disabled, and VNC functionality is restricted to the local network to reduce external visibility. All internet-bound traffic is routed through Cloudflare, which serves as a reverse proxy and secures data transmissions with SSL encryption. Furthermore, user accounts are grouped according to role-based permissions, ensuring that privileges are granted only as needed and preventing inadvertent or malicious misuse. Multi-tenancy challenges are managed by allocating separate virtual machines with dedicated storage to each user, thereby containing any potential security incidents within isolated environments.

4. Technologies and Services

4.1. Proxmox Virtual Environment (Proxmox VE)

Proxmox Virtual Environment (Proxmox VE) [26] is an open-source platform for efficiently managing virtual machines (VMs) and containers. Built on Debian Linux, it integrates Kernel-based Virtual Machine (KVM) for full virtualization and Linux Containers (LXC) for lightweight containerization, providing flexibility to handle diverse workloads. Proxmox VE supports both single-node setups and scalable cluster deployments, making it suitable for various infrastructure requirements.

The platform features a centralized management interface accessible through a web-based graphical user interface (GUI), which simplifies tasks such as live migration, back-ups, and high-availability configurations. Advanced users can utilize a RESTful API, a command-line interface, and role-based access control for more granular management. The GUI is designed for ease of use, featuring a resource tree, real-time log monitoring, and secure SSL-encrypted access. It also supports multi-factor authentication and integrates with authentication sources like LDAP and Microsoft Active Directory.

Proxmox VE offers a flexible storage model, supporting both local and shared storage solutions such as NFS, SAN, Ceph RBD, and ZFS. This flexibility enables live migration and efficient management of storage resources across cluster nodes. An integrated backup tool, vzdump, is included to create consistent snapshots of running VMs and containers, ensuring data integrity and simplifying restoration processes.

In comparison with other similar systems, like OpenStack, Proxmox is a more lightweight and straightforward platform, making it well-suited for labs or small-to-medium enterprises (SMEs), where simplicity and ease of setup are key. Its intuitive web interface, which bundles virtualization (KVM) and containers (LXC) into one cohesive system, minimizes the need for multiple specialized services and extensive configuration. This makes Proxmox faster to deploy and easier to maintain, especially for teams with limited time or resources. In contrast, OpenStack is a full-fledged private cloud solution that offers a rich ecosystem of components for large-scale, multi-tenant environments. However, it is typically more complex to install, operate, and scale, requiring specialized expertise. For lab environments or SMEs that value a more direct, out-of-the-box virtualization solution without extensive overhead, Proxmox's simplicity, resource efficiency, and robust feature set present a compelling choice.

4.2. Docker

Docker [27] revolutionized containerization by making it accessible and user-friendly. It packages applications and their dependencies into standardized containers, ensuring consistency across development, testing, and production environments. Docker's integration with tools like Docker Hub and orchestration platforms like Kubernetes has cemented its position as a cornerstone of modern DevOps practices. Its focus on portability, scalability, and efficiency makes Docker essential for cloud-native application development.

4.3. Kubernetes (K8s)

Kubernetes [9] is a robust orchestration platform designed to manage, scale, and automate containerized applications. It organizes containers into pods, enabling seamless communication and scaling across clusters. Kubernetes' features, such as rolling updates, self-healing, and persistent storage integration, have made it the industry standard for container orchestration in cloud-native and distributed systems. While its complexity presents a learning curve, Kubernetes remains essential for managing complex, large-scale applications.

4.4. Desktop Environments

Desktop environments offer a graphical workspace that abstracts the complexities of command-line interfaces, providing users with a visually appealing and customizable experience. They are designed to cater to diverse needs, from resource-efficient options like Xfce to more feature-rich and visually intensive alternatives. These environments support extensive customization, enabling users to personalize layouts, themes, and workflows, and they typically come with integrated tools like file managers and system settings to streamline usage.

Xfce is a lightweight desktop environment known for its balance of performance, functionality, and resource efficiency. Built on the GTK toolkit, Xfce is modular in design, allowing users to replace or customize components. Its low system requirements make it ideal for older hardware or resource-constrained environments. Xfce supports extensive theming and customization, offering a user-friendly and stable desktop experience suitable for both personal and professional use.

4.5. Graphical Desktop-Sharing Systems

Graphical desktop-sharing systems enable remote access to and control of computers over networks, facilitating tasks like troubleshooting, collaboration, and telecommuting. These systems use a client-server model, with protocols such as Remote Desktop Protocol (RDP) and Virtual Network Computing (VNC) enabling secure and seamless interactions. Desktop-sharing technologies are widely adopted in modern hybrid work environments, offering real-time collaboration and remote system management.

VNC is a cross-platform remote desktop protocol that uses the Remote Framebuffer Protocol (RFB) found in [28] to share graphical desktops between devices. Its simplicity and compatibility across operating systems make it a popular choice for remote access.

VNC operates via a client-server model, capturing the host desktop environment and transmitting it to a remote client, and bandwidth-intensive advancements like compression and adaptive quality have improved its performance. VNC is widely used in IT support, education, and system administration.

TigerVNC Client [29] is high-performing and uses secure software to access remote desktops. It supports modern graphics capabilities, high-resolution displays, and encryption protocols like TLS, ensuring smooth and secure remote desktop connections. Its cross-platform compatibility and intuitive interface make it an efficient tool for remote work, system management, and troubleshooting, catering to both novice and advanced users.

4.6. Cloudflare

Cloudflare is a prominent web infrastructure and security platform that functions as a powerful reverse proxy server. By sitting between client requests and backend servers, Cloudflare intercepts, optimizes, and secures web traffic while protecting backend servers from direct exposure to the internet. This intermediary role enables Cloudflare to deliver enhanced performance, robust security, and flexibility in managing web applications.

Cloudflare secures web traffic with signed SSL/TLS certificates, offering end-to-end encryption to protect data from eavesdropping and tampering. By terminating SSL connections at the edge of its global network, Cloudflare reduces the computational load on backend servers. It supports various SSL configurations, such as flexible SSL for basic encryption and strict SSL for maximum security, catering to diverse business needs.

Cloudflare also provides extensive customization options through its rules engine. Users can create firewall rules to block malicious traffic, implement rate-limiting to prevent abuse, and use page rules to optimize performance or redirect visitors. Advanced features like Workers and Transform Rules enable scripting and dynamic customization, allowing businesses to deploy serverless applications, rewrite requests, and adapt content on the fly. These capabilities make Cloudflare a versatile and essential tool for securing and optimizing web applications, enhancing user experiences, and simplifying traffic management.

5. Applications, Use Cases, and Cost Analysis

The deployment of a cluster of commodity hardware provides an affordable and scalable solution to run various workloads, including hosted services, machine learning experiments, and big data processing. This section highlights the primary applications and use cases supported by the cluster, which demonstrate its versatility and resource efficiency. In addition, a cost analysis will be conducted to evaluate the economic benefits of utilizing commodity hardware, emphasizing its cost-effectiveness compared to traditional high-end computing solutions.

We implemented our proposed architecture in a research laboratory at the University of Patras equipped with 16 personal computers (PCs), each powered by an Intel® Core™ i7-10700 CPU @ 2.90 GHz processor, except for two systems that are equipped with an intel i7-3770K processor at 2.50 GHz and one that uses an Intel® Core™ i7-4770 @ 3.40 GHz. Fifteen of these PCs are configured with 16 GB of memory, and one has 24 GB. In terms of storage, the systems are equipped with 512 GB Solid State Drives (SSDs). Detailed specifications are shown in Table 1. Additionally, the three systems with different processors feature RAM and storage components of varying speeds and manufacturers, allowing us to evaluate CommC's performance under heterogeneous hardware conditions. The first version of our system is publicly available on GitHub and can be accessed using the GitHub link, accompanied by a detailed installation tutorial and troubleshooting documentation.

Table 1. System specifications.

Model	Number of PCs	Processor	RAM	Storage
Dell OptiPlex 5090	13	Intel® Core™ i7-10700 @ 2.90 GHz (Intel Corporation, Santa Clara, CA, USA)	16 GB	512 GB
Custom System 1	2	Intel® Core™ i7-3770K @ 2.50 GHz (Intel Corporation, Santa Clara, CA, USA)	16 GB	2 × 512 GB
Custom System 2	1	Intel® Core™ i7-4770 @ 3.40 GHz (Intel Corporation, Santa Clara, CA, USA)	24 GB	2 × 512 GB
Total	**16**	**240 Threads**	**266**	**8.5 TB**

5.1. Application and Use Cases

The cluster serves as a backbone for various hosted services that support research, collaboration, and operational needs. These services include APIs, web hosting, and collaborative platforms, which leverage the cluster's distributed architecture for reliability and performance. DiscMycoVir is an elegant and user-friendly web platform for biological analysis, emphasizing a comprehensive and detailed result-reporting user experience. The Overleaf Server is a collaborative LaTeX editing platform within a Kubernetes environment tested specifically on a MicroK8s cluster, utilizing NFS for persistent storage. Table 2 summarizes the hosted services and their respective types.

Table 2. Services hosted in the Lab's cluster.

Description	Type	Hardware Reservations	Software
HypeRec's backend [30]	API	2 CPUs, 4 GB of memory	Python 3.11 (sklearn, keras, Flask), MongoDB 7.0
DiscMycoVir	Platform	12 CPUs, 16 GB of memory	Docker, Python 3.11 (Flask), MySQL 8.0, Trimmomatic 0.36, FastQC 0.12, Trinity RNA-Seq 2.11
Lab's Website	Web Hosting	1 CPU, 1 GB of memory	Wordpress 6.7, PHP 8.3, MariaDB 11.6
Lab's Overleaf Server	Platform	1 CPU, 1 GB of memory	Sharelatex 5.3, MongoDB 7.0, Redis 7.4
VMs for Lab Staff	IaaS	Varies	Varies

The cluster is extensively utilized for Apache Spark experiments, enabling distributed computing for tasks such as natural language processing, graph analytics, and community detection. These experiments make full use of the cluster's computational and memory resources, as shown in Table 3. It should be noted here that the RAM shown in Table 3 expresses the memory usage for each experiment and does not account for the necessary RAM that the operating system and the containers need. Therefore, an overhead of 10–20% exists, which our system is more than capable of handling.

The SparkNLP experiment leverages Spark for natural language processing tasks. It utilized seven nodes, consuming 28 CPUs and 68 GB of RAM over a total duration estimate of 80 h. Neural Networks with Torch Distributor integrates Spark with TorchDistributor to run distributed neural networks. The resources for the experiment were four nodes, 16 CPUs, and 32 GB of RAM over 80 h. The Spark Bayesian Networks [31] experiment is focused on implementing Bayesian networks in Spark. It used seven nodes, consuming 28 CPUs and 68 GB of RAM over 40 h.

Table 3. Resource allocation and usage exclusively for Apache Spark experiments.

Description	Apache Spark Version(s)	Nodes	CPU	RAM (GB)	Usage (h)	Avg. RAM Usage
Research on NLP	3.4.1	7	28	68	80	92%
Research on NNs with TorchDistributor	3.5.3	4	16	32	80	95%
Spark Bayesian Networks [31]	2.4.3, 3.4.1	7	28	68	40	87%
Triangle Counting in Large Historical Graphs [32]	3.5.3	9	36	104	192	78%
Community detection in large social networks (part 1)	3.5.3	13	28	80	144	91%
Community detection in large social networks (part 2)	3.5.3	11	44	148	240	96%

The next series of experiments form part of ongoing research on graph analytics, with objectives like counting triangles in graph communities [32]. The algorithms under evaluation were implemented in three phases. The first one utilized nine nodes, 36 CPUs, and 104 GB of RAM, running for a total of 192 h. The second and third phases have a broader aim to detect community structures within graphs, and the resource usage was adjusted accordingly. The second phase was executed on 13 nodes with 28 CPUs and 80 GB of RAM for 144 h, and the third one required significant resources, with 11 nodes, 44 CPUs, and 148 GB of RAM, making it the most resource-intensive experiment, lasting for 240 h.

5.2. Cost Analysis

In the previous subsection, several applications of the cluster were presented, showcasing its versatility and utility. In this subsection, we will focus on conducting a detailed cost analysis by estimating the expenses associated with running the same experiments on prominent cloud platforms, namely Google Cloud, Amazon Web Services (AWS), and Microsoft Azure. This analysis aims to compare the resource requirements of each experiment and their corresponding costs on these industry-leading platforms, providing insights into the economic efficiency of utilizing a commodity hardware cluster versus well-established cloud services.

The analysis will focus on three key aspects: computational resources, storage, and the network. The computational resources aspect includes the number of required instances, the core count, and the RAM necessary for each experiment. Regarding storage, each experiment was allocated 512 GB, and this will be factored into the cost analysis for each cloud platform. However, network traffic costs—both inbound and outbound—will be excluded from the analysis, along with any additional costs associated with disk read and write operations enforced by certain services. This simplification will result in a slightly lower overall estimated cost for the experiments, but it ensures a more streamlined and focused cost comparison.

Each cloud platform provides a cost estimator tool that allows users to estimate the necessary funds for their system based on the specific requirements of an experiment. However, this approach often results in a rough estimation of the cost, or a "guestimate", due to variations in how each service calculates expenses. For instance, differences in pricing models, regional availability, instance types, and additional fees for features, such as data transfer or specialized hardware, can cause discrepancies in the estimations. As a result, although these tools offer a useful starting point, they may not always reflect the precise costs incurred during actual usage. Therefore, we will analyze each axis separately.

Firstly, for each cloud service, we selected the appropriate configuration and virtual machine (VM) type based on the resource requirements of each experiment. Table 4 presents the chosen specifications for each platform. It is evident that, in some cases, the RAM requirements slightly exceed the experiment's actual needs. However, we opted for the configuration that most closely aligned with the resource demands of the experiments while ensuring the necessary computational capacity was met. This pragmatic approach balances cost efficiency and functionality, leveraging the most suitable VM types available on each platform.

Table 4. Cost comparison of VM types across different cloud services.

	GC	Azure	AWS
VM Type	c4a-standard-4	B4as	t4g.xlarge
Cores	4	4	4
RAM (GB)	16	16	16
Base Core Frequency (Ghz)	3	2.45	2.5
Cost Rate (€)	0.21	0.165	0.13

Cost rates and specifications may vary depending on the time of access.

Table 4 provides details about the cost rate per hour of usage for each selected configuration. It is important to note that these rates may vary depending on the date of access or the user's geographic location. For this analysis, the cost rates were retrieved on 22 January 2025, from Greece, with the servers hosted in the Central Europe region. These factors may slightly influence the overall cost and should be taken into consideration when interpreting the results.

$$\text{Cost} = \text{Number of Instances} \cdot \text{Cost Rate} \cdot \text{Usage Hours} \tag{1}$$

The costs of each experiment regarding the computational aspect were calculated as shown in Equation (1). The results are depicted in Table 5. The table summarizes the estimated costs for running the experiments on three major cloud platforms: Google Cloud, Azure, and AWS. Each experiment is listed, along with its corresponding cost on each platform, calculated based on the VM configurations and usage hours. The total cost for all experiments is also provided, highlighting the differences in cost efficiency among the platforms.

Table 5. Estimated costs for running experiments on Google Cloud, Azure, and AWS.

Description	**Google (€)**	**Azure (€)**	**AWS (€)**
Research on NLP	117.6	92.4	75.26
Research on NNs with TorchDistributor	67.2	52.8	43.00
Spark Bayesian Networks [31]	58.8	46.2	37.63
Triangle Counting in Large Historical Graphs [32]	362.88	285.12	232.24
Community detection in large social networks (part 1)	393.12	308.88	251.59
Community detection in large social networks (part 2)	554.4	435.6	354.81
Total	**1554.00**	**1221.00**	**994.56**
Total + Storage [1]	**1564.24**	**1343.88** [2]	**1006.34**

[1] Total cost, including a 512 GB Standard SSD across cloud platforms. [2] Azure costs might include additional features or configurations.

The aforementioned cost is considered the monthly cost of the service used to conduct each experiment. Each service offers various storage options. However, we chose a persistent storage SSD of 512 GB, with the cost depicted in the last row of Table 5.

Our cluster comprises 16 computers, each with a maximum power supply of 200 watts. This results in a total energy consumption of 3.2 kWh for the entire cluster per hour. In a worst-case scenario, where all systems operate at full capacity continuously for 24 h a day over a month (720 h), the total cost would amount to $3.2 \cdot 24 \cdot 30 \cdot 0.15 = 345.6$€, where 0.15 is the average cost of 1 kWh in Greece. This duration is more than sufficient to complete the required experiments, and the overall cost remains significantly lower compared to alternative cloud platforms.

If commodity hardware were not utilized, the alternative approach would require the acquisition of pre-configured enterprise systems from major vendors, which comes with significantly higher costs. Systems such as the Dell PowerEdge R7525, HPE ProLiant DL385 Gen10 Plus, Lenovo ThinkSystem SR645, and Supermicro A+ Server offer high-performance configurations, typically featuring 128 to 240 threads, 256–512 GB RAM, and 8–16 TB of SSD storage. However, these solutions come at a steep price, ranging from EUR 15,000 to over EUR 30,000, depending on CPU configurations, memory capacity, storage options, and vendor support packages. For instance, a Dell PowerEdge R7525 with 128 cores, 256 GB RAM, and 8 TB NVMe storage can cost between EUR 18,000 and EUR 25,000, while an HPE ProLiant DL385 Gen10 Plus with similar specifications ranges from EUR 20,000 to EUR 30,000. These prices vary further based on warranties, performance tiers, and additional enterprise support options. In contrast, CommC leverages existing commodity hardware, significantly reducing upfront investment costs while providing a scalable and flexible computing environment. This cost-effective approach makes high-performance computing more accessible to organizations that may otherwise struggle with the financial burden of traditional enterprise solutions.

Discussing the performance of our design among popular cloud alternatives is a difficult task. The reason that gave birth to the proposed system design, apart from sustainability, was the high cost one must face in order to take advantage of such structures. However, when observing the hardware specifications of our implemented cluster and the alternatives in Table 4, one would expect that our approach should outperform Azure and Amazon Web Services due to higher CPU clock speed, and Google Cloud would be the best performer regarding tackling the tasks shown in Table 3. This statement by itself might be wrong. There are multiple factors that affect the performance of such systems, such as disk read/write speeds and network transfer speeds. In this study, we chose that each design would contain similar a SSD disk, in order to minimize the speed difference and the cost of the alternatives. Finally, in terms of networking, our commodity hardware is connected to a local network, which can sometimes be slower than virtual machines running on a single physical system. However, cloud providers do not guarantee that their virtual machines will be hosted on the same physical machine. They offer network plans for a fixed fee or based on the amount of transferred data, which will further elevate the cost. To sum up, this study does not attempt to compare the performance of our design versus cloud alternatives but offers a sustainable, highly customizable solution that can satisfy the needs of at least medium-sized organizations and enterprises.

6. Conclusions and Future Work

The proposed CommC system effectively demonstrates the potential of repurposing commodity hardware into a high-performing, multi-purpose cluster. By integrating advanced virtualization and containerization technologies, including KVM, Docker, and Kubernetes, CommC provides a scalable, flexible, and cost-efficient solution tailored to diverse workload demands. This hybrid architecture optimizes underutilized hardware and significantly reduces operational costs while offering enhanced fault tolerance and security.

The use of commodity hardware extends the lifecycle of existing systems, contributing to sustainability by addressing the growing concern about E-waste. Through strategic resource allocation and innovative management approaches, CommC empowers medium-sized enterprises and educational institutions to adopt high-performance computing solutions without incurring prohibitive costs.

However, we were unable to find any recent (2024–2025) peer-reviewed publications specifically addressing the energy consumption and environmental impact of cloud providers. Many articles [33–35] exist, revealing the fact that over the past five years, the operation of data centers by major tech companies has led to more than USD 5.4 billion in public health costs due to air pollution. In 2023 alone, these costs were estimated at

USD 1.5 billion, marking a 20% increase from the previous year. Google's data centers were identified as the largest contributors, accounting for USD 2.6 billion in health-related expenses between 2019 and 2023. The energy consumption of these centers, often reliant on fossil fuels, has been linked to air pollution's contribution to ailments such as cancer and asthma.

Furthermore, popular cloud service providers do not offer a carbon emission estimator based on a chosen configuration, making it challenging to accurately assess the environmental impact of different setups. As a result, a direct comparison of energy consumption between our system and cloud-based alternatives remains difficult without standardized emission metrics. However, one of the key sustainability advantages of our approach is that it extends the lifespan of existing hardware, preventing it from being discarded as electronic waste. By repurposing commodity hardware instead of decommissioning it, our system reduces E-waste accumulation and contributes to a more sustainable computing model.

The CommC system offers a cost-effective and scalable solution for leveraging commodity hardware, but it is not without its limitations. One significant challenge lies in the complexity of maintaining the architecture. The system integrates multiple advanced technologies, including KVM, Kubernetes, Docker, and Proxmox, which, while offering flexibility and functionality, introduce a level of operational complexity. Organizations with limited technical expertise or small IT teams may find it difficult to manage and troubleshoot such a setup. The steep learning curve associated with configuring and maintaining these tools could require considerable time and effort, potentially offsetting the system's benefits.

Another limitation of CommC that is related to the underlying systems, mainly Proxmox, is the number of nodes that can be added to a single cluster. Proxmox relies on Corosync for synchronization, which can introduce overhead when achieving consensus. According to Proxmox's official documentation, "There is no explicit limit for the number of nodes in a cluster. In practice, the actual possible node count may be limited by the host and network performance. Currently (2021), there are reports of clusters (using high-end enterprise hardware) with over 50 nodes in production". However, other accounts indicate that Corosync may exhibit communication issues in clusters with more than 30 nodes. The primary bottleneck and root of these types of problems lies in the computers' network interface controllers, which, unfortunately, cannot be remedied without a hardware upgrade. Our own deployment, consisting of 16 physical nodes, has not encountered any noticeable delays or discrepancies.

The cost analysis presented in this paper highlights the significant financial advantages of the CommC system over cloud services, even when factoring in additional costs, such as operational expenses and hardware failures. While the comparison primarily focuses on initial expenses and general operational costs, it is important to note that long-term factors like power consumption, cooling, periodic hardware upgrades, and occasional hardware failures do not substantially diminish the cost-efficiency of the CommC system. Even when accounting for these additional expenses, the overall affordability remains superior to cloud-based alternatives, where recurring costs such as subscription fees and data transfer charges often accumulate rapidly over time. Furthermore, the flexibility to avoid vendor lock-in and the ability to use open-source tools further enhance the system's financial viability. By carefully managing operational expenses and leveraging commodity hardware, the CommC system provides a robust and cost-effective alternative to traditional cloud services, even when considering the potential for additional costs.

In the future, we aim to enhance the CommC system by focusing on two primary areas of improvement. First, we aim to enhance resource allocation through intelligent automation powered by machine learning algorithms. These algorithms will dynamically analyze workload patterns to optimize resource distribution, ensuring higher efficiency and improved utilization of the underlying hardware. This approach will minimize the need for manual intervention, making the system more adaptable to varying demands. Additionally, we plan to develop an API that securely integrates with the laboratory's Proxmox cluster to automate processes, generate telemetry, and efficiently manage resources for multiple users. Furthermore, we intend to implement a machine learning model capable of predicting future resource utilization based on historical usage trends, leveraging auto-regressive and moving average techniques to enhance forecasting accuracy.

Second, we intend to develop a user-friendly platform for monitoring and accessing the various services hosted within the CommC environment. This platform would simplify administrative tasks by providing an intuitive interface for visualizing resource usage, managing services, and configuring system parameters. It would also enhance the user experience by allowing seamless access to hosted services, thereby lowering the technical barrier for end-users. These enhancements are expected to improve the scalability, usability, and overall efficiency of the CommC system, paving the way for broader adoption across diverse organizational settings.

Author Contributions: Conceptualization, A.B.; methodology, A.B. and N.-R.K.; software, A.B.; validation, A.B., N.-R.K. and C.M.; formal analysis, A.B.; investigation, A.B.; resources, A.B., N.-R.K. and C.M.; data curation, A.B. and N.-R.K.; writing—original draft preparation, A.B. and N.-R.K.; writing—review and editing, A.B., N.-R.K. and C.M.; visualization, A.B. and N.-R.K.; supervision, C.M.; project administration, C.M.; funding acquisition, C.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The code and system configurations required to replicate the CommC system are publicly available on GitHub and can be accessed using the following link: System Code Repository. The datasets used for the individual experiments mentioned in this paper are not included, as their availability and details are provided in the respective research papers that describe those experiments. These datasets, along with their specific usage, are thoroughly documented in those papers, which are beyond the scope of this work. References to these papers can be found in the bibliography section or will be released in the near future, as they are work in progress.

Acknowledgments: Special thanks to the researchers Ioannis Akarepis, Nikolaos Louridas, Ioakeim Kanellopoulos, Konstantinos Christopoulos, and Konstantinos Tsichlas, who used the CommC cluster for running their experiments. This process greatly helped us to evaluate the impact of our system.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster Computing with Working Sets. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), Boston, MA, USA, 22–25 June 2010.
2. Mohammad, A.; Abbas, Y. Key Challenges of Cloud Computing Resource Allocation in Small and Medium Enterprises. *Digital* **2024**, *4*, 372–388. [CrossRef]
3. Wang, D.; Li, Y.; Zhang, W.; Yu, Z.; Tian, Y.C.; Li, K. EVRM: Elastic Virtual Resource Management Framework for Cloud Virtual Instances. *Future Gener. Comput. Syst.* **2025**, *165*, 107569. [CrossRef]
4. Garrison, D.; Leal, R. Towards HPC and Big Data Analytics Convergence: Design and Experimental Evaluation of a HPDA Framework for eScience at Scale. *IEEE Trans. Cloud Comput.* **2021**, *9*, 458–470.
5. Johnson, K.; Patel, A. Big Data Analytics on HPC Architectures: Performance and Cost. In Proceedings of the IEEE International Symposium on High-Performance Computing, Washington, DC, USA, 5–8 December 2016; pp. 23–30.
6. Xia, Y.; Yang, H. Security-Preserving Live Migration of Virtual Machines in the Cloud. *J. Netw. Syst. Manag.* **2013**, *21*, 15–37.
7. Gupta, R.; Venugopal, A. A Comparative Survey of the HPC and Big Data Paradigms. *IEEE Trans. Big Data* **2017**, *4*, 123–136.
8. Clark, C.; Fraser, K.; Hand, S. Live Migration of Virtual Machines. In Proceedings of the NSDI, Boston, MA, USA, 2–4 May 2005; pp. 273–286.

9. Brewer, E.A. Kubernetes and the path to cloud native. In Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC '15, New York, NY, USA, 27 June–2 July 2015; p. 167. [CrossRef]
10. Jain, M.; Kumar, D.; Chaudhary, J.; Kumar, S.; Sharma, S.; Verma, A.S. Review on E-waste management and its impact on the environment and society. *Waste Manag. Bull.* **2023**, *1*, 34–44. [CrossRef]
11. Halim, L.; Suharyanti, Y. E-Waste: Current Research and Future Perspective on Developing Countries. *Int. J. Ind. Eng. Eng. Manag.* **2020**, *1*, 26–39. [CrossRef]
12. Oracle Corporation. *VirtualBox User Manual*; Oracle Corporation: Austin, TX, USA, 2025.
13. VMware, Inc.. *VMware Workstation Pro Documentation*; VMware, Inc.: Palo Alto, CA, USA, 2025.
14. Singh, J.; Walia, N.K. A Comprehensive Review of Cloud Computing Virtual Machine Consolidation. *IEEE Access* **2023**, *11*, 106190–106209. [CrossRef]
15. Alsbatin, L.; Öz, G.; Ulusoy, A.H. An Overview of Energy-Efficient Cloud Data Centres. In Proceedings of the 2017 International Conference on Computer and Applications (ICCA), Doha, United Arab Emirates, 6–7 September 2017; pp. 211–214. [CrossRef]
16. Shelar, M.; Sane, S.; Kharat, V.; Jadhav, R. Autonomic and energy-aware resource allocation for efficient management of cloud data centre. In Proceedings of the 2017 Innovations in Power and Advanced Computing Technologies (i-PACT), Vellore, India, 21–22 April 2017; pp. 1–8. [CrossRef]
17. Huang, Y.; Xu, H.; Gao, H.; Ma, X.; Hussain, W. SSUR: An Approach to Optimizing Virtual Machine Allocation Strategy Based on User Requirements for Cloud Data Center. *IEEE Trans. Green Commun. Netw.* **2021**, *5*, 670–681. [CrossRef]
18. Fujitsu Laboratories. Kernel-Based Virtual Machine Technology. *Fujitsu Sci. Tech. J.* **2011**, *47*, 350–356.
19. Bellard, F. QEMU, a Fast and Portable Dynamic Translator. In Proceedings of the 2005 USENIX Annual Technical Conference, FREENIX Track, Anaheim, CA, USA, 10–15 April 2005; pp. 41–46.
20. Intel Corporation. Intel® Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel® Technol. J.* **2006**, *10*, 167–178.
21. AMD Corporation. *AMD-V™ Nested Paging*; Technical Report; AMD Corporation: Sunnyvale, CA, USA, 2008.
22. AMD Corporation. *Processor-Based Virtualization, AMD64 Style, Part I*; Technical Report; AMD Corporation: Sunnyvale, CA, USA, 2007.
23. AMD Corporation. *Processor-Based Virtualization, AMD64 Style, Part II*; Technical Report; AMD Corporation: Sunnyvale, CA, USA, 2007.
24. Newman, S. *Building Microservices: Designing Fine-Grained Systems*, 1st ed.; O'Reilly Media: Sebastopol, CA, USA, 2015.
25. Linux Containers Project. Linux Containers. 2024. Available online: https://linuxcontainers.org/ (accessed on 30 December 2024).
26. Ahmed, W. *Mastering Proxmox*, 3rd ed.; Packt Publishing: Birmingham, UK, 2017.
27. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
28. Richardson, T.; Levine, J. *The Remote Framebuffer Protocol*; RFC 6143; RFC Editor: Marina del Rey, CA, USA, 2011.
29. TigerVNC Project. *TigerVNC: High-Performance, Platform-Neutral Implementation of VNC*; TigerVNC Developers: Linköping, Sweden, 2025.
30. Bompotas, A.; Triantafyllopoulos, P.; Raptis, G.E.; Katsini, C.; Makris, C. Towards Exploring Personalized Hyperlink Recommendations Through Machine Learning. In Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization, New York, NY, USA, 1–4 July 2024; UMAP Adjunct '24; pp. 528–533. [CrossRef]
31. Akarepis, I.; Bompotas, A.; Makris, C. Efficient parameter learning for Bayesian Network classifiers following the Apache Spark Dataframes paradigm. *Knowl. Inf. Syst.* **2024**, *66*, 4437–4461. [CrossRef]
32. Christopoulos, K.; Daskalakis, E.; Bompotas, A.; Tsichlas, K. Triangle Counting in Large Historical Graphs. In Proceedings of the 40th ACM/SIGAPP Symposium On Applied Computing, SAC 2025, New York, NY, USA, 31 March–4 April 2025; *in press*.
33. Financial Times. Tech Giants' Data Centers Cause Billions in Public Health Costs. *Financial Times*, 2024.
34. International Banker. *The Environmental Impact of Cloud Computing and the Importance of Greening Data Centres*; Finance Publishing, London, UK, 2024.
35. MIT News. Explained: The Environmental Impact of Generative AI. *MIT News*, 2025.