Computing in Hierarchical Virtual Environments

Evgeniy Ibatullin
St. Petersburg University
St. Petersburg, Russia
e-mail: officevg3n@gmail.com

Alexander Bogdanov
St. Petersburg University
St. Petersburg, Russia
e-mail: a.v.bogdanov@spbu.ru

***Abstract*—Distributed and high-performance computing (HPC) continue to evolve in response to escalating computational demands, yet existing architectures face significant limitations in scalability, latency, and data availability. To overcome these constraints, we propose integrating advanced computational paradigms—such as cloud computing, virtualization, and hierarchical storage architectures—into a unified framework. This paper presents an approach to organizing computing and storage resources for efficient large-scale big data processing. We first identify the foundational technologies required to build such a system, then detail their necessary adaptations to meet the unique demands of modern big data workloads.**

***Keywords*—Hierarchy, distributed systems, virtualization, cloud computing.**

I. Statement of problem

Distributed systems have long been a major focus of research and discussion. Recent studies, however, increasingly conclude that fully integrated systems—built across multiple layers (e.g., computation nodes, storage, and networking)—are essential. For instance, the research in [1] describes a computing continuum system that concurrently executes tasks across multiple tiers: Cloud, Fog, Edge, and IoT. This highlights the need for modern systems to adopt hierarchical architectures with precise resource management to achieve high performance. Similarly, [2] addresses the challenge of diverse computational paradigms, proposing heterogeneous integration of modular approaches for scientific computing. Furthermore, the emerging field of federated learning introduces the task of coordinating edge resources. As [3] demonstrates, recent AI advancements rely on federated computing across heterogeneous cloud and high-performance computing (HPC) infrastructures, underscoring that effective resource collaboration is critical for building robust systems.

In summary, the core challenge we address is the demand for seamless resource integration:

1) Data integration requires unifying disparate data sources;
2) Computational integration demands efficient orchestration of heterogeneous paradigms;
3) Infrastructure integration necessitates optimal exploitation of physical devices.

II. Technologies

There are several technologies that should be described to achieve integrated system: Virtualization, Hierarchical architecture and Data management.

A. Virtualization

Virtualization demonstrates the ability of modern systems to abstract physical resources, thereby simplifying computational modeling. This capability enables the construction of high-performance computing (HPC) networks grounded in mathematical or physical models with well-defined scalability, latency, and connectivity properties.

Numerous virtualization approaches exist for diverse resources, including networks, CPU cores, NPUs, GPUs, and file systems. In distributed systems, the representation of nodes is critical. Two primary approaches address this:

- **Containers**: Lightweight, self-contained units sharing the host OS kernel.
- **Virtual Machines (VMs)**: Provide strong isolation through full hardware emulation but incur higher resource overhead.

Containerization, an increasingly prevalent paradigm, offers significant advantages for dynamic systems. Studies [4]–[6] demonstrate its benefits for high availability [4], operational efficiency [5], and heterogeneous edge computing [6]. However, VMs remain preferable for scenarios demanding strict security and stability.

Additionally, *logical abstraction* decoupling architecture from underlying heterogeneity. Thatprovides a unified view of distributed resources.

To address our requirements, the solution has to integrate diverse computing paradigms. This necessitates a system capable of adapting to heterogeneous environments, data, and resources. Consequently, we focus on containerized clusters to achieve the required agility.

B. Hierarchical Architecture

Hierarchical architectures organize systems into layered structures where each stratum exhibits heterogeneous characteristics. This approach enhances scalability and enables adaptation of domain-specific computational methods at individual layers, provided each layer operates as an independent module. Since the architecture defines inter-layer communication protocols, new protocols can be implemented through encapsulation of existing ones. However, hierarchical systems demand precise control: failures in layer synchronization, communication bottlenecks, or mapping/extraction delays propagate system-wide performance degradation.

https://doi.org/10.51408/csit2025_16

Hierarchical design is particularly essential in highly heterogeneous environments. The following example illustrates this necessity:

1) Grid Computing: Grid computing emerged as a global infrastructure for federated resource sharing across administrative domains, addressing the demand for worldwide distributed computing [7]. These architectures typically implement multi-layer computational frameworks for large-scale scientific experiments. Key characteristics include [7]:

- Integration of resources and users spanning disparate administrative domains;
- Standardized multi-purpose protocols for authentication, authorization, resource discovery, and access;
- Coordinated resource utilization to deliver configurable quality-of-service (QoS) levels.

Despite its utility, Grid computing remains largely task-specific. As demonstrated in [8], broader adoption requires not only architectural deployment but also *adaptive module specialization*—tailoring individual components to heterogeneous workloads. Future developments must focus on dynamic resource orchestration to enhance efficacy beyond niche scientific applications.

2) Edge Computing: Edge computing introduces extreme heterogeneity through resource-constrained mobile and IoT devices, leveraging their growing computational capabilities. This paradigm adopts a hierarchical structure—extending from cloud to fog (intermediate edge layer) and endpoint devices—to decentralize computation from centralized cloud infrastructures. While this offloads processing from core systems [9], it introduces additional round-trip latency due to multi-hop communication.

Despite latency trade-offs, hierarchical edge architectures can outperform flat (non-hierarchical) designs in resource efficiency and communication overhead when properly managed [10]. However, realizing these gains requires solving the *dynamic resource orchestration problem*: optimally distributing workloads across heterogeneous layers while balancing latency, energy consumption, and computational constraints. This remains an active research challenge, with emerging solutions focusing on adaptive scheduling and context-aware resource allocation.

3) Other paradigms: Several computing paradigms exemplify the power of layered architectures through strategic decomposition. Beyond grid computing, three prominent approaches demonstrate how hierarchical structuring enables complexity management:

- Microservices Architecture.
- Blockchain Architecture.
- High-Performance Computing.

In these approaches, layering provides critical *abstraction* (hiding implementation details), *modularity* (enabling independent component evolution), and *specialization* (optimizing each tier for specific functions)—collectively allowing complex systems to be managed, scaled, and maintained with precision. The consistent recurrence of this pattern across domains underscores hierarchy as a fundamental principle for engineering distributed systems.

C. Data Management in Distributed Systems

Data is fundamental to computational systems. While prior sections addressed computational paradigms, this part focuses on adapting data technologies for highly distributed environments. Evidence shows that increasing distribution and heterogeneity directly correlate with data management complexity [11].

To address these challenges, modern systems employ distributed file systems that abstract underlying computational nodes. Two dominant architectures exist:

1) **Data Warehouses**: Optimized for structured data stored on physical disks, accessed via SQL queries. Effective for transactional operations but ill-suited for Big Data scenarios characterized by high volume, velocity, and variety.
2) **Data Lakes**: Designed for unstructured/semi-structured data, enabling low-cost storage of diverse datasets. Their key advantage is storage that supports on-demand analytics.

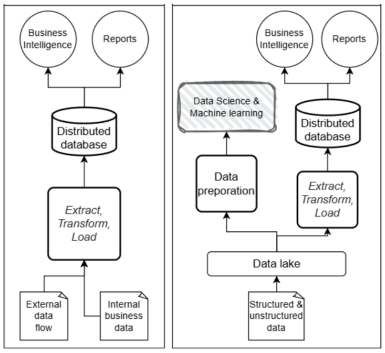

Fig. 1. Data-lake and Data-warehouse processes

Despite rapid adoption, data lakes face critical challenges:

- Integration of heterogeneous data lake implementations across administrative domains
- Scalability beyond petabyte-scale workloads
- Unified access interfaces spanning geographically distributed sites
- Ensuring data freshness and version consistency in dynamic environments

These limitations have driven data lake integration with cloud ecosystems. While cloud platforms offer partial solutions for resource heterogeneity, significant gaps remain in hybrid architectures. We argue that *hierarchical data management*—extending the architectural principles established in previous sections.

Two hierarchical approaches show promise:

1) **Multi-tiered physical storage**: Combining storage media (e.g., SSDs, HDDs, tape) with orchestration balancing access latency and cost. This approach is validated

in high-energy physics workflows at CERN, where the EOS storage system manages exabyte-scale data across hierarchical tiers [12].

2) **Compute-aware data placement**: Integrating data lakes with cloud/fog/edge hierarchies. By caching frequently accessed data at edge/fog layers (analogous to CPU cache hierarchies), latency-sensitive requests are served locally. However, this demands sophisticated metadata management to maintain consistency across layers—a challenge requiring adaptive orchestration mechanisms.

III. Mesh Architecture

In this section, we present the overall design of the unite system. As noted previously, modern computing demands accommodate multiple paradigms—each of which imposes its own requirements on data access and control. To reconcile these heterogeneous demands, we propose a multilayered, hierarchical architecture supporting virtualized computing resources.

Our architecture can be viewed as a high-throughput computing (HTC) "network of networks," where each managed cluster is optimized for high-performance computing (HPC) workloads. Beginning at the bottom, we organize the system into (*A*) a *computation layer* comprising worker nodes; (*B*) one or more *virtualization and local management layers* that abstract and coordinate resources; and (*C*) a unified *access layer* at the top, which provides a single entry point for users and applications.

A. Computational Layer

The computational layer consists of a set of heterogeneous physical devices (CPUs, GPUs, FPGAs, etc.) organized into worker-node clusters. These nodes execute computational jobs under a multi-paradigm framework; the selection of an appropriate paradigm for each task is delegated to the global and local managers. We assume that this selection—whether optimal or suboptimal—is provided by a higher-level scheduling policy.

To accommodate diverse hardware, we employ container-based virtualization as described in Section II-A. Containers enable isolation, portability, and rapid deployment, thereby mitigating the complexities of heterogeneous environments. For large-scale deployment, the system leverages cloud-native methodologies [13]: on-demand provisioning, namespace isolation, and policy-driven automation. Without extensive automation, achieving true elasticity and fault tolerance at scale is infeasible. Accordingly, our approach integrates container-orchestration tools to automate cluster deployment, scaling, and health-monitoring.

Data management in the computational layer is designed for stateless execution: individual worker nodes cache only the data needed for their current tasks. Persistent storage of shared datasets is delegated to the local manager tier, which maintains high-throughput caches and coordinates prefetching and replication. This division of responsibility minimizes I/O bottlenecks and ensures that data-intensive jobs can proceed with low latency.

B. Local Management Layer

The local management layer provides a unified, virtualized view of the underlying computational layer. Because worker nodes may differ in hardware, software stack, and supported programming models, this layer performs abstraction to simplify higher-level control and enable uniform resource management. To support this function, we maintain a metadata repository that records the capabilities, configuration, and current state of each node, thereby enabling protocol virtualization and dynamic adaptation.

Other responsibility of the local manager is to select the appropriate execution environment for tasks dispatched by the global scheduler. This requires implementation of customizable scheduling and autoscaling policies. Task descriptions—supplied by end users and annotated by the global manager—must include resource requirements (e.g., CPU cores, GPU types) and performance objectives (e.g., latency targets, throughput goals) to inform effective placement decisions.

Resource governance and automation are also critical at this layer. By interfacing with container-orchestration platforms and building bespoke automation workflows, the local manager can provision, update, and heal worker-node pools without manual intervention. This level of automation is essential for maintaining performance and reliability as the system scales.

Finally, the local management layer optimizes data locality through a hierarchical storage architecture. A data-placement engine continuously monitors access patterns and migrates or replicates data blocks to node-local caches as needed. While designing such a system is challenging, recent work has demonstrated the efficacy of reinforcement-learning–based policies for automated data-placement decisions [14], [15].

C. Global Management Layer

The global management layer serves as the single entry point for end users, abstracting all resource details and exposing a unified API for job submission and result retrieval. To enable opaque access, the layer maintains a comprehensive global metadata catalog that tracks resource availability, topology, and performance metrics across all local managers. This metadata must be highly available and partition-tolerant to support continuous operation under peak loads.

Given the volume of incoming job requests and result queries, the global layer must itself be a massively parallel, distributed service. We adopt a hybrid architecture that integrates proven supercomputing middleware with cloud-native scalability patterns [16]. Incoming requests are load-balanced across stateless front-end clusters, which orchestrate task dispatch to local managers and aggregate results upon completion.

Data orchestration at this level addresses both metadata traffic and bulk data movement. While metadata operations (e.g., scheduling decisions, status updates) benefit from low-latency key–value stores, application working data typically originates

from large-scale, Big Data repositories. A hierarchical storage strategy—similar to that described in [14], [15]—coordinates prefetching, caching, and replication to minimize end-to-end latency.

For resilience and maintainability, each component of the global management layer is designed as an independent, self-managing microservice. This decoupled approach avoids the pitfalls of monolithic designs, allowing modules to evolve, scale, and recover autonomously. Comprehensive automation—spanning deployment, scaling, health monitoring, and failover—is indispensable to prevent administrative bottlenecks and ensure continuous service availability.

From the user's standpoint, the entire stack appears as a logically virtualized compute fabric. Users submit job descriptions via a RESTful or gRPC-based API, specifying resource requirements and performance objectives. The global management layer then transparently selects the optimal execution environment—be it high-performance clusters, GPU pools, or on-demand cloud instances—and returns results through the same API interface.

IV. Conclusion

In this paper, we have presented the design of a hierarchical, virtualized computational system that unifies diverse computing paradigms under a single access point. Starting from low-level worker nodes, we introduced container-based virtualization and metadata-driven management to address hardware and software heterogeneity. We then detailed a two-tier control hierarchy—local managers for node-level scheduling, autoscaling, data placement, and a global manager for unified job submission, resource discovery, and result aggregation. Throughout, we emphasized automation, modularity, and a hierarchical storage architecture to optimize performance, reliability, and scalability.

Our proposed model reconciles the strengths of HTC, HPC, and cloud/fog/edge environments, enabling organizations to integrate disparate resources into a coherent, high-throughput platform. By decoupling components into independently deployable, self-managing services, the system avoids monolithic bottlenecks and supports continuous evolution and maintenance. Reinforcement-learning–based data-placement strategies further enhance data locality and reduce I/O latency.

References

[1] S. Dustdar, V. C. Pujol, P. K. Donta, "On Distributed Computing Continuum Systems", *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 4092–4105, 2023.

[2] F. J. Seinstra, B. Bal, D. H. J. Epema, "Jungle Computing: Distributed Supercomputing Beyond Clusters, Grids, and Clouds", *Grids, Clouds and Virtualization*, M. Cafaro and G. Aloisio (Eds.), Springer, London, UK, 2011.

[3] Z. Li et al., "Secure Federated Learning Across Heterogeneous Cloud and High-Performance Computing Resources: A Case Study on Federated Fine-Tuning of LLaMA 2", *Computing in Science & Engineering*, vol. 26, no. 03, pp. 52–58, 2024.

[4] S. Deochake, S. Maheshwari, R. De, A. Grover, "Comparative Study of Virtual Machines and Containers for DevOps Developers", [Online]. Available: https://arxiv.org/abs/1808.08192, 2018.

[5] W. Li, A. Kanso, "Comparing Containers versus Virtual Machines for Achieving High Availability", *2015 IEEE International Conference on Cloud Engineering*, Tempe, AZ, USA, pp. 353–358, 2015.

[6] H. Sturley, A. Fournier, A. Salcedo-Navarro, M. Garcia-Pineda, J. Segura-Garcia, "Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge", *Future Internet*, vol. 16, no. 11, 427, 2024.

[7] I. Foster, C. Kesselman, "The History of the Grid", [Online]. Available: https://arxiv.org/abs/2204.04312, 2022.

[8] I. Foster, C. Kesselman, "Computational Grids", [Online]. Available: https://arxiv.org/abs/2501.01316, 2025.

[9] M. Satyanarayanan, "The Emergence of Edge Computing", *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[10] L. Tong, Y. Li, W. Gao, "A hierarchical edge cloud architecture for mobile computing", *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, San Francisco, CA, USA, pp. 1–9, 2016.

[11] S. Azzabi, Z. Alfughi, A. Ouda, "Data Lakes: A Survey of Concepts and Architectures", *Computers*, vol. 13, 183, 2024.

[12] M. Afonso et al., "The CERN Tape Archive Beyond CERN An Open Source Data Archival System for HEP", *EPJ Web of Conferences*, vol. 295, 2024.

[13] A. Al-Said Ahmad, P. Andras, "Scalability Analysis Comparisons of Cloud-based Software Services", *Journal of Cloud Computing*, vol. 8, 2019.

[14] D. Vengerov, "A reinforcement learning framework for online data migration in hierarchical storage systems", *The Journal of Supercomputing*, vol. 43, pp. 1–19, 2008.

[15] T. Zhang, A. Hellander, S. Toor, "Efficient Hierarchical Storage Management Empowered by Reinforcement Learning", *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 6, pp. 5780–5793, 2023.

[16] I. Gankevich et al., "Constructing Virtual Private Supercomputer Using Virtualization and Cloud Technologies", *Computational Science and Its Applications – ICCSA 2014*, vol. 8584, pp. 341–354, 2014.