# Multilayer Computing Systems

Evgeniy Ibatullin(✉) and Alexander Bogdanov

St. Petersburg University, St Petersburg, Russia
officevg3n@gmail.com, a.v.bogdanov@spbu.ru

**Abstract.** The increasing complexity of computing and cloud systems, coupled with the growing demand for flexible and efficient solutions, has led to the emergence of efficient management need. This study investigates the potential for improving system manageability through virtualization approaches, with particular emphasis on the integration of systems based on layered abstractions. We examine the scalability challenges present in contemporary systems and argue that layered architecture represents a promising structural approach for addressing these issues. In particular, we focus on the primary limitation of large distributed systems—management complexity. To evaluate this problem, we pose the idea that abstraction layers can simplify system management while the performance overhead introduced by these abstraction mechanisms can be mitigated.

**Keywords:** Cloud management · Layered systems · Virtualization technologies · Distributed systems · Abstraction overhead

## 1 Introduction

### 1.1 Relevance of the Topic

The rapid growth of distributed systems has significantly increased the complexity of system management. This challenge is particularly evident in multi-cloud environments, which we use as a case study in this work, although the conclusions of our research are applicable to distributed systems in general. In such environments, providers expect highly precise and fine-grained management of individual resources. In practice, however, extremely large-scale systems—despite consisting of loosely coupled components—often introduce substantial latency and operational delays. A common approach to addressing the scalability of large systems is to decompose them into smaller, more manageable components. This principle aligns with the concept of *Separation of Concerns* introduced by Dijkstra, E. W. [1]. Also there is an idea of virtual infrastructure [2] that abstracts heterogeneity and simplifies management. Modern system design therefore increasingly adopts microservice or event-driven architectures. High-load systems are rarely implemented as monolithic structures due to their limited scalability and maintainability. Nevertheless, even these architectural approaches exhibit scaling limitations at extreme system sizes. We argue that contemporary

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16763, pp. 481–492, 2027.
https://doi.org/10.1007/978-3-032-30530-5_30

distributed systems require more hybrid architectural models, potentially including adaptive mechanisms capable of evolving in response to emerging operational challenges.

In this paper, we focus on layered architecture based on the following considerations:

1. Advantages:
   (a) **Separation of Concerns**—Each layer is responsible for a specific function. This division reduces system complexity and improves conceptual clarity.
   (b) **Abstraction**—Modifications within one layer typically do not affect other layers as long as the interfaces remain stable. This property allows internal architectures to evolve independently according to the capabilities and requirements of the layer maintainers.
   (c) **Standardization**—Layered architecture is a widely recognized design pattern. Its common structural model facilitates collaboration, maintenance.
2. Limitations:
   (a) **Performance Overhead**—Requests often traverse multiple layers even for simple operations, which can introduce additional latency.
   (b) **Rigid Structure**—Strict separation between layers may hinder cross-layer optimizations or specialized workflows that require tighter integration.
   (c) **Reduced Flexibility**—Operations that deviate from the standard layered flow, such as bypassing layers or integrating external systems, may require additional structural workarounds.

### 1.2 Literature Review

As noted in the previous section, cloud systems represent a primary example of large-scale distributed systems. This is particularly evident in multicloud environments, federated clouds, and sub-cloud paradigms such as edge computing. To evaluate the relevance of the management problem, we analyzed several recent publications to determine whether this issue is actively addressed in the literature.

Most of the analyzed works are survey papers that focus on current challenges and emerging trends in cloud computing. Figure 1 presents the most frequently mentioned trends identified in these studies, while Fig. 2 summarizes the challenges most commonly discussed by the authors. The analysis indicates that hybrid cloud architectures and resource allocation remain among the most prominent concerns in contemporary research.

We also highlight the study presented in [3], which provides a detailed analysis of potential architectural layers within modern cloud environments. The authors compare several computing paradigms—including fog computing, dew

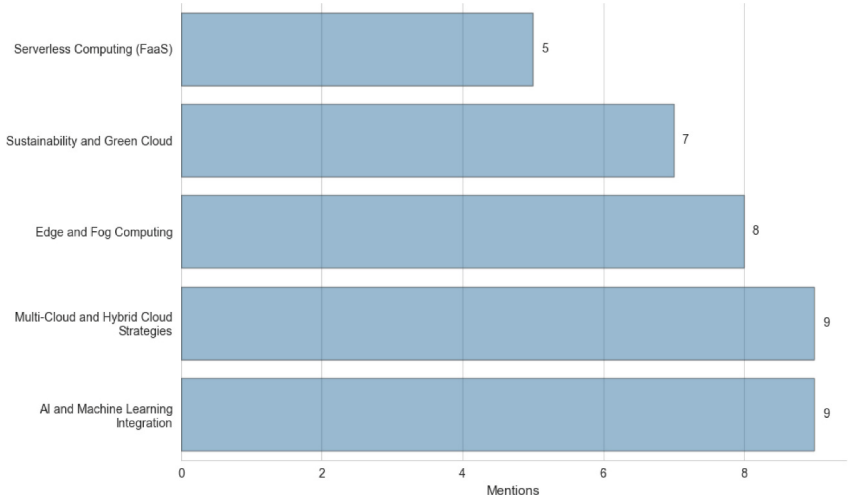


**Fig. 1.** Trends.

computing, and related models—using evaluation metrics such as latency, security, and manageability. Based on their comparative analysis, we constructed the representation shown in Fig. 3.

This representation highlights an important trade-off: introducing additional architectural layers can reduce end-user latency and improve the proximity of infrastructure to users, but it also increases the complexity of system management. In this context, the trade-off is largely determined by how closely infrastructure components can be positioned relative to end users.

More generally, this trade-off reflects a fundamental property of layered architectures. As systems scale, designers inevitably encounter a balance between improved modularity and the additional coordination required to manage multiple abstraction layers. The objective of this research is to mitigate the management complexity associated with layered architectures, thereby reducing the cost of adopting such scalable system designs.

## 2 Main Section

### 2.1 Virtualization Overview

Virtualization denotes an abstraction layer above physical hardware (and sometimes above logical components) that enables treating distributed resources as a cohesive, manageable whole. It can be applied to hardware, OS-level environments, and to interfaces that unify disparate systems.

**Primary Technologies**

– **Virtual machine (VM):** full emulation of a computing environment (hardware or OS) usually provided by a hypervisor. Strong isolation and OS independence; higher resource footprint and slower startup.

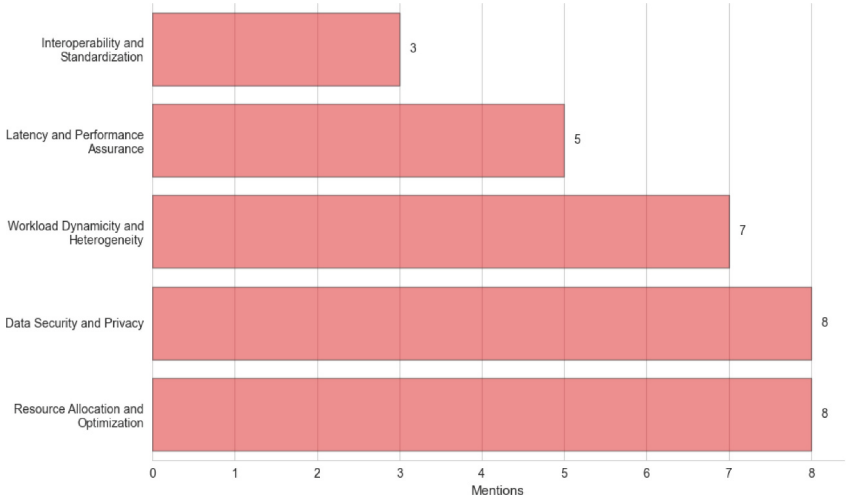


**Fig. 2.** Challenges.

– **Container:** lightweight isolation of applications using shared-kernel mechanisms. Faster startup and lower overhead than VMs; requires a container runtime.

Containerization and virtual machines have become standard solutions in modern distributed systems. These technologies can also be extended to more complex infrastructures, enabling consistent deployment and management across heterogeneous environments. Research [4] indicates that container-based environments provide an effective balance between flexibility and performance, identifying containers as a particularly suitable choice for distributed system implementations due to their relatively manageable overhead.

**Logical Virtualization**

– **Adapter/Facade (wrapper):** provides a uniform API over heterogeneous subsystems by adapting or hiding their differences. Useful for integrating components and presenting a stable abstraction to higher layers.
– **Plugin (pluggable interface) architecture:** exposes extension points and well-defined interfaces so independent modules can be added, replaced, or loaded at runtime. Enables extensibility and customization without changing core logic.

This logical or interface-level virtualization is commonly realized with design patterns. By implementing them we hide the system's heterogeneity.

### 2.2 Hierarchy Overview

Hierarchical models are commonly employed in environments where centralized control over the entire system is infeasible, necessitating delegation of manage-

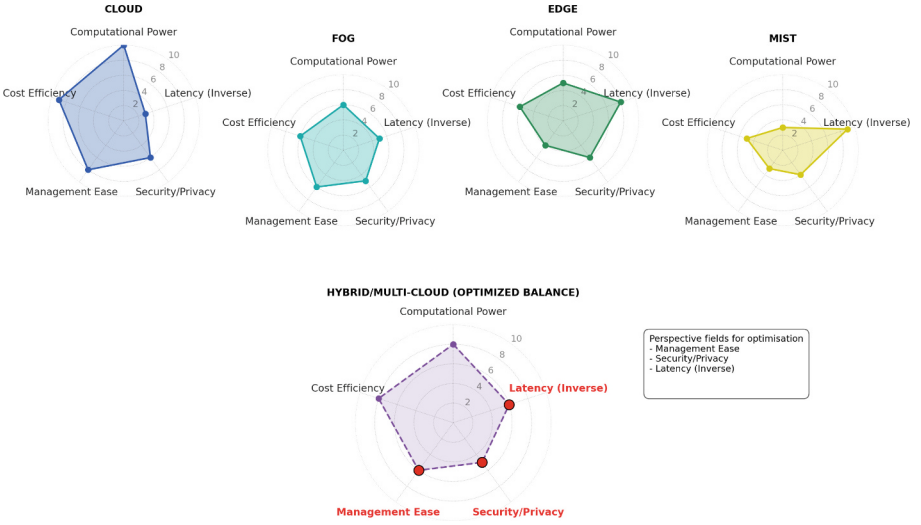


**Fig. 3.** Latency vs. manageability trade-off.

ment responsibilities to lower layers. Such approaches are typically unnecessary for simple systems and are introduced only when management complexity exceeds practical limits.

This work focuses on large-scale distributed systems as the primary target for abstraction. In general, a hierarchy represents a layered architecture in which higher-level components do not directly control lower-level ones, but instead influence system behavior through defined coordination and control mechanisms to achieve a desired global state.

Common hierarchical structures can be observed in storage systems (e.g., distributed file systems) and in computing architectures (e.g., fast versus slow processing layers). Modern applications, however, are often stateful, requiring a combination of both storage and computational considerations. In contrast, stateless systems primarily depend on computational scalability.

At a fundamental level, layers can be defined in two principal ways:

- **Logical decomposition:** the system is divided into functional blocks. This approach improves transparency and observability, as it clearly identifies which layer (or combination of layers) is responsible for specific operations.
- **Technical decomposition:** the system is partitioned according to technological concerns (e.g., storage, networking, computation). This enhances maintainability by isolating dependencies within well-defined boundaries.

In large-scale distributed systems, an additional criterion becomes critical: **manageability**. When traditional decomposition approaches are insufficient, systems must be structured to minimize management overhead, even at the cost of increased architectural complexity. This perspective assumes that system decomposition is necessary to maintain performance and operability at scale.

A layered system requires at least two levels. A common example is a dual-layer architecture consisting of a fast-access layer and a slower, persistent layer (e.g., in database systems). Upper layers typically provide higher performance at increased cost, while lower layers offer greater capacity and efficiency.

There is no universally optimal number of layers; the appropriate depth depends on system scale, workload characteristics, and management constraints. Excessive layering may introduce overhead, while insufficient layering can limit scalability and control.

### 2.3 Scaling Limits

In this section, we discuss why scaling in distributed systems has inherent limits. We refer to *Gustafson's law* [5] with communication latency, as modern systems typically evaluate performance and speedup in terms of the increased number of processed jobs.

Additionally, concepts such as *Metcalfe's law* [6] and *Dunbar's number* [7] illustrate the limitations of management and coordination, showing the challenges of scaling system control and the role of separation in overcoming these constraints.

**Computation Scaling.** The first aspect to consider is computational power. In the ideal case, near-linear scaling can be achieved according to Gustafson's law, provided that the task volume increases proportionally with available processing capacity. However, such conditions are rarely attainable in practice. Communication overhead and resource contention introduce diminishing returns as the system scales. These factors impose practical limits on achievable performance gains, even when computational resources continue to increase.

On Fig. 4 under conditions of efficient task utilization (parallel fraction of 0.3, latency factor of 0.1, and the number of communications approximated as $log_2 n$), a significant performance gap emerges due to latency overhead. From a cost-efficiency perspective, minimizing this gap is critical, as it represents unproductive resource utilization.

In this work, we propose that system decomposition should occur before the cost of such inefficiencies becomes dominant. Scaling within smaller subsystems is more efficient, as task allocation becomes more layer-specific. If computational resources are increasingly consumed by communication overhead, reallocating them toward improving system organization can yield more effective overall performance.

**Manageability Scaling.** Another key consideration in system decomposition is manageability. There are inherent limits to how much complexity can be effectively managed by humans, both in architectural design and infrastructure operation. This limitation is reflected in Fig. 5 as the blue curve, based on [7].

Automation is typically introduced to extend these limits. However, automation itself becomes insufficient when the system experiences exponential growth

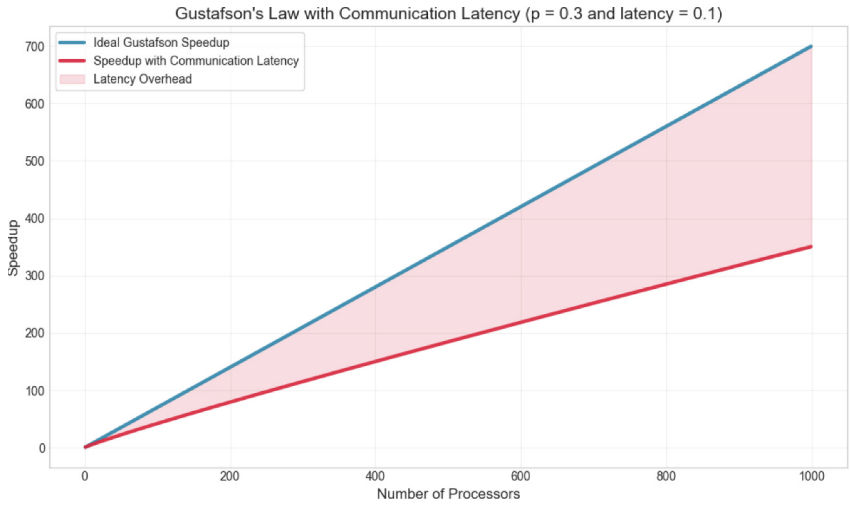


**Fig. 4.** Gustafson's law with latency.

in the number of nodes or communication channels. This effect is represented by the green curve in Fig. 5.

Therefore, for an efficient *layer split point*, it is necessary to identify the stage at which system manageability remains reliable, but any further increase in complexity would lead to an exponential growth in management difficulty.

### 2.4 Grid Case Study

A notable example of an intensively distributed system is Grid computing [8], originally developed to support large-scale scientific research. One prominent implementation is deployed at CERN. This system is organized into hierarchical tiers (or layers), each with distinct responsibilities and, in many cases, separate administrative domains. Even within a single tier, management may be distributed among multiple maintainers. Additionally, each layer is based on different technical principles, as illustrated in Fig. 6. This architecture provides a practical example of layered system design in a highly distributed context.

However, Grid systems have not achieved widespread adoption. One of the primary reasons is the complexity introduced by the tiered architecture. As the system scales, challenges arise in coordinating security, resource discovery, scheduling, monitoring, and data management across independent administrative domains. These limitations have contributed to the transition toward cloud computing as a more flexible and manageable paradigm.

Nevertheless, the core idea demonstrated by Grid computing remains relevant. Intensive distribution can be achieved through layered architectures, but modern implementations should leverage advances in cloud, edge, and fog computing to address the limitations observed in Grid systems and enable broader adoption of highly distributed infrastructures.

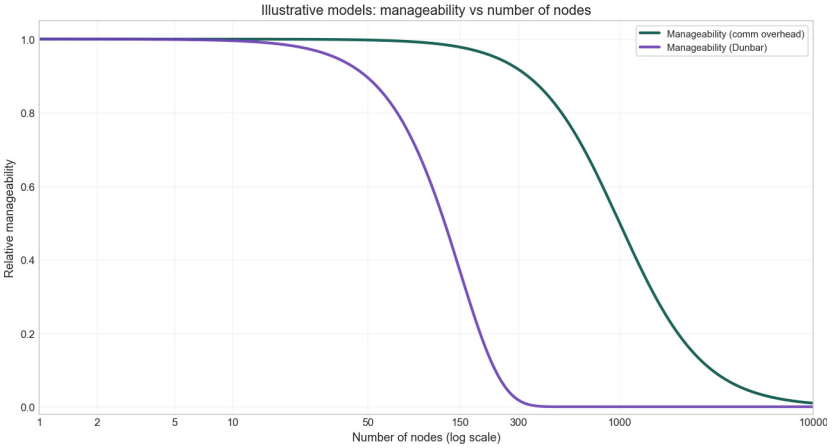


**Fig. 5.** Manageability.

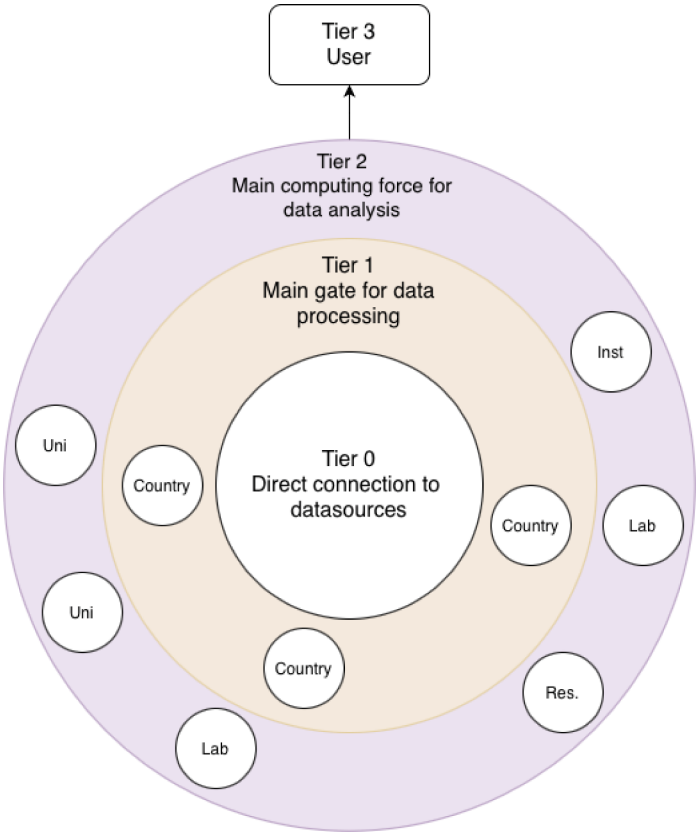


**Fig. 6.** Infrastructure at CERN.

### 2.5 Applications

We now outline the conditions under which such a system is expected to be effective, taking into account current technological developments. This approach is not intended for general-purpose distribution tasks. Instead, it is most suitable for systems designed around task- or logic-based distribution, where components and layers are highly decoupled.

Under these conditions, it becomes feasible to identify appropriate *split points* that simplify system management while introducing minimal latency overhead. In the following, we examine three primary problems that are strongly associated with layered architectures:

- AI federative learning
- Data storage systems
- Stateless computational systems

Those are primary cases for effective implementation.

**Federated Learning.** A current trend [9] in distributed systems is the decentralization of training for deep learning tasks, particularly for large language models. Three primary approaches are commonly used to distribute computation:

1. **Data Parallelism**—replication of the full model across multiple workers.
2. **Model Parallelism**—partitioning of the model across multiple devices.
3. **Federated Learning**—training across decentralized clients that retain local data.

A defining characteristic of federated learning is data heterogeneity. Because client data is decentralized, data samples held by each client may not be independently and identically distributed. Also the access to data is limited by security consideration. Also there could be a case there large datasets are used that could not be moved easily.

This study focuses on federated learning, as it naturally aligns with layered architectures. Such systems can be decomposed into hierarchical levels, typically represented as Cloud → Fog → Edge. On Fig. 7 we give the general idea on how to organize the system based on ≪device reliability≫ as core metrics for manageability.

These layers emerge from the need to process data generated at highly localized devices, where a large number of low-computation nodes operate at the edge. In this structure, management responsibilities are delegated to intermediate layers (e.g., fog), which coordinate aggregation and communication.

While the introduction of intermediate layers adds overhead, directly connecting edge devices to centralized infrastructure leads to increased network ingress, coordination complexity, and synchronization costs that grow with the number of clients.

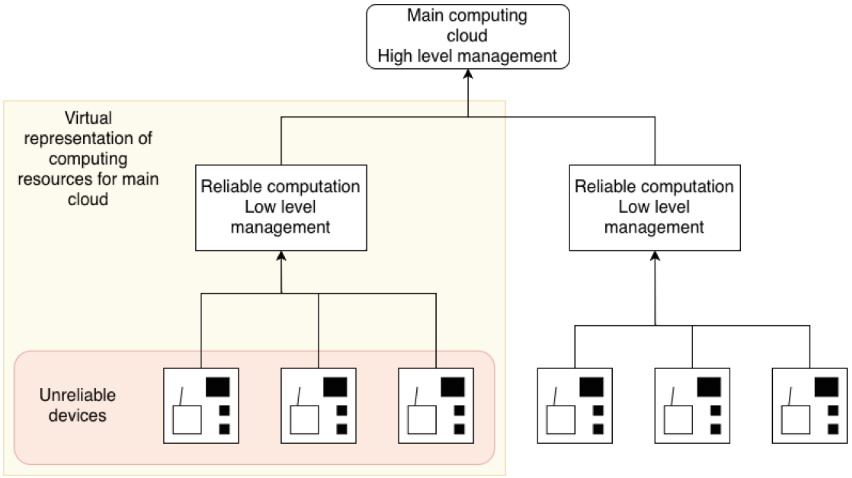


**Fig. 7.** Federated learning architecture.

**Storage Systems.** In storage systems, a fundamental trade-off exists between access time and storage cost [10]. On Fig. 8 we demonstrate system separation based on that metrics. Faster storage technologies are significantly more expensive, which leads to the common approach of organizing data into hierarchical layers, such as hot and cold storage. In this case, layered architecture arises naturally, driven primarily by logical and functional partitioning rather than by management scalability.

This approach becomes even more pronounced in the context of large-scale data systems, such as those encountered in Big Data applications. Here, system decomposition is often dictated by data locality. An upper orchestration layer coordinates multiple distributed storage units, each responsible for managing its own subset of data. These units are typically designed to be independent and self-managed, as the high degree of heterogeneity in such environments should remain encapsulated within localized domains.

Both examples demonstrate that, in data-intensive systems, the underlying data characteristics largely determine the system topology. As a result, architectural decisions must be made with careful consideration of these constraints. The most effective scenario occurs when system requirements naturally lead to a layered architecture, allowing this structure to simultaneously address management challenges.

**Computational Systems.** In computational systems, architectural constraints are less restrictive, as stateless applications can be deployed across a wide range of distributed topologies. However, as the system scales, logical or resource-based partitioning becomes necessary. In such cases, hierarchical structures provide a practical solution, enabling these partitions to improve local manageability while

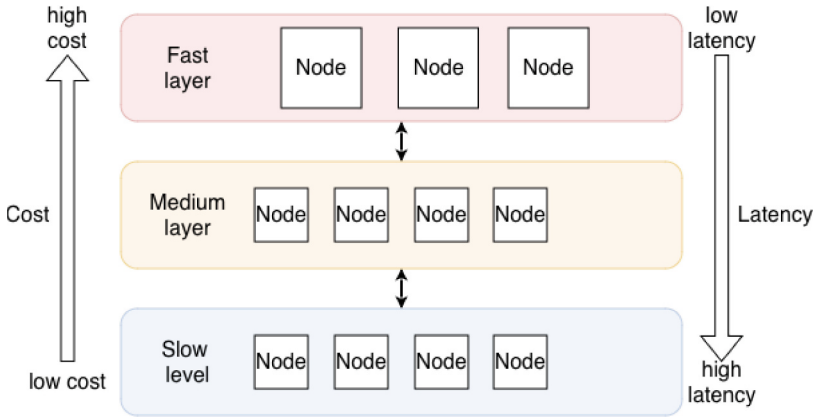


**Fig. 8.** Latency vs. cost trade-off.

maintaining overall system coherence. This, in turn, shifts the effective scaling limit.

This consideration is particularly relevant for geographically distributed resources. In such environments, a natural approach is to introduce management layers that oversee specific regions and delegate control responsibilities accordingly. This model exemplifies globally distributed systems, where hierarchical organization supports scalability by structuring management across multiple levels.

## 3 Conclusion

This work examines a potential approach to addressing the increasing complexity of continuously scaling computing systems. We argue that current trends lead to systems that encounter scaling limitations not due to insufficient computational resources, but due to the growing difficulty of managing heterogeneous and highly distributed infrastructures.

We have described the concept of hierarchical architecture as a means of achieving Separation of Concerns in such environments. Conceptual models were presented to highlight the existence of an effective *split point* for system decomposition, where the benefits of layering outweigh the associated overhead. Additionally, we discussed representative cases of highly distributed systems and outlined scenarios in which layered architectures can be effectively applied.

A central recommendation is to identify appropriate metrics that clearly define how a system can be partitioned. Once such metrics are established, they can guide the formation of functional or logical blocks that improve manageability without significantly degrading performance.

The applicability of this architectural approach depends on specific characteristics of real-world systems, many of which are abstracted in this study. Further

empirical evaluation is required for large-scale deployments. Nevertheless, the approach demonstrates potential for extending beyond localized systems and enabling more effective coordination across widely distributed infrastructures.

## References

1. Dijkstra, E.W.: On the role of scientific thought. In: Dijkstra, E.W. (ed.) Selected Writings on Computing: A Personal Perspective. Texts and Monographs in Computer Science. Springer, New York (1982). https://doi.org/10.1007/978-1-4612-5695-3_12
2. Korkhov, V., Kobyshev, S., Degtyarev, A., Bogdanov, A.: Light-weight cloud-based virtual computing infrastructure for distributed applications and Hadoop clusters. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10408, pp. 399–411. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62404-4_29
3. Mahapatra, A., Mishra, K., Pradhan, R., et al.: Next generation task offloading techniques in evolving computing paradigms: comparative analysis, current challenges, and future research perspectives. Arch. Comput. Methods Eng. **31**, 1405–1474 (2024)
4. Gankevich, I., et al.: Constructing virtual private supercomputer using virtualization and cloud technologies. In: Murgante, B., et al. (eds.) ICCSA 2014. LNCS, vol. 8584, pp. 341–354. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09153-2_26
5. Gustafson, J.L.: Reevaluating Amdahl's law. Commun. ACM **31**(5), 532–533 (1988). https://doi.org/10.1145/42411.42415
6. Metcalfe, R.M.: Metcalfe's law: a network becomes more valuable as it reaches more users. Presentation to the 3Com Sales Organization (1983)
7. Dunbar, R.I.M.: Neocortex size as a constraint on group size in primates. J. Hum. Evol. **22**, 469–493 (1992). https://doi.org/10.1016/0047-2484(92)90081-J
8. Foster, I., Kesselman, C.: The History of the Grid. In: High Performance Computing: From Grids and Clouds to Exascale. IOS Press, Amsterdam (2011)
9. Yao, Y.: Federated Large Language Models: Current Progress and Future Directions. arXiv:2409.15723 (2024)
10. Lucas Filho, E.R., Efstathiou, A., Yang, L., et al.: DITIS: an end-to-end system-level simulator and optimizer for distributed tiered storage. SN Comput. Sci. **6**, 746 (2025)