

Scale-Up-First: The Cloud as One Logical Computer

Vision

Daniel Barcelona-Pons
daniel.barcelona@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Pedro Garcia Lopez
pedro.garcia@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Stepan Klymonchuk
stepan.klymonchuk@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Abstract

Cloud systems have normalized scale-out-by-default: workloads are fragmented across nodes, accepting coordination, serialization, transport, and fault-handling overheads as the price of scalability. We argue that this default is increasingly wrong for stateful, communication-heavy workloads. TPC-H experiments show that distributed execution can require 4× the resources to match a strong single-machine baseline, with some queries still falling short.

We present a *scale-up-first* doctrine: keep stateful computation inside the largest practical shared-memory domain before crossing an application-visible network boundary. Larger shared-memory systems, chiplet hierarchies, and CXL-class memory expansion and pooling make this doctrine increasingly plausible. The stronger implication is a cloud resource model: *one logical computer*, where users submit processes and state instead of provisioning machines or clusters. The platform would internalize placement, recovery, elasticity, and fault handling within a hierarchical regional substrate. This is a systems vision and research agenda, not a claim that current platforms have reached that endpoint.

CCS Concepts

• Computer systems organization → Cloud computing; • Software and its engineering → Operating systems.

Keywords

Scale-up-first, stateful cloud systems, locality, shared-memory systems, cloud abstractions

ACM Reference Format:

Daniel Barcelona-Pons, Pedro Garcia Lopez, and Stepan Klymonchuk. 2026. Scale-Up-First: The Cloud as One Logical Computer: Vision. In *ACM Symposium on Cloud Computing V.2 (SoCC '26)*, November 18–20, 2026, Singapore, Singapore. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3828161.3857874>

1 Introduction

For more than a decade, cloud systems have normalized a doctrine: scale out first. When workloads grow, we add nodes; when demand spikes, we replicate services; when state becomes too large or active, we partition it across the network. This reflex delivered elasticity and operational modularity, but it also normalized a costly assumption: explicit distribution is the natural starting point for scalable

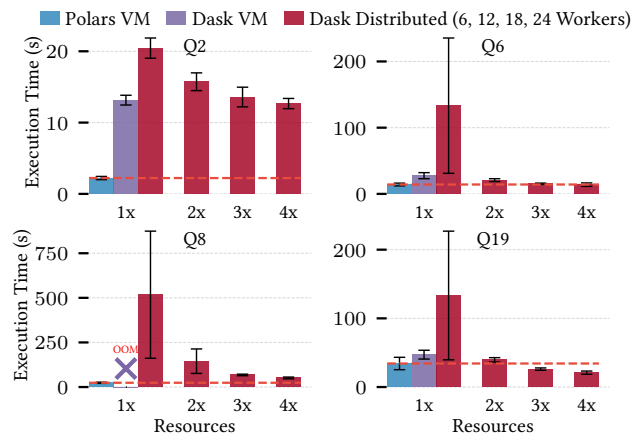


Figure 1: TPC-H query execution across local Polars, same-machine Dask, and distributed Dask. Local Dask helps distinguish framework overhead from the distributed-deployment penalty. Six workers match the baseline’s compute and memory budget; even 24 often do not match its performance.

computing. COST [18] challenged this assumption by asking not whether a distributed system can scale, but at what complexity and hardware cost. We push that question further: for stateful, communication-heavy workloads, how long can computation remain local before distribution becomes necessary?

We focus on workloads dominated by moving, synchronizing, or repeatedly touching shared state, such as analytics engines, databases, graph processing, stateful stream processors, and information management services. Their network tax can be steep. Serialization, shuffling, coordination, transport, and fault handling are often accepted as unavoidable, although they can be symptoms of distributing too early. Our TPC-H experiment illustrates this practical resource penalty: distributed Dask requires up to four times the nominal compute and memory budget to match an optimized local engine, and many queries remain slower within the evaluated range (Figure 1). Locality therefore challenges scale-out-by-default for stateful cloud computation.

These costs also matter for energy efficiency: datacenter electricity demand is large and rising [15, 25–27], and power provisioning and networks can be first-order efficiency constraints [1, 7, 11]. Avoidable network-mediated coordination is therefore likely to affect energy use as well as performance.

Concurrently, the practical boundary of a machine is expanding. Multi-socket systems, NUMA-aware software, chiplet-based packaging, and memory interconnects such as CXL are increasing the

Table 1: Scale out versus scale up for stateful cloud workloads.

Dimension	Scale-Out-First	Scale-Up-First
<i>Coordination</i>	Network-Mediated	Memory-Mediated
<i>Model</i>	Distributed Programming	Parallel Programming
<i>Domain</i>	Multiple Nodes in Clusters	Single Large Machine

compute and memory that systems can manage without application-visible network RPCs [6, 17, 20]. The shared-memory execution domain—the application-visible address-space boundary within which state is accessed through load/store semantics—is expanding. However, the substrate may be non-uniform: CXL supports coherent device attachment and memory expansion or pooling, yet pooling alone does not create rack-wide CPU cache coherence. We do not expect flat rack- or datacenter-wide coherence in the near term, but we argue that larger coherent hosts and adjacent memory-semantic tiers can absorb more stateful computation before explicit distribution becomes the better trade.

These developments support *scale-up-first*: keep stateful computation inside the largest practical shared-memory execution domain before crossing the network. Distribution should justify itself rather than be the default. Contention remains a serious systems problem, but it can be more tractable than prematurely paying the network tax of distribution. Operating systems, runtimes, and middleware must therefore manage locality, isolation, recovery, and elasticity inside larger shared-memory domains.

The stronger implication is architectural: *one logical computer* may become the primary abstraction for stateful computation (Table 1). Today, programmers often approach services as clusters: choosing nodes, partitioning state, defining network boundaries, and incorporating RPCs, shuffles, and distributed recovery into the application. One logical computer reverses that default. Programmers target a shared-memory environment of processes, threads, memory, files, and service endpoints; the platform manages placement, recovery, elasticity, and internal topology below the application boundary.

This paper contributes two claims: a scale-up-first doctrine for stateful systems, and a long-term resource model in which users run processes and state directly on the cloud rather than provision computers. The contribution is this cloud-level design vision, not a new interconnect, coherence mechanism, or single-system-image implementation. A diagnostic experiment and hardware trends motivate the vision and its research agenda. We first examine the empirical cost of early distribution.

2 Empirical Evidence: The Hardware Cost of Premature Distribution

We quantify the scale-out burden as the multiple of distributed resources required to match an optimized single-machine baseline. Following COST [18], scalability must account for the resources and complexity consumed to achieve it.

We evaluate TPC-H at scale factor 300 (74.4 GiB of compressed Parquet data) through three configurations: Polars on one machine

as a strong local baseline, Dask on that same machine as a framework control, and Dask across Fargate workers as a practical distributed deployment. The baseline EC2 instance (c7a.24xlarge) has 96 vCPUs and 192 GiB of RAM. Each Fargate worker has 16 vCPUs and 32 GiB of RAM over a VPC. We scale from 6 workers (matching the baseline’s compute and memory budget) to 24 (4×). Same-machine Dask helps distinguish framework overhead from the additional distributed-deployment penalty. Although the comparison does not isolate the cost of distribution entirely, the results are a diagnostic signal of the practical resource penalty of a cluster-first stack.

Table 2 summarizes the outcomes, with selected queries in Figure 1. For this deployment, the penalty is severe: eight of the twenty-two queries match the baseline with 2× or 3× its resource budget, three require 4×, and ten remain slower even at that point.

Query 1 exhausts memory in the tested Polars configuration, whereas distributed Dask completes it. A larger local machine might also succeed, but workloads will eventually exceed the practical shared-memory domain. At that boundary, aggregating distributed resources and paying the network tax becomes the right choice, as the scale-up-first doctrine explicitly allows.

When a typical distributed stack requires up to 4× the resources to match a strong local baseline, scale-out can no longer be accepted as the neutral starting point for stateful systems. Not all workloads belong inside one machine, but the burden of proof shifts: how far can a workload scale locally before incurring the network tax? A distributed architecture can outlast the capacity constraint that originally justified it. A workload that required several nodes may later fit within one local execution domain, yet retain the partitioning and coordination costs of its original design. Emerging hardware suggests that this local boundary is expanding.

3 Why Now: The Machine Boundary Is Expanding

The case for revisiting scale-up follows from a material change in machine capacity. AWS offers a high-memory instance with 1,920 vCPUs and 32 TiB of RAM, while general-purpose scale-up instances reach 384 vCPUs and 1.5 TiB [3, 4]. Intel Xeon 6+ reaches 288 cores per socket and AMD’s 5th-generation EPYC reaches 192, with denser successors on the roadmap [2, 14]. NVIDIA’s GB200 NVL72 packages 36 Grace CPUs with 17 TB of aggregate LPDDR5X and 72 GPUs in a rack-wide NVLink domain with 13.4 TB of aggregate HBM3E [21]. Its main memory is not a unified coherent domain, but GB200 illustrates tight, hierarchical rack-scale integration. Useful locality domains are growing from multicore servers toward rack-scale macro-machines, although without uniform latency or freedom from contention.

The surrounding hardware stack is accelerating this trajectory. Chiplet-based processors, high-bandwidth memory, NUMA-aware systems, and memory-semantic interconnects such as CXL enlarge the capacity and topology accessible without conventional application-level networking [6, 10, 12, 17, 20]. CXL can provide coherent processor–device attachment and expose expanded, tiered, or pooled memory through load/store interfaces. These capabilities do not imply full cache coherence among all hosts sharing a fabric, but the trend hints towards increasingly hierarchical machine-like

Table 2: The resource penalty of distribution. For 45% of the 22 TPC-H queries, distributed Dask does not match the single-machine Polars baseline even with four times its resource budget.

Smallest Distributed Resources Required to Match Baseline	Query Count (total = 22)	TPC-H Queries	Systems Implication
Polars OOMs	1 (4.5%)	Q1	The tested local configuration exhausts memory; distributed Dask completes the query.
2× baseline	3 (13.6%)	Q13, Q14, Q18	Minor penalty; distributed Dask matches the local baseline with twice the resources.
3× baseline	5 (22.7%)	Q3, Q12, Q15, Q19, Q21	Moderate penalty; matching the baseline requires substantial additional resources.
4× baseline	3 (13.6%)	Q4, Q6, Q10	Severe penalty; distribution requires quadruple resources to match the baseline.
Never catches up by 4×	10 (45.5%)	Q2, Q5, Q7, Q8, Q9, Q11, Q16, Q17, Q20, Q22	Scale-out fails to match local performance within the evaluated range.

domains that combine coherent regions with more distant memory-semantic tiers managed by hardware and system software.

This shift has been visible for years in adjacent architecture and systems research. *The Datacenter as a Computer* treated warehouse-scale infrastructure as a single design object [9]. Factored and multikernel operating systems, such as fos and Barrelfish, reworked the software stack for hardware that already behaved like a distributed system internally [22, 29]. Scale-out NUMA and FireBox extended this concept at the architectural level by exploring low-latency domains that blur the distinction between scale-up and scale-out [6, 20]. While early software-based efforts like Kerrighed [19] pursued an enlarged resource domain but struggled against the latency of standard networks, later hardware-centric architectures such as LegoOS and dReDBox successfully decoupled and expanded the resources visible to software [16, 24]. Taken together, these lines of work demonstrate that the machine-cluster boundary has been eroding for some time; what has changed is that this hardware reality is now crossing into mainstream cloud design.

These systems are hierarchical, locality-sensitive, and heterogeneous. Remote memory remains slower than local memory; coherence, bandwidth, and placement still constrain performance [30]. The plausible near-term outcome is larger shared-memory domains supported by memory-semantic tiers, yet not datacenter-wide uniformity. Such domains may retain an advantage over application-visible RPCs despite internal distance and modularity. The question is whether that advantage warrants deferring explicit distribution.

Scale-up thus describes a hierarchical shared-memory substrate, not a monolithic motherboard pushed past its limits. Its complexity is managed below the application boundary. The next question is whether cloud software should default to network-mediated coordination or first exploit these widening locality domains.

4 Scale-Up-First as a Cloud Design Doctrine

Empirical evidence and hardware trends motivate *scale-up-first*: keep shared state within the largest practical application-visible shared-memory environment while it preserves useful locality. This is not simply a preference for big machines: introduce explicit distribution when its benefits exceed the cost of crossing the network boundary.

This doctrine is strongest for workloads whose dominant cost is moving, synchronizing, or repeatedly touching shared state. Shared memory, process placement, and locality-aware scheduling can avoid serialization and application-level RPCs for internal coordination. Table 2 illustrates the resource penalty of a distributed stack. Once the machine boundary starts expanding again, the right question is no longer whether distribution can scale in principle, but whether it is being invoked prematurely for workloads whose dominant cost is still coordinating state.

The standard objection is that scale-up exchanges one set of bottlenecks for another. This is correct but incomplete. As shared-memory domains grow, coherence traffic, NUMA distance, memory bandwidth, and kernel contention impose a *contention tax*. Distributed systems instead pay a *network tax*: serialization, protocol overhead, shuffle, stragglers, retries, and software-managed RPCs. This difference changes the programming model, failure model, and minimum cost of accessing state across boundaries. For many stateful workloads, managing contention within a locality-aware hierarchy remains preferable to paying the network tax by default. Scale-up-first is therefore a decision rule: remain local while the total cost of contention, capacity, reliability, and hardware remains below the communication and operational cost of distribution for the required service objectives.

This crossover is economic as well as technical. Large multi-socket systems can command a nonlinear price premium, require spare capacity for elasticity, and be unavailable in some regions. More cores only help when memory capacity and bandwidth grow appropriately; bandwidth saturation and tenant interference can undermine throughput and tail latency even when individual accesses remain fast. Hardware developments may move the crossover outward, but do not eliminate it. The goal is to find the largest *practical* local domain for a workload, rather than select the largest machine regardless of cost or predictability.

Here our doctrine departs from adjacent research. COST established the hidden overheads of scale-out [18], but did not turn that warning into a cloud design doctrine. Hybrid scaling systems such as Themis balance vertical and horizontal reactions under demand [23]; we argue that verticality should be the default for stateful computation and distribution should justify itself

rather than being assumed. Disaggregation work, including LegoOS, dReDBox, and Serverless End Game [13, 16, 24], composes remote resources behind a transparency layer. We instead seek tighter, locality-preserving logical integration into a larger local shared-memory domain, minimizing remote composition. Burst computing [8] similarly packs collaborative work into fewer domains to preserve locality; scale-up-first generalizes this instinct into a cloud-wide doctrine for stateful computation.

Under scale-up-first, cloud software stops treating the network as the natural home of state. Large shared-memory domains become the primary unit of computation, while explicit distribution moves outward to edge interaction, geo-replication, or workloads for which capacity, cost, or reliability favors crossing the local boundary. The remaining challenge is the application-facing abstraction. If the platform manages placement, recovery, and scale across a larger domain, programmers should not be forced to think in terms of a cluster first.

5 The Emerging Cloud Abstraction

If scale-up-first is the doctrine to guide current deployments, *one logical computer* is the corresponding longer-term cloud abstraction. For a broad class of stateful workloads, the cloud should present one shared-memory execution environment rather than a cluster of nodes as the default target of programming and deployment. The programming model therefore shifts as well: from nodes, containers, partitions, and RPCs toward processes, threads, and shared memory. As illustrated in Figure 2, that abstraction separates concerns across three layers: the user-visible interface, the machine-internal management layer, and the underlying hierarchical substrate.

Applications execute in one environment with shared and persistent state, direct memory access, and scheduler-managed placement. Resources can be assigned and resized during execution, much as an operating system manages processes. This environment maps onto a distance-aware topology of chiplets, sockets, memory pools, and rack-scale resources, preserving shared-memory access below the programming boundary.

5.1 User-Visible Logical Abstraction

As depicted in the top layer of Figure 2, the user-visible interface stays close to the process model programmers already understand. A tenant receives one or more isolated execution contexts with memory for shared state, persistent storage interfaces for files and objects, and network-facing service endpoints. The platform grants elastic access to compute and memory within this bounded domain, but the domain still feels like one machine. The application need not partition state across nodes, orchestrate RPC boundaries for internal parallelism, or implement distributed coordination unless the workload exceeds the local domain or must cross explicit geographic or fault-isolation boundaries.

The AWS Lambda MicroVMs service already points in this direction. Through a managed serverless interface, it combines VM-like isolation, snapshot-based suspend and resume, state retention while idle, network-facing execution, and vertical resizing [5, 28]. Although not the complete abstraction proposed here, it shows mainstream cloud interfaces moving beyond stateless functions toward stateful, longer-lived, and vertically elastic execution units.

5.2 The Hierarchical Substrate

The hardware substrate beneath this interface is not monolithic. It has internal distance, hierarchy, and heterogeneity, but it is meant to behave as one execution domain rather than as a cluster the application must manage explicitly. Some memory is physically closer, some farther; some cores share caches, while others cross sockets or fabrics; some resources may be pooled rather than strictly attached to local buses.

Crucially, the design goal is logical reintegration, not remote composition for its own sake. The system must navigate multiple tiers of physical locality: intra-socket chiplet fabrics, coherent socket-to-socket interconnects, memory-semantic tiers, and rack-scale links. In that setting, coherence and memory-semantic communication matter because they preserve load/store-style access across physical boundaries that would otherwise become explicit software boundaries. This is the key contrast with conventional disaggregation. The point is not to expose more remote parts behind a transparency layer, but to enlarge the practical local domain so the application can stay inside one logical computer longer. What matters is not hardware uniformity, but whether the system can manage these differences without turning them into application-level distribution.

5.3 Machine-Internal Management Layer

To maintain the single-machine illusion over this substrate, the intermediate management layer absorbs operational complexity through three core functions. First, *locality-aware placement* moves threads and data toward favorable hardware regions, with admission control and resource isolation limiting tenant interference. Second, *resilience and recovery* mechanisms—e.g., checkpointing, migration, and rapid restart—handle internal failures below the runtime boundary. Third, *elastic resizing* expands or contracts execution within the machine-like substrate (scale up) before introducing explicit fan-out across nodes (scale out).

The operating system and runtime must therefore reclaim locality, isolation, recovery, and elasticity from applications, and determine when the local domain is exhausted. Moving these responsibilities below the application boundary makes one logical computer viable without requiring a distributed application architecture from the outset.

5.4 Failure and Boundary Conditions

This abstraction requires a fault contract. Inside one logical computer, the goal is that failures of compute, memory, files, and local service endpoints are handled below the application boundary. To the programmer, the environment should behave like an operating system that does not go down: processes appear continuous even as components fail, restart, migrate, or are replaced underneath. Delivering that illusion requires redundancy, checkpointing, replication, migration, and hardware-assisted containment, with stated bounds on recovery time and recoverable state.

Not every workload belongs inside this domain. As indicated by the outward boundary in Figure 2, WAN/edge-facing services, geo-replicated applications, and computations whose scale or regulatory requirements exceed the local domain still require explicit distribution. Regional failures require protection beyond that domain. Geo-replication may remain a managed platform service, but

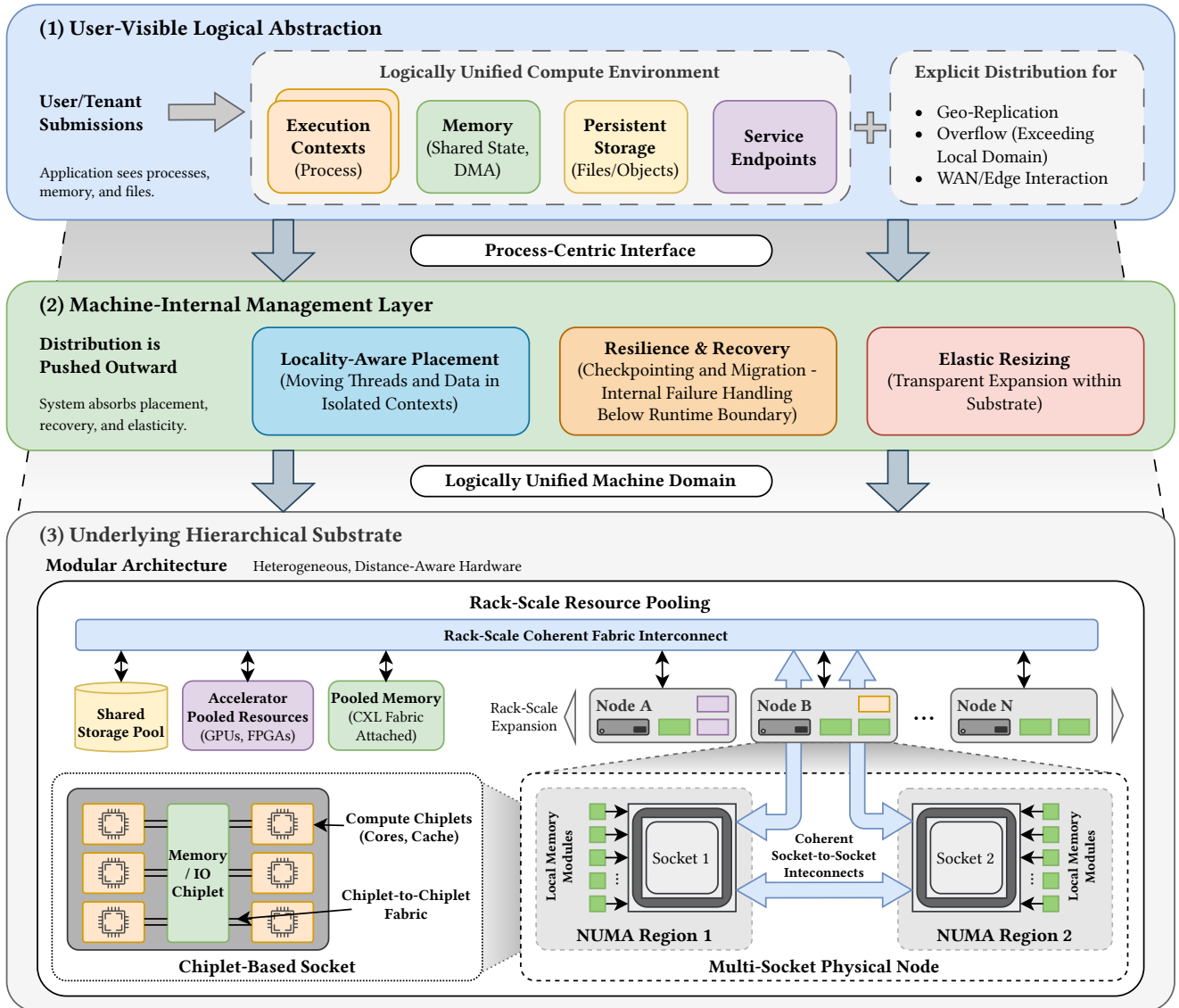


Figure 2: The cloud as one logical computer. A process-centric cloud abstraction over a hierarchical machine-like substrate.

its consistency, latency, and failover semantics must be visible to applications that depend on them. Once the application itself manages those external boundaries, it also resumes responsibility for the corresponding fault-tolerance logic, just as it resumes responsibility for explicit distribution.

Thus, the abstraction does not seek to eliminate distributed systems; it seeks to relocate them outward. Network-mediated distribution should become the deliberate fallback for cases that truly require it, rather than the undisputed starting point for all scalable computation. This boundary defines the horizon of our systems vision. If the application-facing abstraction continues moving in this direction, the cloud will increasingly function not as a federation of explicitly managed nodes, but as one logical computer.

6 The Cloud as a Logical Computer

The abstraction above could be adopted as a machine-like environment provisioned for each application. The longer-term step is more fundamental: users would stop provisioning computers. Instead of selecting VMs or constructing a cluster, they would submit processes, state, and endpoints directly to a common cloud substrate. That substrate would be modular, elastic, replicated, and self-repairing. Isolation would protect tenants and applications without imposing a fixed topology or capacity envelope on each deployment.

The cloud would then cease to be a store of computers and become the computer. Compute and memory could expand, contract, and move beneath running processes; placement, networking,

and ordinary component failures would become platform concerns. VMs, node counts, subnets, and failure domains might remain essential to providers, but disappear from the application interface. Scale up and scale out would likewise become internal mechanisms rather than architectures chosen in advance. This goes beyond treating the datacenter as a computer for infrastructure design: it presents the computer itself to cloud applications [9].

This endpoint does not require flat datacenter-wide coherence. A hierarchy of machine-like domains could preserve the interface while the platform moves computation and state within and between them. One plausible organization would place large stateful processes at the core, surrounded by a distributed periphery of mostly stateless components for ingress, caching, edge interaction, and geo-replication. Explicit distribution would remain at these inherently distributed boundaries. The long-term claim is therefore precise: for stateful computation within those boundaries, physical distribution should stop determining application structure.

This raises a sharper question. The COST paper [18] quotes: “You can have a second computer once you’ve shown you know how to use the first one.” If the default machine keeps getting bigger, which mainstream stateful services and applications still need the second computer at all?

Our hypothesis is that most stateful applications could execute within one regional logical domain, with explicit distribution reserved primarily for geographic replication, edge interaction, and workloads beyond its practical limits. Establishing how broadly this holds is part of the research agenda.

6.1 What We Are and Are Not Claiming

We do not claim that all workloads fit within one shared-memory domain, or that one global scheduler can govern the entire cloud. Cross-region services, edge interaction, globally replicated state, regulatory isolation, and workloads beyond the largest practical domain still require explicit distribution. Nor does abstraction eliminate physical costs. Applications need ways to specify performance, cost, placement, and reliability requirements, and operators need enough visibility to diagnose NUMA effects, contention, and failures. Our claim is about the default: expose physical topology only when it affects application semantics or explicit objectives, rather than making it the starting point of every stateful design.

6.2 Research Agenda

The question posed above—which applications still need the second computer?—defines the central research problem and requires work along five connected directions.

Where is the boundary of the macro-machine? The boundary should not be fixed by today’s server or rack. It will move as hardware expands shared-memory domains, and it may also move at runtime with load, contention, locality, bandwidth, failures, and cost. Schedulers therefore need criteria that compare the contention tax of staying inside the macro-machine with the network tax of crossing its boundary, and decide when a workload genuinely needs the second computer.

Which workloads benefit? Stateful, communication-intensive applications are the strongest candidates, but they are not a uniform

class. Databases, analytics, graph processing, and stream processing differ in their access patterns and fault-tolerance requirements. Conversely, some nominally stateless workloads may benefit from faster process creation, shared caches, or tighter communication. Comparative evidence must establish where scale-up-first wins and where explicit sharding or replication remains necessary.

What programming model and cloud interface follow? If coordination remains within one execution environment, parallel programming may replace distributed programming as the default model for cloud-side state. A process-centric interface must support elastic compute and memory, persistent state, and service endpoints without requiring VM sizes, node counts, or network layouts. It must hide incidental topology while exposing constraints that affect correctness, performance, reliability, and price.

Where do networked-systems functions and middleware move? Placement, replication, coordination, recovery, and elasticity are now divided among applications, middleware, and cloud services. One logical computer pushes many of these functions toward cloud operating systems and runtimes. Research must determine which functions should become machine-internal mechanisms, which middleware abstractions remain useful, and which responsibilities must stay with applications because they cross geographic or administrative boundaries.

How can one logical computer remain isolated and continuously available? A larger shared domain must not become one large failure or interference domain. The platform must contain hardware and software faults, isolate tenants, control contention, and recover or migrate stateful processes while minimizing application interruptions. This requires a precise contract for transparent recovery: which failures the logical computer absorbs, what state it preserves, how long recovery can take, and which exceptional conditions remain visible. Eliminating interruptions within that contract is the long-term research goal.

These questions connect scale-up-first to a concrete agenda across operating systems, networked systems, runtimes, cloud platforms, and hardware. The goal is not to eliminate distributed systems, but to move distribution below the application boundary when it is an implementation concern and expose it only when it is fundamental. The future cloud may scale by becoming more like one computer, rather than by requiring every application to become a distributed system.

Acknowledgments

This work is part of project PID2023-148202OB-C21, funded by MICIU/AEI/10.13039/501100011033 and by FEDER/UE.

The authors utilized ChatGPT and Gemini to assist with literature review and to refine the drafting of this manuscript. The authors have independently verified all references, reviewed and edited all AI-assisted text, and take full responsibility for the originality, veracity, and integrity of the final work.

References

- [1] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. 2010. Energy proportional datacenter networks. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (Saint-Malo, France) (ISCA '10)*. Association for Computing Machinery, New York, NY, USA, 338–347. doi:10.1145/1815961.1816004

- [2] Advanced Micro Devices, Inc. 2026. 5th Gen AMD EPYC Server Processors (Turin and Zen 5c Architecture). AMD Product Website. Accessed 2026-07-09. <https://www.amd.com/en/products/processors/server/epyc.html>
- [3] Amazon Web Services. 2026. Amazon EC2 High Memory U7i Instances. AWS EC2 Documentation. Accessed 2026-07-09. <https://aws.amazon.com/ec2/instance-types/u7i/>
- [4] Amazon Web Services. 2026. Amazon EC2 M8i Instances Powered by Intel Xeon 6. AWS EC2 Documentation. Accessed 2026-07-09. <https://aws.amazon.com/ec2/instance-types/>
- [5] Amazon Web Services. 2026. MicroVM images - AWS Lambda. AWS Lambda Documentation. Documentation page describing snapshot-based resume and baseline-to-peak sizing; accessed 2026-07-04. <https://docs.aws.amazon.com/lambda/latest/dg/microvms-images.html>
- [6] Krste Asanović. 2014. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. Presentation at the 12th USENIX Conference on File and Storage Technologies (FAST 14), Santa Clara, CA. <https://www.usenix.org/node/179918>.
- [7] Jayant Baliga, Robert W. A. Ayre, Kerry Hinton, and Rodney S. Tucker. 2011. Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport. *Proc. IEEE* 99, 1 (2011), 149–167. doi:10.1109/JPROC.2010.2060451
- [8] Daniel Barcelona-Pons, Aitor Arjona, Pedro García-López, Enrique Molina-Giménez, and Stepan Klymonchuk. 2025. Burst Computing: Quick, Sudden, Massively Parallel Processing on Serverless Resources. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. USENIX Association, Boston, MA, 39–56. <https://www.usenix.org/conference/atc25/presentation/barcelona-pons>
- [9] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. 2018. *The Data-center as a Computer: Designing Warehouse-Scale Machines* (3 ed.). Morgan & Claypool Publishers, San Rafael, CA, USA. doi:10.2200/S00874ED3V01Y201809CAC046
- [10] Compute Express Link Consortium. 2026. Compute Express Link (CXL). CXL Consortium Website. Accessed 2026-07-09. <https://computeexpresslink.org/>
- [11] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture* (San Diego, California, USA) (ISCA '07). Association for Computing Machinery, New York, NY, USA, 13–23. doi:10.1145/1250662.1250665
- [12] Alessandro Fogli, Bo Zhao, Peter Pietzuch, and Jana Giceva. 2026. CHARM: Chiplet Heterogeneity-Aware Runtime Mapping System. In *Proceedings of the 21st European Conference on Computer Systems* (McEwan Hall/The University of Edinburgh, Edinburgh, Scotland UK) (EUROSYS '26). Association for Computing Machinery, New York, NY, USA, 2308–2323. doi:10.1145/3767295.3769390
- [13] Pedro Garcia Lopez, Aleksander Slominski, Bernard Metzler, Michael Berhendt, and Simon Shillaker. 2024. Serverless End Game: Disaggregation enabling Transparency. In *Proceedings of the 2nd Workshop on SErverless Systems, Applications and MEthodologies* (Athens, Greece) (SESAME '24). Association for Computing Machinery, New York, NY, USA, 9–14. doi:10.1145/3642977.3652094
- [14] Intel Corporation. 2026. Intel Xeon 6+ Processors. Intel Product Website. Accessed 2026-07-09. <https://www.intel.com/content/www/us/en/products/details/processors/xeon/6-plus-series.html>
- [15] George Kamiya and Paolo Bertoldi. 2024. *Energy Consumption in Data Centres and Broadband Communication Networks in the EU*. Technical Report JRC135926. Publications Office of the European Union. doi:10.2760/706491
- [16] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. 2016. Rack-scale disaggregated cloud data centers: The dReDBox project vision. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Piscataway, NJ, USA, 690–695.
- [17] Alberto Lerner and Gustavo Alonso. 2024. CXL and the Return of Scale-Up Database Engines. *Proc. VLDB Endow.* 17, 10 (June 2024), 2568–2575. doi:10.14778/3675034.3675047
- [18] Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! But at What COST?. In *Proceedings of the 15th USENIX Workshop on Hot Topics in Operating Systems (HotOS)*. USENIX Association, Kartause Ittingen, Switzerland, 6 pages. <https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry>
- [19] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, Gaël Utard, R. Badrinath, and Louis Rilling. 2003. Kerrighed: A Single System Image Cluster Operating System for High Performance Computing. In *Euro-Par 2003 Parallel Processing*, Harald Kosch, László Böszörményi, and Hermann Hellwagner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1291–1294.
- [20] Stanko Novaković, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. 2014. Scale-out NUMA. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) (ASPLOS '14). Association for Computing Machinery, New York, NY, USA, 3–18. doi:10.1145/2541940.2541965
- [21] NVIDIA Corporation. 2026. NVIDIA GB200 NVL72. NVIDIA Data Center Website. Accessed 2026-07-09. <https://www.nvidia.com/en-us/data-center/gb200-nvl72/>
- [22] Simon Peter, Adrian Schüpbach, Dominik Menzi, and Timothy Roscoe. 2011. Early experience with the Barrelfish OS and the Single-Chip Cloud Computer. In *Proceedings of the 3rd Many-core Applications Research Community (MARC) Symposium*. KIT Scientific Publishing, Karlsruhe, Germany, 5 pages. https://barrelfish.org/publications/barrelfish_marc11.pdf
- [23] Kamran Razavi, Mehran Salmani, Max Mühlhäuser, Boris Koldehofe, and Lin Wang. 2024. A Tale of Two Scales: Reconciling Horizontal and Vertical Scaling for Inference Serving Systems. arXiv:2407.14843 [cs.DC] <https://arxiv.org/abs/2407.14843>
- [24] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiyi Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 69–87. <https://www.usenix.org/conference/osdi18/presentation/shan>
- [25] Arman Shehabi, Sarah Smith, Dale Sartor, Richard E. Brown, et al. 2016. *United States Data Center Energy Usage Report*. Technical Report. Lawrence Berkeley National Laboratory. doi:10.2172/1372902
- [26] Arman Shehabi, Sarah Josephine Smith, Alex Hubbard, Alexander Newkirk, Nuoai Lei, Md AbuBakar Siddik, Billie Holecek, Jonathan G. Koomey, Eric R. Masanet, and Dale A. Sartor. 2024. *2024 United States Data Center Energy Usage Report*. Technical Report. Lawrence Berkeley National Laboratory. doi:10.71468/P1WC7Q
- [27] Sarah Smith, Alexander Hubbard, Alex Newkirk, Mohan Ganeshalingam, Billie Holecek, Dale Sartor, Michael Mills, and Arman Shehabi. 2026. *United States Data Center Energy Usage Report: 2025 Update*. Technical Report. Lawrence Berkeley National Laboratory. doi:10.71468/P1RP4F
- [28] Micah Walter. 2026. Run isolated sandboxes with full lifecycle control: AWS Lambda introduces MicroVMs. AWS News Blog. Published 2026-06-22; accessed 2026-07-04. <https://aws.amazon.com/blogs/aws/run-isolated-sandboxes-with-full-lifecycle-control-aws-lambda-introduces-microvms/>
- [29] David Wentzlaff and Anant Agarwal. 2009. Factored operating systems (fos): the case for a scalable operating system for multicores. *SIGOPS Oper. Syst. Rev.* 43, 2 (April 2009), 76–85. doi:10.1145/1531793.1531805
- [30] Tong Xing, Jiaxun Yang, Javier Picorel, and Antonio Barbalace. 2026. Rethinking Tiered Memory Management in Cloud Data Centers. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC '25)*. Association for Computing Machinery, New York, NY, USA, 74–87. doi:10.1145/3772052.3772213