International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 729-734
http://www.ijcesen.com

ISSN: 2149-9144

Copyright © IJCESEN

Research Article

Comparing Globally Distributed Databases and Traditional Relational Databases: An Architectural Analysis

Amit Kumar Garg*

Independent Researcher, USA
* **Corresponding Author Email:** ami2t@gmail.com - **ORCID:** 0000-0002-1047-6650

Article Info:

DOI: 10.22399/ijcesen.4900
Received : 29 November 2025
Revised : 25 January 2026
Accepted : 02 February 2026

Keywords

Globally Distributed Databases,
Relational Database Architecture,
Disaster Recovery Mechanisms,
Consensus-Based Consistency,
Operational Complexity Trade-Offs

Abstract:

Globally-distributed database designs differ from customary RDBMS designs. Trade-offs in multi-region operational capacity, disaster recovery, and long-term operational viability can lead to both conflicting and complementary planned priorities. Globally-distributed databases make use of synchronous replication protocols, automated failover procedures, and consensus-based consistency models to provide strong availability guarantees across physical and geographic boundaries. However, these designs still suffer from the intrinsic latency and performance tradeoffs of cross-region coordination, while relational database architectures like PostgreSQL are able to achieve superior local performance characteristics and cost efficiency when workloads are localized to regions. That said, they require explicit operational documentation and orchestration for cross-region disaster recovery and failover. Beyond technical considerations, the architectural choices also include differing assumptions of system complexity: is it centralized within the database platform or distributed among operational processes and organizational practices? Organizational preference for a high base cost with automated resilience or for a low base cost with flexibility and explicit DR responsibilities affects the choice of architecture. An organization's decision depends on the requirement for geographic distribution, the tolerance for downtime and access latency, the cost of the infrastructure, and the availability of operational expertise.

1. Introduction

The model of data management systems has changed, and architects are considering a new perspective on the architecture of databases as the transition from silo architectures to multi-region and multi-failure domain solutions. The choice of databases in the current cloud-native world is an architectural choice that has a deep impact on the resiliency, performance characteristics, and operation of the domain. The rise of cloud DBMS has moved the enterprise DBA to manage the cloud data infrastructure from single-region deployments to the realization that the customary single-region deployments are no longer sufficient to meet the business needs of global access and high availability [1]. There are two leading methods of providing such environments. The leading options are global database distribution built for multi-region deployment, and customary SQL databases with replication, backup, and disaster recovery facilities.

To analyze these architectural choices, the usage of real-life operation data, disaster recovery systems, and workloads is assessed to assess the characteristics of databases. The chosen representative globally distributed SQL databases and PostgreSQL-compatible relational database systems as representatives of the architectural solutions to explore the differences between them. Given the importance of total cost of ownership and operational considerations as organizations build out their data infrastructure, it is increasingly relevant to understand the architectural trade-offs of these options in the context of the overall long-term viability of the data infrastructure investment [2]. The goal of this paper is not to advocate on behalf of one or the other approach, but rather to provide clarity on the contexts in which different data management models become helpful and the trade-offs of each.

2. Architectural Foundations and Design Philosophy

The architectural principles of globally distributed databases differ from other databases, including relational databases, in that they assume that failures are common and that failures are strongly correlated with geography. In other words, globally distributed databases often assume failure is inevitable in at least one part of the network. Shared properties of all these systems are the ability to replicate synchronously over site boundaries, automatic leader election and promotion of replicas, strong consistency guarantees across all replicas, and transparent partitioning and rebalancing of data and transactions. Their consistency models tend to differ from customary consistency models. In particular, there is a trade-off between the consistency of geographically dispersed nodes and the latency caused by distributed consensus algorithms [3]. This architectural pattern is intended to make handling application-level failures easier, at the cost of more coordination overhead in normal conditions.A characteristic of global databases is external consistency, which requires that replicas of the database make atomic progress in all transactions. In this sense, all transactions are simultaneous at a point in time, regardless of the geographical location of the replicas. In practice, this requires high-precision clock synchronization and multi-version concurrency control. These properties manifest in the design of the systems. For example, they realize automatic replica placement in a database and maintain sufficient replicas at all times to reach quorum in case of any regional network partition or a regional outage. These are made possible by a design strategy that focuses on correctness and availability over raw performance, and that treats cross-region latencies as a first-class design consideration.In contrast, legacy relational databases such as PostgreSQL were originally designed around a single primary architecture, and the history of extending these original, single primary databases to meet availability goals can be seen in read replicas (same region or cross region), automated or semi-automated failovers (also same region or cross region), periodic backups, point in time recovery, and cross region disaster recovery. PostgreSQL provides logical replication capabilities, which are a work in progress to support more complex replication topologies. These include selective table replication, bidirectional replication, and more advanced conflict resolution operations, which allow for less overhead in distributed PostgreSQL deployments [4]. While their key advantages are local performance and cost reduction, their major disadvantage is the more involved planning needed when operationally recovering from failures.

Architectural principles of PostgreSQL-based systems focus on predictable performance and efficiency for a defined operating range. Instead of hiding the complexity of distributed systems, PostgreSQL deployments expose this complexity, giving operators fine-tuned control over replication topology, failover behavior, and consistency trade-offs. This transparency allows organizations to optimize their deployments for their specific workload characteristics and availability requirements and places greater responsibility on operational staff to design, deploy, and implement the appropriate disaster recovery strategies. There is a philosophical dimension to this; globally distributed systems internalize the complexity of coordinating data in the database platform, while customary relational systems externalize this complexity to operational processes and procedures, allowing organizations to be explicit about their engineering cost and architectural trade-offs.

3. Availability Architecture and Disaster Recovery Mechanisms

However, it is arguably the handling of failure detection and recovery that is the most important distinguishing feature. Infrastructure-level failures in globally distributed networks are handled in a largely autonomous fashion. Loss of replicas, availability zones, and even regions can be tolerated via quorum-based replication protocols and automatic leadership election. In many cases, recovery is automatic, taking seconds or less, and requiring no operator action. These systems reset globally distributed databases to an internally consistent state. The architecture of such a system is concerned with automatic consistency mechanisms for geographically distributed replicas, often using consensus-based, agreement protocols for synchronous data replication [5]. The automation in these systems is derived from the architectural choices made in the system, since failures are handled in the database platform itself (using heartbeat protocols to get failure information, leader election using distributed consensus protocols, and traffic adjustments to working replicas without external orchestration).These availability models rely on consensus algorithms that balance the conflict between strong consistency and network latency and partition tolerance. In the case of globally distributed databases, the data is replicated across multiple nodes at geographically disparate locations, and a write is not considered committed until it is acknowledged by a majority quorum of the replica nodes. This pattern is intended for when

data needs to be available and consistent in an entire region (or one or more replicas) is unavailable. The remaining replicas in other regions can continue responding, so the system remains available. This great benefit is also complemented by the reduced operational costs, since operational teams do not have to rush to promote a replica in case of a common disaster or reconcile state differences.While PostgreSQL systems can be made highly available within the same region, their properties differ fundamentally when it comes to cross-region disaster recovery. Replicas usually require pre-provisioned infrastructure and explicit promotion/switchover events, and while modern managed services have stepped up their automation game, PostgreSQL systems still rely on human operators to make decisions when recovering from a disruption. Organizations that use PostgreSQL and desire high availability and disaster recovery must attempt to reconcile the trade-off between the consistency and durability guarantees of PostgreSQL synchronous replication with the write latency of PostgreSQL asynchronous replication, or vice versa, in terms of data loss, which is observed in operational documentation. This trade-off difference manifests as lower operational overhead during unplanned regional failure for globally distributed databases, and greater clarity during controlled PostgreSQL database recovery events.Disaster recovery for PostgreSQL deployments usually consists of standby replicas in regional failover locations used for streaming replication, backups that are configured and automated for point-in-time recovery (PITR), and a documented procedure to promote standby replicas to primary. Disaster recovery for PostgreSQL adds management overhead of ensuring replication is functioning, backups are retained for business needs, and failover processes are practiced and work correctly. Such explicitness provides a finer-grained control over RPO and RTO as they relate to business needs, at the price of added operational effort. Comparison examines the relative nature of RPO and RTO by contrasting distributed databases, which tend to converge on near-zero RPOs with extremely low RTOs, with PostgreSQL deployments, which tend to tolerate measurable replication lag across regions, resulting in non-zero RPO on failover.

4. Performance Characteristics and Scalability Dynamics

In practice, the network and the database have measurable differences. A PostgreSQL-based architecture provides lower latencies for read and write operations within the same region or nearby replicas. Global databases are slower than regional databases that do not use cross-region consensus and commit protocols, due to the added latency incurred by breaking the physical boundaries of regions. The performance of distributed databases is a trade-off of consistency, availability, and partition tolerance, and all distributed systems with strict consistency guarantees have a latency penalty for writes, as they must coordinate writes to geographically distributed replicas of the database to keep all nodes consistent with the committed data [7]. This is more pronounced at high percentiles, in that the effect of coordination dominates the performance characteristics of the system, and tail latencies are dominated by the slowest replica participating in the consensus protocol than the median performance of the system.

The write amplification of distributed consensus protocols determines the performance of globally distributed databases. Every write must be replicated to multiple nodes and then acknowledged from a quorum of replicas before it can be committed. The round-trip time for this operation is proportional to the number of replicas and the distance between them. Coordination cost is of particular concern in workloads with numerous smaller transactions, where coordination cost dominates transaction processing cost. In exchange for the overhead of coordination, strong consistency guarantees that the application's developers no longer have to manage the problems of eventual consistency and reconciling concurrent transactions across different replicas.

A comparison of peak and non-peak operation shows the elasticity characteristics of database architectures. For most database systems, peak operation is better enabled by automatic scaling and dynamic rebalancing of partitions across the available nodes. This smooths contention and distributes the load across the available nodes without the intervention of a database administrator. PostgreSQL-based systems require more important capacity planning to avoid resource saturation under load variability, but they achieve excellent baseline performance. The analysis of compression schemes and memory management strategies shows how modern DBMS can benefit from architectural optimizations in the system, with different advantages deriving from whether the database is distributed or centralized [8]. This suggests a philosophical difference between the targets of automatic elasticity and predictable data locality.In terms of scalability, globally distributed databases are expected to directly support horizontal scaling across partitions with minimal application intervention. Their partitioning schemes

achieve near-linear scalability for many workloads more or less directly, enabling both storage capacity and computation throughput to incrementally increase with the addition of each new node to the system. The horizontal scale model is well-suited to workloads where there is uniform access to all rows within a table (for example, a large table). In this case, vertical scaling of nodes will not result in a meaningful performance improvement. PostgreSQL systems can scale with read workloads across replicas, but not with write workloads without vertical scaling, application-layer sharding, or major architectural change. As such, globally distributed systems are often introduced, with write scalability and scalability along geographical dimensions being the primary objectives, while PostgreSQL is still the preferred option when the workload tends to be more locally bound, and it is generally worth the effort of manually distributing the data.

5. Operational Complexity and Cost Considerations

At a high level, the operational dimension shows how complexity manifests itself in the two types of architectures. Globally distributed databases move much of the complexity to the system architecture. This increases the importance of schema design and data access pattern optimization. It also requires a careful definition of transaction scope. Routine maintenance tends to be limited, and many organizations running globally-distributed databases find that operation is less about responding to incidents and more about the architecture of the system. Design-time decisions about data locality, transaction granularity, and query patterns have far-reaching implications for the long-term performance and economics of the system. PostgreSQL systems push operations complexity into the database: backup management, replica health checking and failover testing, disaster recovery rehearsals, and other such activities remain a regular part of database operations. Application semantics between models are similar, but neither model is conceptually simpler: each manages complexity differently in the stack. Globally distributed systems require more knowledge of distributed systems in application design and, for PostgreSQL, complex operational procedures to achieve high availability and expertise in the database domain.Much of the daily effort involved in the operation of a PostgreSQL deployment involves operating the database lifecycle. Database administrators may configure and manage regular backups and recovery point objectives, monitor replication lag to prevent replicas from falling too far behind the primary for failover, conduct disaster recovery drills, and manage schema migrations across both primary and replica servers to avoid breaking the logical replication. These activities are engineering-intensive both in terms of their domain knowledge and special operations, but allow for a fine-grained understanding of the system and workload tuning. The operational model pays dividends as automation tooling and operational processes mature, allowing highly mature operational teams to operate large-scale PostgreSQL workloads via advanced monitoring, alerting, and automated repair tooling.

Although no specific figures are provided regarding cost and performance, there are general trends: highly distributed systems sacrifice the relatively high costs of their base infrastructure for simpler operation and higher availability. The total cost of ownership of a distributed database system includes infrastructure, human resources cost for management and operation of a distributed system, potential slowdown of application development, and opportunity cost of engineering time spent on learning patterns for optimizing distributed transactions [9]. Compared to other distributed databases, PostgreSQL has a lower cost per unit of performance. This is because it can be run at low cost and fairly well on distributed workloads on a local scale. Cost increases with redundancy and availability requirements. This means ingraining cost estimation not just in relation to the cost of the infrastructure but also the attitude towards risk and the division of costs among team members: the total cost of ownership includes staffing, downtime, and opportunity cost of delaying feature development.

In addition to the costs of the infrastructure and staff, there are the business costs of the availability and performance of the system. For the organization, this may be in the loss of revenue during an outage, the business benefit of the low latency of worldwide data access, or the operational simplicity that allows rapid repeat iterations. The architectural decisions associated with distributed databases versus customary databases reflect some of the differences in trade-offs regarding where operational complexity is handled within the stack. Distributed databases have generally chosen to handle operational complexity through the database implementations themselves. Customary databases have generally chosen to expose the operational complexity so that it may be customized and optimized [10]. Trade-offs such as these may help inform choices that align a database's operational design and trade-offs with an organization's business and strategy. Neither model

is intrinsically better; it depends on workload characteristics, the availability of resources, geographic distribution, and organizational capabilities.

Table 1: *Cloud Database Management Systems and Cost Implications [1][2]*

Aspect	Cloud Database Systems	Traditional Deployments
Infrastructure Model	Multi-region native support	Single-region with extensions
Availability Approach	Built-in global distribution	Requires configuration
Cost Structure	Higher baseline, lower operations	Lower baseline, higher operations
Scalability Pattern	Automated horizontal scaling	Manual capacity planning
Operational Burden	Minimal routine maintenance	Significant ongoing attention

Table 2: *Consistency Models and Replication Architectures [3][4]*

Database Characteristic	Distributed Systems	PostgreSQL Systems
Consistency Guarantee	External consistency across replicas	Configurable consistency levels
Replication Mode	Synchronous multi-region	Streaming with lag tolerance
Conflict Resolution	Automated consensus protocols	Manual or semi-automated
Transaction Coordination	Cross-region atomic commits	Regional transaction boundaries
Architectural Complexity	Internalized platform complexity	Externalized operational complexity

Table 3: *Disaster Recovery and Availability Mechanisms [5][6]*

Recovery Aspect	Globally Distributed	PostgreSQL-Based
Failure Detection	Automated heartbeat mechanisms	Monitoring-dependent detection
Replica Promotion	Consensus-based leader election	Explicit promotion procedures
Geographic Resilience	Multi-region quorum maintenance	Pre-provisioned replica requirements
Recovery Automation	Minimal operator intervention	Deliberate operational actions
Data Loss Tolerance	Near-zero RPO scenarios	Measurable replication lag

Table 4: *Performance and Scalability Characteristics [7][8]*

Performance Dimension	Distributed Databases	Traditional Relational
Write Latency Profile	Higher due to coordination	Lower for local operations
Read Performance	Geographic distribution dependent	Optimized for locality
Scalability Direction	Horizontal by design	Vertical with sharding extensions
Load Elasticity	Automated partition rebalancing	Manual capacity adjustments
Optimization Focus	Consistency over latency	Performance over automation

6. Conclusions

Both globally-distributed database systems and PostgreSQL-based relational database architectures are mature, production-proven implementations of the same data storage at scale model. The difference between the two approaches is not in their features or technical implementations but in their assumptions and expectations about failure modes, distribution, and ownership of the operations. Globally distributed databases are generally appropriate when: high availability in multiple regions is a requirement, the tolerance for downtime is near zero, maintenance over latency is prioritized, and multi-region workloads are expected. PostgreSQL-based systems should be used when: the workload is mainly regional, low latencies are desired, cost is a major consideration, and the organization has the capacity and willingness to operate its own disaster recovery capabilities. When it comes to selection, it is often the case that an important mental model is that the key decision is where to place the complexity, in the database platform itself, or in the operational processes and procedures that must deal with the platform. It is far more important to understand the fundamental assumptions in making these choices, and satisfying business needs and engineering priorities, than to obsess over individual benchmark results and technical specifications. In globalized operational environments, as organizations develop data architectures, this framework for analyzing database systems remains one avenue for making context-specific architecture decisions concerning technical and organizational issues. The future of database architecture lies not in one form of architecture being more helpful than other architectures, but rather in the matching of database capabilities to a particular operational environment,

where an architecture decision represents a cluster of trade-offs that takes into account organizational objectives, risk appetite, and the enterprise strategy.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Cecilia Lin, "Cloud Database Management Systems: A Comprehensive Guide," Larksuite, 2025. [Online]. Available: https://www.larksuite.com/en_us/blog/cloud-database-management-systems

[2] Siemens, "The True Cost of Downtime 2024," 2024. [Online]. Available: https://assets.new.siemens.com/siemens/assets/api/uuid:1b43afb5-2d07-47f7-9eb7-893fe7d0bc59/TCOD-2024_original.pdf

[3] Daniel Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," ACM Digital Library, 2012. [Online]. Available: https://dl.acm.org/doi/10.1109/MC.2012.33

[4] Ahsan Hadi, "PostgreSQL 16 Logical Replication Improvements in Action," pgEdge Blog, 2023. [Online]. Available: https://www.pgedge.com/blog/postgresql-16-logical-replication-improvements-in-action

[5] James C. Corbett, et al., "Spanner: Google's Globally Distributed Database," ACM Digital Library, 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2491245

[6] Arun Seetharaman, "High Availability and Disaster Recovery: PostgreSQL on AWS — Part 5," Medium, 2024. [Online]. Available: https://medium.com/@arunseetharaman/high-availability-and-disaster-recovery-postgresql-on-aws-part-5-d0521f5af470

[7] Michael Stonebraker, Ariel Weisberg, "The VoltDB Main Memory DBMS," IEEE Data Engineering Bulletin, 2013. [Online]. Available: http://sites.computer.org/debull/a13june/voltdb1.pdf

[8] Carsten Binnig, et al., "Dictionary-based order-preserving string compression for main memory column stores," ACM Digital Library, 2009 [Online]. Available: https://dl.acm.org/doi/10.1145/1559845.1559877

[9] Feifei Li et al., "Modernization of Databases in the Cloud Era: Building Databases that Run like Legos," VLDB, 2023. [Online]. Available: https://www.vldb.org/pvldb/vol16/p4140-li.pdf

[10] Milvus, "What is the Difference Between a Distributed Database and a Traditional Relational Database?" [Online]. Available: https://milvus.io/ai-quick-reference/what-is-the-difference-between-a-distributed-database-and-a-traditional-relational-database