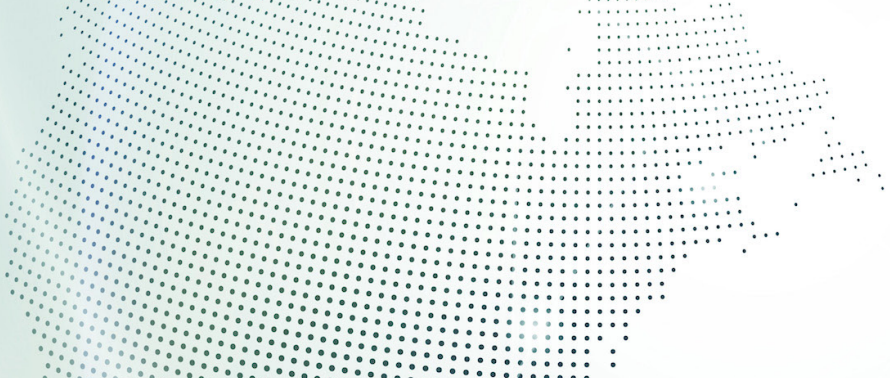
EBOOK

DATABASE CONSIDERATIONS FOR GLOBAL APPLICATIONS



NUODB®

Global Operation is the subject of running a logical service across multiple physical locations.

Increasingly, it is a requirement to deliver software as a service and for enterprise deployments. Because many solutions are custom and based on loose best practices, it can be hard to quantify the common challenges that need addressing or the likely trade-offs to consider, especially in regards to data management.

This ebook is a short introduction into this space.

Introduction

Historically, client-server architectures focused on running a service on a single server, accessible to multiple clients. To achieve higher throughput or handle more clients, that single server can be scaled up and replaced with a bigger single server. While this is expensive and somewhat inflexible, it does support scaling, at least up to the capacity provided by large servers. What it does not address is availability, disaster recovery, and cost-effective resource utilization. To this end, single-server systems started scaling out through replicated, stateless or shared-nothing architectures that typically focus on one of the following: performance, resource efficiency, or safety.

A decade ago, this was mostly the concern of a few high-end applications. Today, however, focus on cloud architectures has made horizontal scale a requirement. Systems are built as collections of many small systems instead of one or two large servers. Originally focused on running within a data center over low-latency networks, these scale-out models are increasingly operating globally, running across data centers and between geographies. Partly this is because a move to commodity infrastructure, and public clouds has made global resources available. Equally, the increase in wireless access, smartphones, and other technology has made it easier to interact with systems from different locations.

At a high-level, there are at least three key motivations for operating at global scale:

+ **Localized Latency.** If users need to interact with a service from multiple locations, then it makes sense to provide access to that service from locations close to those users. This lowers latency to the service while improving total throughput by providing more points of access.
+ **Availability and Fault Tolerance.** If a single data center can fail (through power loss, network outage, cascading failures, physical disasters etc.) then it makes sense to run in multiple data centers. As long as a service is able to do the same work in each physical location, then availability is improved. Further, if data is replicated between sites, then the service can tolerate complete data center failures.
+ **Localized Data Management.** If data is mostly used in one specific location then it makes sense to keep that data at that location as long as it's still available as-needed on a global scale. Increasingly, legal or regulatory requirements state that data must be stored only in a specific location to meet data residency (also known as data sovereignty) concerns.

Scaling a service from a single location to run across wide areas will always bring with it trade-offs. Understanding these issues is critical to characterizing your needs and successfully addressing the above three motivations. In all three cases, the real requirement is with the underlying data model and how it supports global access; stateless services with no dependency on underlying data are relatively easy to scale.

> "Measuring and quantifying elements of global deployment requires a different perspective."

How to Characterize Global Data Models

Traditional benchmarking is focused tightly on metrics like operations per second or average latencies. Measuring and quantifying elements of global deployment requires a different perspective. For instance, measuring average and peak latency is still important but interpreting what that means for an application varies depending on visibility and durability rules. Likewise, throughput in a single location may or may not matter as much as aggregate rates. When you're thinking about building a new application or moving an existing application into a global deployment model, there are a few broad ways to categorize its behavior.

Local Versus Global Access

One way to categorize a global data model is to think about where application or service data is accessed. On one end of the spectrum are applications that decompose to entirely site-local access. In other words, any given piece of data is only ever accessed in a single location. This might be something like site-specific web content or local session management.

At the other end are applications that access all data in all locations. This might be common content for a global application or company-wide news and updates. In practice, most data models fall somewhere in between these two extremes. There is some data that is typically accessed at one given site and some that is needed in multiple or all sites. This may also be temporal in nature, where the same data is needed at multiple sites over time but at any given window of time is only accessed at one site.

For this simple access model, the key trade-off to think about is where data is stored and how that affects availability in the face of failures. Keeping all data in all locations makes failure handling simple, but at the cost of total resource utilization. For compliance reasons, it may not be acceptable to keep all information in multiple sites. Partitioning data sets, on the other hand, addresses residence and total cost of ownership (TCO) at the cost of operational complexity and possible vulnerability on network or site failures. These decisions become more important when a second factor, update patterns, is added.

Local Versus Global Contention

Another way to categorize a global data model is to identify where changes are made and therefore where access may be contended. As with general data access, contention can be thought of on a continuum. At one extreme is data that may be shared between sites but is only updated in a single location or by a single actor. In this model, the contention is local and updates shared as needed. Whether the contention is transactional in nature or unmediated and left to the application to sort out, localizing update contention simplifies the data model.

At the other extreme are applications where all data is contended in all locations. This pattern tends to be the hardest to handle and will either result in high latencies or considerable conflict for an application to mediate. As with the previous discussion of general data access, however, most real-world update patterns tend to fall somewhere in the middle of this spectrum. Again, often the pattern is temporal, with updates happening in different sites but at different times so that contention is still fairly localized.

The additional trade-offs to consider around contended changes have to do with latencies and correctness of data. A change that is made locally and propagated asynchronously to remote sites will be fast but could be lost in the face of system or network failures. Making propagation synchronous guarantees correctness on failure but at the cost of higher latencies. Synchrony can make contention management easier or guarantee an application always sees the canonical state of data but at the cost of performance and localization of activity.

Common Use Cases

To help give context to these two spectrums, here are some global operations scenarios. Some are common models today while others are the new architectures that services are trying to move towards. Each of these illustrates the trade-offs discussed above.

+ **Replicated State.** Common examples of this today are implementing a hot-standby site, or maintaining a remote copy of data for disaster recovery. In this pattern all activity is localized to a single site but all data is replicated to each secondary.
+ **Active Replication.** Also referred to as Active-Active or Multi-Master, this approach typically runs a service in two locations where all data is available at both sites and a load-balancer chooses where a given client does work at any time so contention is mostly (but not always) localized.

Common Global Operations scenarios:

+ Replicated State
+ Active Replication
+ Local Activity Log
+ Coordinated Activity
+ User-Driven Activity
+ Clustered Activity
+ Global, Time-Shifted Activity
+ Publish-Subscribe
+ Global Insight
+ Data Residency and Audit

+ **Local Activity Log.** Maintaining session state, activity logging, call records etc. is standard practice in many systems and sometimes a regulatory requirement. Entries are created, accessed, and contended locally, but there is usually the need to track the aggregate data for audit or read the data as a whole (see Global Insight on page 9).
+ **Coordinated Activity.** In this pattern, clients coordinate to work together using shared data at a single local site but "check-in" or otherwise coordinate amongst other groups through small amounts of globally accessed and contended data.
+ **User-Driven Activity.** Any application that serves individual users, like SSO, commerce, or account management is driven by the activity of its users. Typically those users are accessing and updating their own data in one place, but that site may change if the user moves location. A user's data may be replicated or may be constrained to a geography.
+ **Clustered Activity.** User-driven applications that are social in nature exhibit natural clustering. Often this clustering maps to physical locality (friends live in the same area, discussion is about a local landmark or event), but it tends to draw a few users who are physically remote. Typically, these applications localize data and activity paying a latency cost for remote users, but that changes as remote users start to dominate a grouping.
+ **Global, Time-Shifted Activity.** Applications like financial trading have large sets of shared data and smaller clusters of data that is updated on a global scale. However, because different areas of the world are active at different times (often called a "follow the sun" model), global contention rates tend to be fairly low with most updates clustered at a single site at any given time.

+ **Publish-Subscribe.** Some applications create or modify data at a single place (e.g., driven by a user or tied to local infrastructure), but need to make that data globally available. All contention is local but all data is accessed globally. A popular example is Internet of Things (IoT) networks, where small devices manage their own states and/or reporting but anyone may monitor specific devices or be looking for aggregate trends.
+ **Global Insight.** The "aggregate trends" mentioned in the previous scenario is a more general motivator for global access. Increasingly, ETL is being replaced by the need to keep data in one service but use it for different problems. This makes operational use-cases that might otherwise have local data access models (e.g., voice over IP log management) exhibit a global access model. Complicating this is that global analytics often conflicts with local privacy laws. Luckily, these are typically read-only tasks.
+ **Data Residency and Audit.** By definition, data residency is about controlling where data can reside, in some cases localizing storage, and in other cases limiting the number of storage sites. Access to that data, however, may still be global but would tend to exhibit the User-Driven Activity or Global, Time-Shifted Activity patterns.

> **"Different use cases will dictate different trade-offs such as data correctness, latency, or operational complexity."**

Summary

As global operations increasingly is a requirement for applications and services, underlying data models must support these deployments. In order to support scale, flexibility, governance, and fault tolerance, it's important to design with the right model, which means understanding your key requirements and what trade-offs may incur. Global data management models can be considered on the axis of access and contention. By understanding where a given application falls in this space, it may be easier to categorize common behaviors and think about what optimizations will best match your data needs.

About NuoDB

First envisioned by industry-renowned database architect and innovator Jim Starkey, NuoDB was developed to tackle the multiple challenges associated with cloud computing and the rise of global application deployments.

Backed by three former CEOs of the four original relational database companies, NuoDB addresses a seemingly impossible problem: Build a database suitable for mission-critical workloads – maintaining both SQL capabilities and full ACID compliance – while simultaneously delivering global access, on-demand scalability, and cloud- or container-based deployment.

In short, become the only database that can maintain transactional consistency and integrity at global scale.

NuoDB is headquartered in Cambridge, MA, USA, with offices in Dublin and Belfast. For more information, visit nuodb.com.

info@nuodb.com | +1 (617) 500-0001 | www.nuodb.com