NuoDB®

WHITE PAPER

4 Key Considerations for Maximizing Database ROI for Banking Applications

Website:
www.nuodb.com

Email:
sales@nuodb.com

Phone:
+1 (617) 500-0001

World Headquarters
150 Cambridgepark Drive
Cambridge, MA 02140
United States

Executive Summary

As the pace of innovation accelerates, enterprise financial services organizations are considering how to increase efficiency and reduce costs as they deploy banking applications that meet the demands of customers today. Existing banking applications dependent on traditional monolithic databases can be difficult to deploy in the cloud, costly to scale, and unable to quickly adjust to the increasingly dynamic needs of today's marketplace. Moving to a cloud-native model introduces many new features and considerations, necessitating an examination of database costs and an evaluation of the return on investment for selecting the right database model. As financial services organizations weigh how to move applications to the cloud, they must seek out the right partners to help them maximize the benefits of that shift.

Your database choice significantly impacts the cost structure of your application. Licensing cost considerations for databases can be complex, particularly as you move from a traditional monolithic database to a cloud-based database. Sizing traditional databases relies on modeling the maximum capacity required, plus additional capacity to handle growth and even more built in as a buffer. That's commonly referred to as pre-provisioning, essentially sizing the database to accommodate the peak loads of end-of-day, end-of-month, and end-of-year processing needs now and for the next couple of years. This need to accommodate peak loads significantly increases costs, as the maximum hardware necessary is utilized only a fraction of time. A traditional database can only be scaled up to meet demand, it cannot dynamically scale out and in again based on the needs of the application. In addition, these traditional databases require significant investment to ensure resiliency and redundancy in case of hardware or other failures. That means purchasing database hardware built for peak demand plus more backup hardware to provide failover and disaster recovery capabilities.

As financial services organizations consider their next generation multi-data center strategies to meet the demands of customers, they need new strategies to keep complexity and management efforts to a minimum, while still keeping a close eye on managing costs. A distributed SQL database eliminates many of the traditional trade-offs required when using a monolithic database architecture by ensuring access to a single logical database that can be natively distributed across multiple nodes, in on-premises data centers, in public and private clouds, and in a combination of on-prem and cloud deployments. In addition to offering cloud-native and cloud-agnostic architecture, a distributed SQL database eliminates pre-provisioning needs while also changing how resiliency and redundancy are provided, significantly reducing the total cost of ownership for the database infrastructure.

With several different options to choose from, the total cost of ownership becomes an important factor when it comes to choosing the right database architecture, both for deploying new banking applications and for making legacy applications available in the cloud. In this white paper, learn how choosing cloud-native, cloud-agnostic database solutions for financial services applications such as core banking and payments can help you reduce complexity and expense by: eliminating the costs of pre-provisioning, deploying a database with built-in resiliency, meeting demanding availability requirements, improving performance, and lowering costs by changing licensing models. Making a change to banking applications and the database these applications are deployed on can have far reaching benefits for financial services organizations of all sizes.

1. Evaluate Database Costs

An ROI calculation is a financial estimate that helps buyers calculate the direct and indirect costs of a product or system. Clearly, direct costs are fairly easy to calculate, but determining the indirect costs is a far less precise process. Direct costs for a database depend on the type of deployment, but typically include database servers, licenses, add-on components such as memory and storage upgrades, software tools and components, and general data center costs (for example, energy usage, cooling requirements, and rack space). Let's not forget the support costs and potential training or hiring costs.

In addition, databases don't exist in a vacuum. Your developers also must build the applications that rely on the database to perform as required. If you've implemented the application and database in the cloud, many of the direct costs become outsourced and billed based on usage. If you omit evaluation of the indirect costs of a database, you might be under the misapprehension that the total cost of ownership (TCO) is the cost of the monthly cloud rental plus the costs of hiring and training. However, the following are indirect costs that you need to consider in order to have a complete understanding of database costs.

Vendor Lock-In

Today, nearly all databases run on commodity hardware, which also means that they can run in the cloud. An additional indirect cost occurs when you decide to move to other hardware, move to the cloud, or choose to run an application as a service. Being tied to a particular cloud provider has significant hidden costs of getting your data out of the cloud provider if you need to switch cloud providers for regulatory or business reasons. For financial services organizations moving existing applications to the cloud or building new applications in the cloud, it could be a costly oversight if you don't plan for flexibility in your deployment.

Porting Costs

There are three essential database concepts to keep in mind as you evaluate a move to the cloud: the query language, which is structured query language or SQL, the way that the database handles transactions, and how consistency requirements are met. There are numerous databases available for cloud deployment today, but many of those handle these concepts differently. The indirect cost of porting an existing application to a database that implements these concepts differently can be significant. This is especially true for critical banking applications, which are built using SQL and require the ability to handle high volumes of transactions while maintaining ACID (Atomicity, Consistency, Isolation, and Durability) properties. Since banking applications typically require on strongly consistent data, ensure any database considered meets the level of consistency needed.

Ease of Development

Not all databases are equal when it comes to development. For example, some NoSQL databases may involve a significant learning curve for your development team. The indirect

costs involved can accrue in terms of the time needed to learn a new solution as well as learning how to build it with the functionality you need, while still seeking to get a product to market quickly. Sophisticated critical banking applications typically rely on relational database management systems (RDBMS). It's important to select a database that offers both the functionality and flexibility you need, while still enabling your developers to use existing skills and code to build a new application in order to deliver it to market faster than your competition.

Ease of Management

In addition to any specialized skills your development team might need to learn in order to address changes required by non-relational databases, databases need administration staff to monitor and manage them, to tune the database performance, and to manage the use of resources. It's important to select a modern database that fits well with microservices, containers, and container orchestration solutions such as Kubernetes, so you can take advantage of cloud native technologies and capabilities and reduce management costs through automation.

> "**Over three quarters** of Americans used a mobile device the last time they checked their account balance." ***- Statista***

Continuous Availability

If your customers can't use your banking application when and where they want to, they'll find another bank that offers solutions that do. Customer expectations have changed dramatically since the introduction of mobile phones, especially since Apple introduced the first iPhone in 2007. Today, customers expect their applications to be available, wherever and whenever required. When you think about application availability, you may not consider how the database fits in. Essentially, when a database is down due to hardware, software, or unavailable due to planned maintenance or for upgrade purposes, your application isn't available because it relies on the database to provide the information essential to your banking application. The indirect cost is the cost of the lost business caused by downtime. In a world that's on 24/7, this cost continues to rise each year.

Migration Costs

The ability of a database to be easily distributed across multiple data centers or between the cloud and a data center is very important to consider. Very few databases are engineered to offer this type of flexible distribution while still maintaining ACID semantics for mission critical SQL dependent applications. The ability to distribute data on-premises, in public and private clouds, and in hybrid environments significantly lowers the cost of delivering high availability and disaster recovery. Additional advantages of a distributed SQL database include the ability to provide data locality to meet regulatory requirements, migrate data or databases from one geographic location to another, or to deploy a single logical database across multiple clouds without incurring significant additional costs.

2. Eliminate Pre-provisioning

Traditional databases require you to pre-provision your database to meet peak capacity needs, so that your database can accommodate the peak loads of end-of-day, end-of-month, and end-of-year processing. Database servers in a traditional data center typically run at about 5%-10% utilization, which means that most of the time you're paying for 90%-95% capacity that goes unused. Using traditional SQL databases, adjusting for increasing volume has typically been a balance between pre-provisioning and scaling up with new monolithic hardware, which requires a service outage.

As more applications today require scale-out capabilities, add-on solutions (such as read replicas) have entered the market from traditional vendors. Such solutions add both cost and complexity to the SQL database architecture without providing true on-demand scalability. It has become apparent that retrofitting single-server technology for a distributed, cloud-based world frequently fails to solve the problem of on-demand scalability for the database.

Instead of trying to force legacy databases into a cloud-native world, look for a distributed SQL database designed to deliver on-demand scalability, strict transactional consistency, rich ANSI-SQL support, and continuous availability. This type of database enables deployment of enterprise applications on cloud infrastructures while maintaining the ecosystems, skill sets, and best practices of the traditional SQL database.

> "In normal English usage the word resilience is taken to mean the power to resume original shape after compression; in the context of database management the term data base resilience is defined as the ability to return to a previous state after the occurrence of some event or action which may have changed that state."
>
> ***From "P. A Dearnley, School of Computing Studies, University of East Anglia, Norwich NR4 7TJ , 1975"***

3. Design for Resilience

When is a database truly resilient? In general, resiliency is really about flexibility. Individual failures, such as server crashes, hardware failures, and network outages are inevitable. However, these inevitable failures don't have to result in system outages. There are many great examples of models that plan for failure such that these individual failures are non-disruptive to the overall system, such as the redundant networks in modern airplanes. Other systems watch global state to proactively reroute activity or replace components that may be failing, which is the model that keeps modern telecom networks running reliably. In the database world, a resilient database is one designed to always be available even as some components fail, so applications continue operating with full access to their data even as failures occur.

It's essential that a database can react to failure efficiently, which is why replicating data to multiple locations is critical to resiliency. In order to be considered a resilient data management solution, the database you choose cannot lose data and must be able to provide access to all data even when individual servers fail. This is particularly important as your organization moves to deployments public clouds, private clouds, and hybrid cloud environments. In cloud deployments, failures are to be expected, they are not exceptions.

If your application is handling critical bank transactions, you need to ensure that your database will provide the level of transactional consistency that your application requires while still providing the resiliency you need. There are three critical components you need to look for when designing for resilience:

1. **Distributed architecture,** the foundation for any system to survive individual failure but remain available.
2. **Simple provisioning and monitoring,** so your system can react to failures efficiently and orchestrate recovery processes without downtime.
3. **Clear model for data replication** in multiple locations and recovery capability in case of failure.

Cloud databases generally provide very good resilience capabilities. However, most cloud databases still have some level of service outage on failure.

Risks of Downtime

When a database fails it can take down core services, applications, and impact customers. That downtime in an always-on world can result in lost revenue, lost opportunity, and disruption across an organization. To protect against data loss against local, site, or region failures, data is generally replicated multiple times in different locations. The performance impact of replicated data to ensure no data loss can be significant with traditional single master databases. A distributed SQL database system, designed to maintain a single logical database across multiple deployments, eliminates these risks because there are multiple consistent copies of the database available in case of failure.

Disaster Recovery Needs

In traditional systems there is a primary location where the database is "active" and updates from that database are replicated from there to other locations. If there's a failure at the primary site, one of those replicas can take over as the active database. Maintaining those replicas is critical to provide disaster recovery for those traditional systems. This model, however, typically involves a period in which the database is unavailable during the time it takes for the failed database to switch over to the replica. In addition, sometimes the replica is missing the latest changes, so some data is lost.

There are a number of options in case of "failover" that result in a faster process, but recovery is still delayed, complicated, and failure prone. A resilient database solution continues operations even in the face of planned or unplanned outages, including hardware and software upgrades, network updates or outages, and service migration or hardware failure. Essentially, a distributed database system designed for resilience is able to continue operations in the face of disaster without needing to introduce complicated and failure prone solutions.

Additional Resiliency & Availability Considerations

Frequently, we talk about **high availability** - most often in terms of "five nines" - or 99.999% - which is only 5.26 minutes of downtime per year. The question, of course, is what gets lost during that downtime? To deliver on demands for high availability, database availability has been handled in a few ways. The first is by running with many replicated servers so that any given server can be taken offline as needed, but there are significant drawbacks to this approach. The second approach is to use a NoSQL database, which makes it simpler to deploy a distributed database in the cloud but places the burden for maintaining consistency on the application. The third option is to use a distributed SQL database, designed for modern architectures.

Redundancy & Replication

One approach to achieving database availability is to run with many redundant, individual, replicated servers. The result is that any given server can fail or be taken offline for maintenance as needed. However, this approach introduces significant operational complexity due to the many independent services running, creates high infrastructure costs, and yet provides no single view of the system. Data is constantly moving between services that weren't designed to handle this kind of coordination, which means that you have to pay close attention to latencies and visibility rules for your applications.

NoSQL Option

The next approach is to use a NoSQL database. In this model, developers need to write each application such that it compensates for the specifics of the availability model and

so they design their service with redundancy. This involves learning a new database model, new programming languages, and rewriting existing applications, all of which incur significant delays and costs as enterprise organizations move existing banking applications to the cloud.

Distributed SQL

A distributed SQL database enables continuous availability through a combination of redundancy and the ability to perform rolling upgrades without requiring downtime. This means that your application and database stay up to date and available, all the time. At the same time, distribution enables performance improvements when compared to existing database infrastructure in the cloud. When choosing a distributed SQL database, here are three critical considerations:

1. Look for a **peer-to-peer architecture** that ensures database services can be natively distributed across multiple nodes, data centers, and clouds without the complexity, expense, and additional software that traditional relational databases require.
2. For always-on applications, choose an **active-active database** that can be deployed on-premises, in the cloud, across clouds, or in a hybrid cloud deployment.
3. Make sure that the database will maintain the **ACID guarantees** essential for critical banking applications while serving data at in-memory speeds.

Expect Failures - from Hardware to Networks to Cloud Providers

Financial services organizations increasingly seek to **take advantage of cloud computing platforms** to achieve business agility, cost savings, improvements to application performance and availability, and to stay on the leading edge of emerging technologies. In spite of all these advantages, it's important to note that there will be failures - it's not a question of if, but when, why, and for how long. Hardware will fail, networks will go down, and **data centers will have outages** - most often due to battery failures, cybercrime, and human error. **Cloud providers have downtime** as well, impacting different services and regions, and Microsoft Azure, Google Cloud Platform, and Amazon Web Services all experience varying amounts of downtime. Although Service Level Agreements (SLAs) require a certain amount of uptime, not meeting the SLA results in a refund for the lost time, but rarely does that refund compensate for the damage caused to the businesses impacted. In addition, the impact of downtime can be significant if your database deployment is limited to a single node in a cloud provider that goes down.

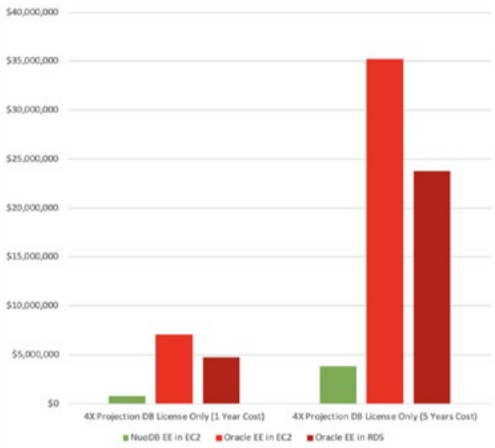
Failures will happen all the time, at every level, and in ways you'd never expect. To avoid single points of failure, you need components, such as containers, that are independent and able to be added as needed in case of failure. By building out a single logical SQL database distributed across multiple nodes, in on-premises data centers, in public and private clouds, and in a combination of on-prem and cloud deployments, you can ensure application availability even in the face of failures.

4. Database Licensing + Cloud Infrastructure Cost Models

Pre-provisioning, or over-provisioning, as covered earlier, results in direct server costs. Those costs cover the ability to scale up to meet peak loads, which means that you're paying upfront for ten times the capacity you typically need, but there are also costs if you design your database but don't have sufficient capacity to meet peak demand. If your database system gets overwhelmed by Black Friday e-commerce transactions, your organization will lose money. In some cases, an overwhelmed database results in a poor user experience, user frustration, and your customers may even migrate to competitors. What you really need is a database system in which you have enough resources deployed to support the load at any given moment in time but can add capacity on demand as needed.

The cost of running enterprise databases is typically the single biggest cost of running data centers. It is very expensive to meet database requirements for:

- Baseline performance
- Peak load performance
- Uptime
- Business continuity & disaster recovery
- Response to new business needs
- Multi-datacenter operation



To address all of these needs, look for a distributed SQL database system designed for modern architectures that allows them to scale out and scale in again on demand.

This scalability enables your organization to lower license costs by paying for peak processing only when needed. When using a distributed SQL database, you can better manage spend by only using as much compute power as needed for the task at hand. Using database models that still rely heavily on hardware, plus the costs of cloud infrastructure, significantly increases costs without improving availability and performance. In addition, those database models don't deliver the same level of benefits offered by cloud native, cloud agnostic databases. When moving critical banking applications to the cloud, it's essential to choose solutions that enable your organization to take advantage of the new capabilities available in the cloud.

Conclusion

While there are many different database options to choose from, choosing the right database architecture for deploying new banking applications and making legacy applications available in the cloud will help you reduce your total cost of ownership while improving customer satisfaction. Financial services organizations face significant disruptions from many fronts, including:

- Changing customer demands and expectations
- Evolving regulations for security, data sovereignty, open banking, and continuous availability
- Competition from smaller, more nimble challenger banks and neo banks

Cloud-native, cloud-agnostic database solutions can help your organization prepare for these disruptions, while also reducing complexity and expense by eliminating pre-provisioning costs, deploying a database designed for resiliency, meeting demanding availability requirements, improving performance, and lowering total costs.

> "Cloud services can facilitate competition and increase the ability of financial services providers to renew their IT systems more efficiently. Improved choice and innovation should deliver commensurate benefits for banks and consumers."
>
> ***- UK's Financial Conduct Authority***

The banking industry is changing dramatically, and it is time to embrace that change. Legacy databases designed for on-premises deployment have served the industry well, but those solutions come with significant costs and limitations. When choosing a database to power critical banking applications, you can choose one that reduces total cost of ownership while enabling your financial services organization to move to the cloud when and how you want.

About NuoDB

Founded in 2010, NuoDB is a cloud-native distributed SQL database funded by market leaders, led by industry veterans, and built on the radical notion that a database should never be what holds your applications or business back.

We've liberated the enterprise-critical database from its inherent limitations, monolithic architecture, and complacent service providers, and replaced it with scale-out simplicity, continuous availability, transactional consistency, and true partnership. So you can take back control of your database, and do more.

NuoDB. The database to build your future on.

To learn more visit **www.nuodb.com**.