NUODB

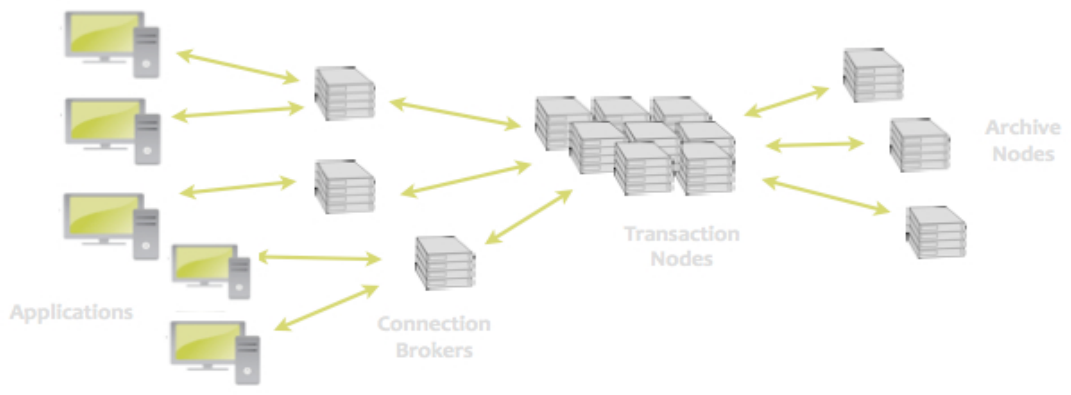
Technical White Paper

Version 3.1

For a long time a few database incumbents and a few open source projects provided sufficiently capable relational database solutions. Almost every data problem fit into a one of these few database solutions. SQL was a common denominator and a vast amount of software was written to use these relational databases. This worked very well until the explosive growth of the Internet took hold and provided companies with an almost infinite amount of data to process. The volume of data pushed the incumbent RDBMS databases beyond their design limits. Companies were forced to build in-house solutions, to innovate on their own.

We have surveyed and learned from the innovation that resulted. We borrow techniques from many different proven solutions. We take those techniques and use them to enable a new SQL RDBMS solution with all the familiarity of yesterday's database but none of the limitations.

NUODB Architecture



EXECUTIVE SUMMARY

NUODB is a relational database; simply connect via JDBC and use SQL as expected. From the perspective of a client application NUODB is no different than any other relational technology, but that's where the similarities end. Unlike centralized, shared state database designs operationally NUODB provides relational database services by running many collaborating processes across any number of machines. Brokers securely connect client applications to their databases. Transaction Nodes interact with clients processing SQL queries. Transaction Nodes run in-memory, and so are not constrained by storage issues. Archive Nodes collect and store all database state to durable storage. They also provide data to Transaction Nodes for processing. To add more capacity simply add more Transaction Nodes. NUODB scales elastically up and down as your needs change.

FOUNDATIONAL TECHNOLOGIES

The fundamental technologies of NUODB are multi-version concurrency control (MVCC), on-demand data replication, and an optimistic asynchronous communication model. Together, these technologies minimize communication cost while providing the full benefits of relational databases - data abstraction and ACID transactions on a cloud of computers.

MVCC

Multi version concurrency control provides lock-free concurrency control at the record level. An update does not replace the stored record and a delete does not remove it. Instead, NUODB manages multiple versions of the record carefully tracking which versions belong to each transactional context. Each transaction reads the most recent version of the record that was committed when the transaction started. Transactions see a stable view of data; except for the changes they make themselves.

MVCC eliminates read/write conflicts by allowing the writer to create a new version without changing the readers' view of data and by allowing a reader access to a stable view of data even if it has been changed by a concurrent transaction.

MVCC eliminates write/write conflicts by restricting transactions so they can modify or delete records only if the most recent version of the records is the one the transaction sees.

MVCC transactions are consistent and require less communication than transactions that use traditional locking or 2PC for concurrency control.

Data Migrates On-Demand

A computer in a NUODB cloud retains only those parts of the database that its application clients require. The data retained changes over time, as the client application's needs change. NUODB has no fixed partitions and no slaved replicants. Data migrates on demand. The distribution of data changes to match activity on the system. Some data may be present on all computers; some may reside only in an archive; some may exist on a few computers. Data distribution shifts to reflect application changes, cyclical changes in requirements, and new applications.

Node Communication

Inside a NUODB database you will find "atoms", self-describing data and metadata that together comprises the "database" – schema, indexes, data. For example, each table is an atom that describes the metadata for the table and references other atoms that describe ranges of records in the table and their versions. Copies of the atom describing a particular table may be present on several computers. When an atom changes that node sends messages to the other nodes that also have copies, replicating the changes. All inter-node messaging is done using asynchronous reliable queues.

NUODB IN DEPTH

MVCC, on-demand replication, and an atomic view of data when properly combined represent a new and complex method for distributed transactional processing in relational database systems. The remainder of this paper describes them in more depth, presents some use cases and operational functions, and ends with a discussion of NUODB and the CAP theorem. Because NUODB introduces a number of new concepts, this section starts by introducing some terms that make it easier to discuss the concepts.

It progresses to a discussion of MVCC, a more in-depth discussion of atom types, and then

Vocabulary

A **node** is a computer, virtual or physical, that will be running processes that together service database requests.

A **communications broker** is a process running on at least one node that provides applications SQL/JDBC access to a NUODB database.

A **transactional node** is a process running on a node that executes the SQL layer operating on database atoms and listening to and communicating changes with its peers.

An **archive node** is a process running on a node that maintains a durable archive of database state on permanent storage, such as a disk drive.

A **chorus** is a group of nodes that together form an operational NUODB database. A database is defined by its SLA, which specifies the kind, quality and quantity of nodes required to be considered available.

A **duet** is the minimum size for a NUODB chorus, consisting of one transactional node and one archive node.

A **multi-tenant** cloud is a set of nodes managing more than one chorus – i.e. a cloud running two or more independent databases. A node can participate in more than one chorus.

The **SQL layer** parses, compiles, and optimizes SQL queries. It runs on transactional nodes. The SQL layer uses the atoms on its node for data, including metadata, and for transaction control.

The **atom layer** controls data access, versioning and concurrency. Every node in the chorus manages atoms by sharing copies with other nodes.

An **atom** is the basic unit of exchange between nodes. The representation of an atom varies to optimize for access in-memory, encoding on the wire, and archival on disk.

Concurrency Control

NUODB uses multi-version concurrency control (MVCC) both to give each transaction a consistent view of data and to prevent concurrent transactions from overwriting each other's changes. This is a new adaptation of the principals behind MVCC, it is distinct from "snapshot isolation". MVCC and snapshot isolation are widely used terms for an alternative to record locking that preserves transactional consistency and isolation.

NUODB virtually eliminates database locking in favor of coordinated distributed version management. Our approach eliminates the need to communicate locking messages between nodes to provide ACID transactions.

Database Consistency

Databases must present consistent views to applications within the context of a transaction. However, a transactional consistent view of the database is relative to the instant the transaction started. A transaction that stays active for an hour continues to see the database as it was at the beginning of the hour. A new transaction sees a different view, even though the two run concurrently.

Communication latency can only delay the recognition of commits on other nodes; it can never make a transaction appear committed when it is not.

ACID

NUODB transactions are **atomic**. All the changes made by the transaction are either made permanent when the transaction commits, or are completely removed from the database.

NUODB transactions are **consistent**. They transform the database from one consistent state to another as defined by the constraints in the schema, explicit (unique constraints, check constraints) or implicit (no alphabetic characters in numeric fields). Each transaction sees a consistent view of the database.

NUODB transactions are **isolated**. No transaction can see changes made by transactions that were not committed when it started – except, of course, its own changes. MVCC also prevents transactions from overwriting changes from concurrent transactions.

NUODB transactions are **durable** with or without disks.

Durability

NUODB handles durability differently from most databases. By default, changes are durable when they are present in memory on at least two nodes, including an archive node, even before that archive has flushed the data to disk. Durability is a matter of degrees; most systems survive a server crash, but not a disk crash on the primary database disk. Replicated systems survive disk crashes, but are vulnerable to data center explosions.

In NUODB, the first line of defense for durability is distributing changes to at least two archive nodes, but that's defined by your Chorus SLA and may not be a requirement.

NUODB archive nodes can write a non-buffered log of replication messages, which will allow the archive node to recover the state of the database at the instant of the crash. Under heavy load, the performance cost of the disk write is significant and should be weighed against other options.

DATA MIGRATES ON-DEMAND

The second core technology of NUODB is partial on-demand replication. Schemas, tables, and data are neither fixed on one node nor present on all nodes. Abstractly, when a node needs data, it asks a peer node that has the data for a copy. While it is using the data, the node keeps it in memory. If it changes the data, it broadcasts replication messages to other copies of the data. When the data is no longer useful, the node releases it, freeing up memory and reducing its message load.

Node Communication

The third core technology of NUODB is inter-node communication to maintain database atoms. When an atom changes, its node broadcasts replication messages to the other nodes maintaining copies of that atom. Nodes in the chorus that do not have copies of that atom do not get those replication messages.

Understanding how partial replication and communication actually work requires understanding the architecture of NUODB in more depth.

NUODB ARCHITECTURE

Layering

From the point of view of a client application, NUODB is very much like all other relational databases. It has a Java/JDBC interface that allows clients to read and write data, create and drop domains, tables, schemas, indexes, etc. Client applications are not aware that the database is not local.

SQL is parsed and executed on transactional nodes, this is where client requests are serviced. Database metadata related to system tables is also represented as atoms. The SQL layer updates system tables with SQL metadata statements that create, alter, and drop metadata objects.

Traditional databases have page caches and I/O subsystems below the SQL layer. NUODB has a data distribution and replication layer that manages manages transactions over atoms. The interaction between the SQL layer and the atoms is primarily related to transaction management, metadata creation and change, and reading and writing records.

Archive nodes

Archive nodes serialize atoms to disk. If a transactional node needs an atom that no longer exists on any other transactional node, it gets the atom from an archive node.

The functions of archive nodes are to write changed atoms to disk, to be the source of last resort when transactional nodes need atoms that do not exist elsewhere in the cloud, and to undo changes made by active transactions on nodes that disappear from the cloud.

Durability, Transaction Commits, and the CAP theorem

Durability is a relative property. What is sufficiently durable for one application may not be durable enough for another. Traditionally, durability in databases has required that all committed data is on permanent stable storage. Unfortunately, disk technology is not 100% reliable, there are many ways in which a single drive might fail.

One way of looking at durability is "how many failures must be survivable?" Most database systems survive a power failure or server crash, but not the loss of a disk holding part of the database. Shadow disk volumes and slaved databases increase the number of survivable failures, but durability in the case of an explosion in the data center requires offsite replication.

NUODB supports several levels of durability, some of which depend on the hardware configuration. At a minimum, before a commit completes, all changes made by a transaction must be in memory in two places, one of which is an archive node. The transaction that wants to commit sends out a pre-commit message. The archive node receives the pre-commit and responds with a commit.

That configuration survives the crash of any one node. Adding more archive nodes increases the number of survivable simultaneous failures, especially if the application has specified that changes must be on two or more archive nodes before a commit completes. Adding a remote archive node in protected data center in a remote location also increases durability with perhaps some cost in latency.

One of the challenges for distributed systems is the CAP theorem, which states that you can have consistency (important), availability (critical), and resistance to network partitions, but not all three. By default, NUODB is consistent and available, but only slightly partition resistant. Because a transaction cannot commit without having all changes on an archive node, any partition that separates transactional nodes from all archive nodes causes those nodes to stall until the partition is resolved. Those transactional nodes that have communication with archive nodes can continue to work.

When transactional nodes disappear from the cloud, the archive nodes dismiss their uncommitted changes. Archive nodes serialize and de-serialize atoms, but the only time an archive node changes the content of atoms is after a transactional node disappears from the chorus. If an archive node recognizes that it has replication messages from a node that has disappeared before its commit messages arrived, the archive node will back-out the changes made by that node.

By configuring "coteries" of nodes, an installation can make its chorus resistant to partitions that include archive nodes on both sides of the cut. A coterie is a set of archive nodes, designed so that there are no disjoint coteries in the chorus. In other words, every coterie shares at least one archive node with every other coterie.

If there are no disjoint coteries, no two coteries can survive a network partition. The coterie that survives completely can continue to work. The others must wait for a network to reconnect. When using coteries, a transaction is not committed until one archive node from each coterie has responded with a commit message.

It is possible that no archive nodes will survive some cataclysmic event. Defining coteries does not guarantee that there will be a surviving sub-chorus, only that there will never be two separate surviving sub-choruses.

If the CAP theorem means that all surviving nodes must be able to continue processing without communication after a network failure, then NUODB is not partition resistant. If partition resistance includes the possibility for a surviving subset of the chorus to sing on, then NUODB refutes the CAP theorem.

CONCLUSIONS

NUODB uses some well-known distributed systems and database techniques in a novel way to create a profoundly different approach to relational database design. Prior to NUODB there existed no SQL database capable of managing data elastically at scale.