Special Relativity
and the
Problem of Database Scalability

James Starkey
NimbusDB, Inc.
The problem, some jargon, some physics, a little theory, and then NimbusDB.
Problem: Database systems scale badly beyond a single computer.

or

How can we get more oomph for more bucks?
**Transaction**: A unit of database work

- **Atomic**: A transaction happens or it doesn’t
- **Consistent**: Logical relationships are preserved
- **Isolated**: A transaction sees only committed data and no partial transactions
- **Durable**: Once committed, it stays committed
Consistency: What does it mean?

1. Transactions are isolated (read-write and write-write)
2. Database constraints are enforced (unique keys, referential integrity, etc.)
3. If you can define it, you can enforce it.
Serializable: A database system in which concurrent transactions have the effect of having been executed one at a time in some order.
**Node**: A computer on a network.
Elasticity: The ability to add or remove a node from a running system.
[ Now, some physics... ]

Newton: A body at rest...

(in other words, a universal reference frame)
Theory of Luminiferous Æther

• Light is a wave
• Waves propagate in a medium
• Ergo “luminiferous æther”
Michelson and Morley: Oops.
Einstein: Observations are relative to the reference frame of the observer.

Theory of Special Relativity, 1905
Serializability: Good idea or bad habit?

- Sufficient condition for consistency
- But not a necessary condition
- Expensive to enforce
- Almost serializable is utterly useless
Serializability: Good idea or bad habit?

Serializable
  Sequential transaction order
  At every point, database has a definitive state

[Gosh, another universal reference frame!]
Some thoughts on time...

• Time is a sequence of events, not just a clock
• Communication, Einstein tells us, requires latency
• Two nodes just can’t see events in the same order
• That’s not a bug, it’s the way it has to be. Deal with it.
Multi-Version Concurrency Control is an alternative to serializability.

- Row updates create new versions pointing to old version(s)
- Each version tagged with the transaction that created it
- A transaction sees a version consistent with when it started
- A transaction can’t update a version it can’t see
- Each transaction sees stable, consistent state
NimbusDB is an elastic, ACID, SQL-based relational database.
NimbusDB modest goals are:

• Elastic, scalable, ACID RDBMS
• Very high performance in data center
• High performance geographically disperse
• Software fault tolerant
• Hardware fault tolerant
• Geological fault tolerant
NimbusDB less modest goals are:

- Zero administration
- Dynamic, self-tuning
- Arbitrary redundancy
- Multi-tenant
- Of, for, and in the cloud
Glossary: A **chorus** is a set of nodes that instantiate a database.
A NimbusDB database is composed of distributed objects called atoms.

- An atom can be serialized to the network or to a disk
- An atom can reside on any number of chorus nodes
- All instances of an atom know about each other
- Atoms replicate peer to peer
- Every atom has a chairman node
Examples of NimbusDB atoms:

- Transaction manager – starts and ends transactions
- Table – metadata for a relational table
- Data – container for user data
- Catalog – tracks atom locations
A NimbusDB chorus has **transactional nodes** that do SQL and **archive nodes** that maintain a persistent disk archive.
NimbusDB communication is fully connected, asynchronous, ordered, and batched
NimbusDB Messaging

- Most data is archival and inactive
- A small fraction is active but stable
- A smaller fraction is volatile but local
- Even less data is volatile and global
- Replicate only to those who care
NimbusDB nodes are autonomous

• A node chooses where to get an atom
• A node chooses which atoms to keep
• A node chooses which atoms to drop
NimbusDB Transaction Control

- A transaction executes on a single node
- Record version based
- A transaction sees the results of transactions reported committed when and where it started
- Consistency is maintained by atom chairmen
- Atom updates broadcast replication messages
- Replication messages precede commit message
NimbusDB is relativistic

- The database can be viewed only through transactions
- Consistency is viewed only through transactions
- There is no single definitive database state
- Nodes may differ due to message skew
- And Dorothy, we’re not in Kansas anymore.
Archive nodes provide durability

• Archive nodes see all atom updates followed by a pre-commit message
• An archive broadcasts the actual commit message
• Transactional nodes retain their “dirty” atoms until an archive node reports the atoms archived.
• Multiple archive nodes provide redundancy
Transactional nodes provide scalability

- Any transactional node can do anything
- Connection broker can give effect of sharding
- Transactional nodes tend to request atoms from local nodes
- Data dynamically trends toward locality
And the little stuff...

- Semantic extensions (shirts are clothes but not pants)
- Unbounded strings (punch cards are oh so yesterday)
- Unbounded numbers
- All metadata is dynamic
- Members of the chorus are platform independent
- And software updates are rolling
- Goal: 24/365 and beyond
Network partitions, CAP, and NimbusDB

• Certain archive nodes are designated as **commit agents**
• Subsets of commit agents form into **coteries**
  (Coteries: subsets where no two are disjoint)
• A pre-commit must be received at least one commit agent in every coterie to commit
• Post partition, the partition that contains a coterie survives
• If a pre-commit was reported to the partition, it commits
Questions, comments and brickbats?