

NoSQL + MySQL

A MySQL Practitioner's Journey to Understanding and Using NoSQL

Boston MySQL Meetup – June 14, 2010

Alex Esterkin

A MySQL Practitioner's Journey to Understanding and Using NoSQL

Who needs NoSQL? Why use something that simplistic?

The Cassandra Syndrome

- Wikipedia: "... a term applied in situations in which valid warnings or concerns are dismissed or disbelieved"

The case for joining You Know Who (Project Voldemort)

NoSQL is not simple: the CAP theorem, NoSQL techniques

Adapting MySQL for NoSQL DB as its persistent store

Configuring/Tuning MySQL for NoSQL query workload

Experimenting with MySQL schema and table design

Taking on Cassandra: use MySQL for range searches, etc.

A MySQL Practitioner's Journey to Understanding and Using NoSQL

Who needs NoSQL? Why use something that simplistic?

The Root Causes of NoSQL

- ▶ Massive data volumes necessitate massive sharding.
- ▶ Extreme query concurrency far exceeding RDMS limits
- ▶ When processing an SQL query, it is not feasible
 - To move data between partitions on multi-gigabyte scale;
 - To assemble or aggregate results from hundreds of shards.
- ▶ ACID transactions do not scale in MPP, cause latency
- ▶ Global transaction management is almost impossible

What Makes NoSQL Work

- ▶ Web Social Networking is naturally partitioned:
 - Each user “bakes” in his/her own data.
 - Query footprint = 1 shard
- ▶ No joins needed
- ▶ No subqueries needed
- ▶ Clients own the data
- ▶ Complex values:
 - Protocol buffers
 - JSON Objects
- ▶ ACID not required



The Essence of NoSQL

- ▶ NoSQL: is it “No SQL” or is it “Not only SQL”? We’ll see.
- ▶ Partitions are little “soldiers” responding to a simple API



A MySQL Practitioner's Journey to Understanding and Using NoSQL

The Cassandra Syndrome

- Wikipedia: “a term applied in situations in which valid warnings or concerns are dismissed or disbelieved”

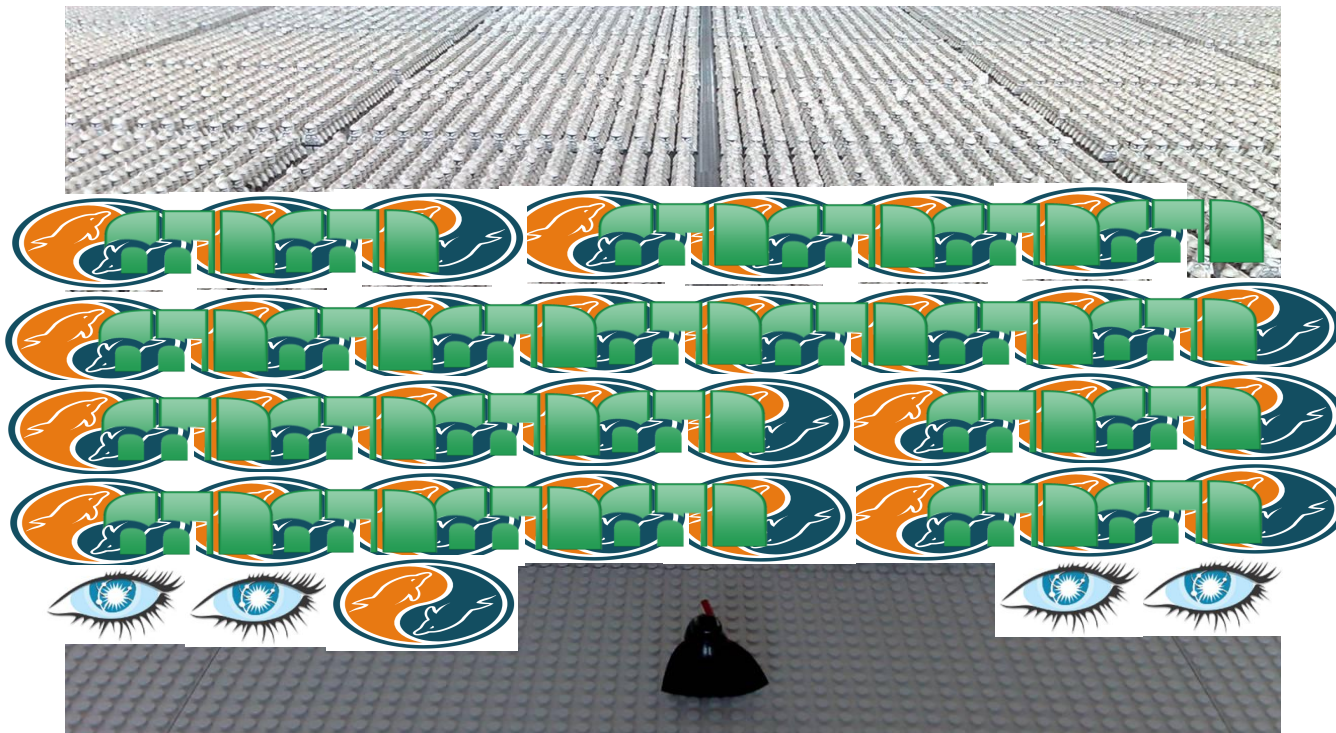
Panel Discussion at the 2010 MySQL Conference: “MySQL or NoSQL...That is the Question”

Moderated by Dr. John Busch, CTO of Schooner Information Technology



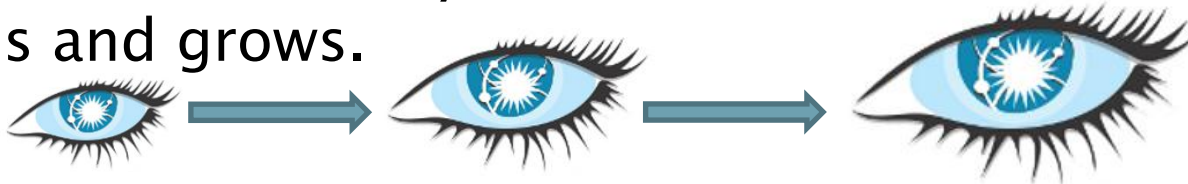
The Role of MySQL: Good Old News

- ▶ Facebook uses MySQL for searches (not as a NoSQL DB)
- ▶ A Year ago, Twitter and Digg still happily used MySQL+Memcached sharded clusters

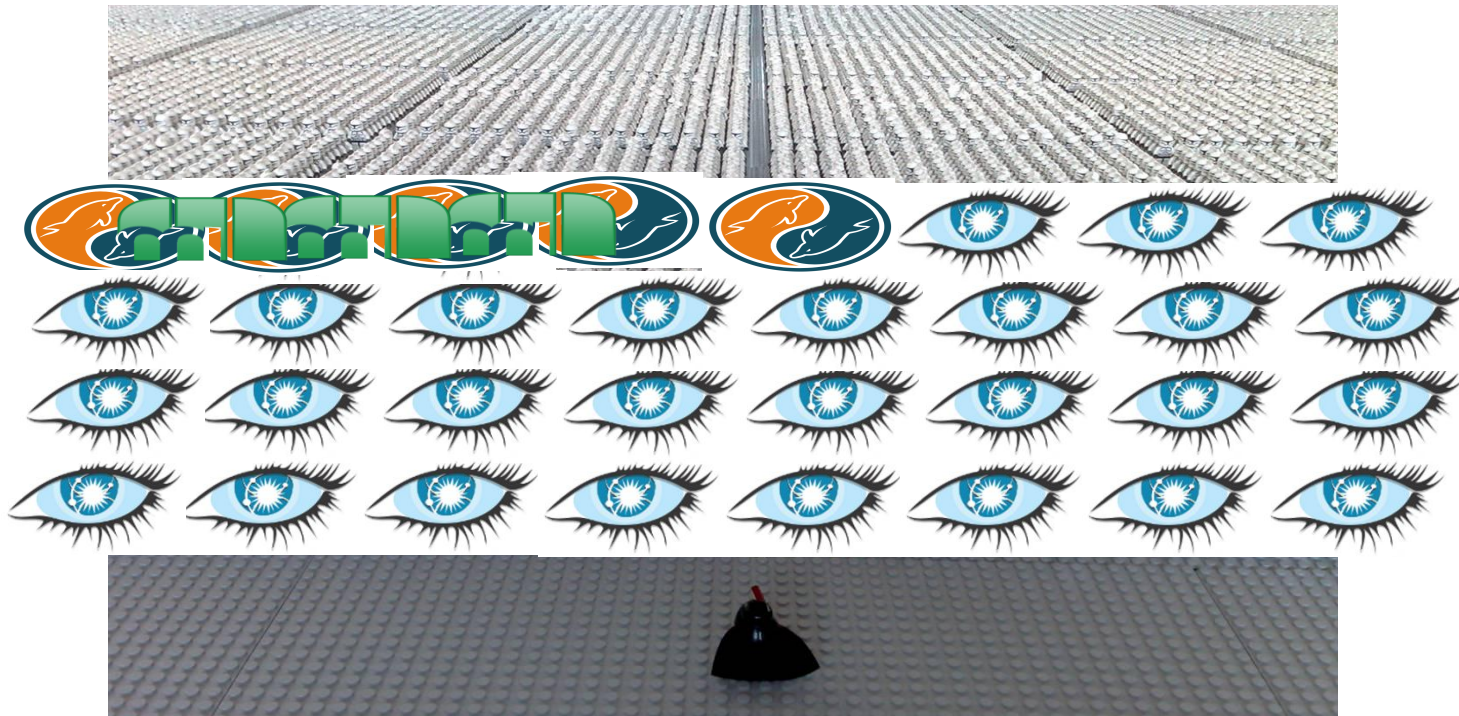


MySQL in Retreat: Bad Recent News

- ▶ Facebook still uses MySQL for searches, but Cassandra evolves and grows.



- ▶ This year, Twitter and Digg converted to Cassandra



A MySQL Practitioner's Journey to Understanding and Using NoSQL

-
-

The case for joining You Know Who (Project Voldemort)

Any Hope for MySQL to Play a Role in the NoSQL Universe?



Many Very Different NoSQL Databases Exist

- ▶ Data models and query APIs in NoSQL databases:

	Data Model	Query API
Cassandra	Columnfamily	Thrift
CouchDB	Document	map/reduce views
HBase	Columnfamily	Thrift, REST
MongoDB	Document	Cursor
Neo4J	Graph	Graph
Redis	Collection	Collection
Riak	Document	Nested hashes
Scalaris	Key/value	get/put
Tokyo Cabinet	Key/value	get/put
Voldemort	Key/value	get/put

Source: Jonathan Ellis Blog, 2009, Rackspace Cloud

Massively Scalable NoSQL Databases are Conceptual Rooted in Dynamo or BigTable

- ▶ Of the listed databases, only Cassandra and Voldemort can scale to petabytes.
- ▶ Cassandra:
 - Borrowed ideas from Oracle, Big Table, and Amazon Dynamo
 - Not a Key-Value store;
 - Cassandra's "column families" and "supercolumns" look very similar to "Nested Tables" in Model 204;
 - There is no place for MySQL in Cassandra.
- ▶ Voldemort:
 - Built on the concepts described in Amazon Dynamo paper;
 - Developed and used by LinkedIn;
 - Based on unconfirmed leaks, is used at Apple.

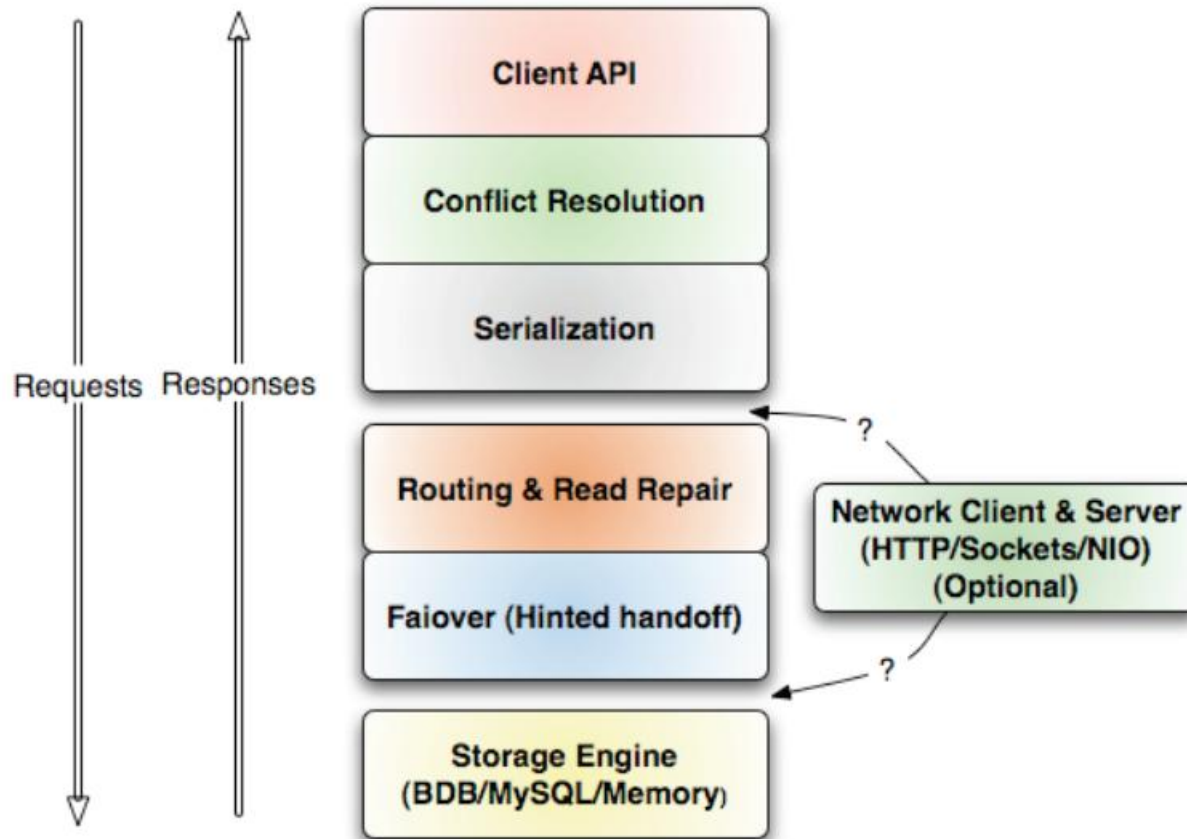
Source: Jonathan Ellis Blog, 2009, Rackspace Cloud

Any Hope for MySQL to Play a Role in the NoSQL Universe? – Yes: Project Voldemort

- ▶ Millions use LinkedIn for professional networking.
- ▶ LinkedIn developed a distributed key-value database based on the principles described in the Amazon Dynamo paper.
- ▶ LinkedIn donated it to open source in 2009 as Project Voldemort.
- ▶ Voldemort has pluggable persistent store architecture that supports BerkeleyDB, **MySQL**, and other data sources.



Voldemort Architecture with Pluggable Storage Engines



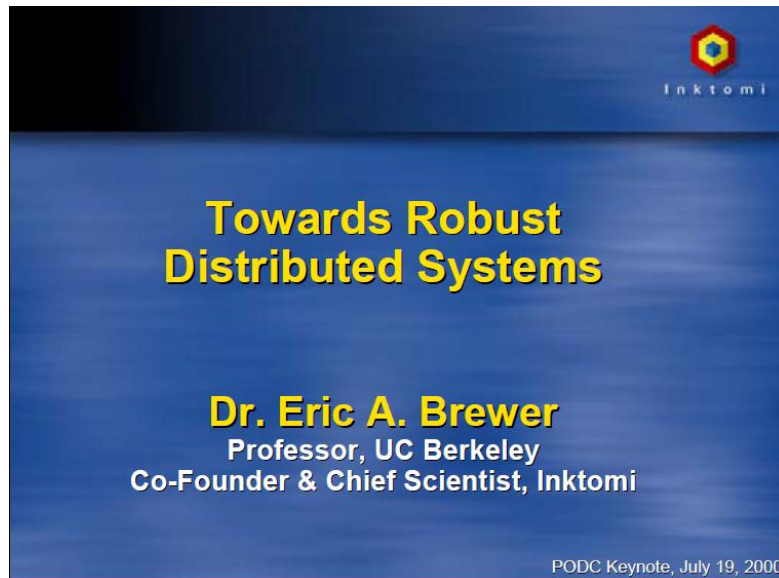
Source: Jay Kreps (LinkedIn and Project Voldemort)

A MySQL Practitioner's Journey to Understanding and Using NoSQL

NoSQL is not simple: the CAP theorem, NoSQL techniques

The Consistency/Availability/Partition Tolerance Theorem

- First presented in the UC Berkeley Prof. Eric Brewer's Keynote Address to the "Principles of Distributed Computing" (PODS) conference in 2000
- Of the three properties of distributed systems—data consistency, system availability, and tolerance to network partition—only two can be achieved at any given time

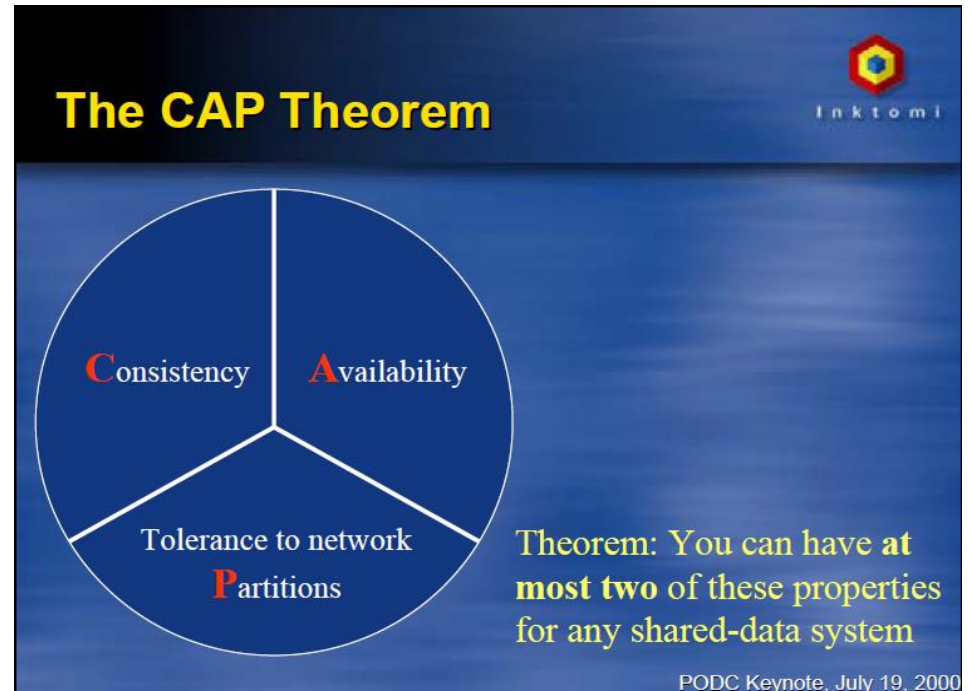


Inktomi

Towards Robust Distributed Systems

Dr. Eric A. Brewer
Professor, UC Berkeley
Co-Founder & Chief Scientist, Inktomi

PODC Keynote, July 19, 2000



Inktomi

The CAP Theorem

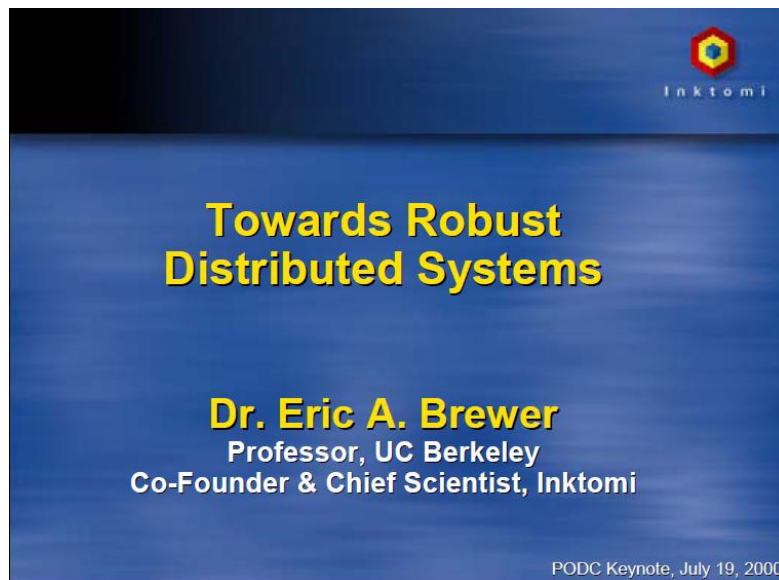
Consistency Availability
Tolerance to network Partitions

Theorem: You can have at most two of these properties for any shared-data system

PODC Keynote, July 19, 2000

The Consistency/Availability/Partition Tolerance Theorem

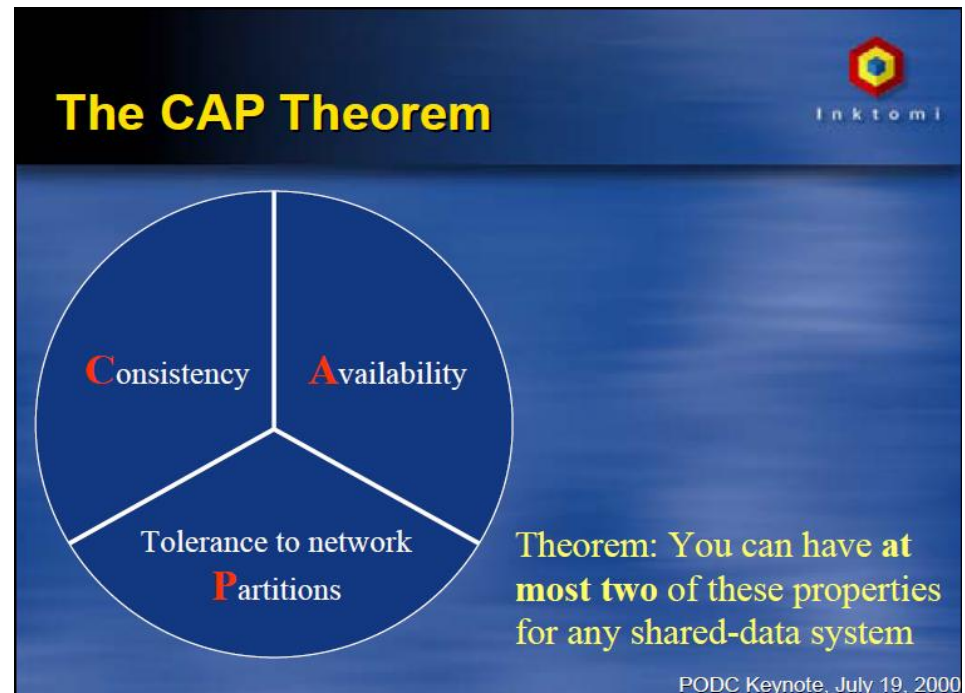
- First presented in the UC Berkeley Prof. Eric Brewer's Keynote Address to the "Principles of Distributed Computing" (PODS) conference in 2000
- Of the three properties of distributed systems—data consistency, system availability, and tolerance to network partition—only two can be achieved at any given time



Towards Robust Distributed Systems

Dr. Eric A. Brewer
Professor, UC Berkeley
Co-Founder & Chief Scientist, Inktomi

PODC Keynote, July 19, 2000



The CAP Theorem

Consistency Availability
Tolerance to network Partitions

Theorem: You can have at most two of these properties for any shared-data system

PODC Keynote, July 19, 2000

- Regarded as proved by MIT Prof. Nancy Lynch and Seth Gilbert in 2002

Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

Seth Gilbert*

Nancy Lynch*

Background: Amazon Dynamo

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Amazon.com

Presented at SOSP'07 October 14–17, 2007, Stevenson, WA, USA

Techniques Used in Amazon Dynamo

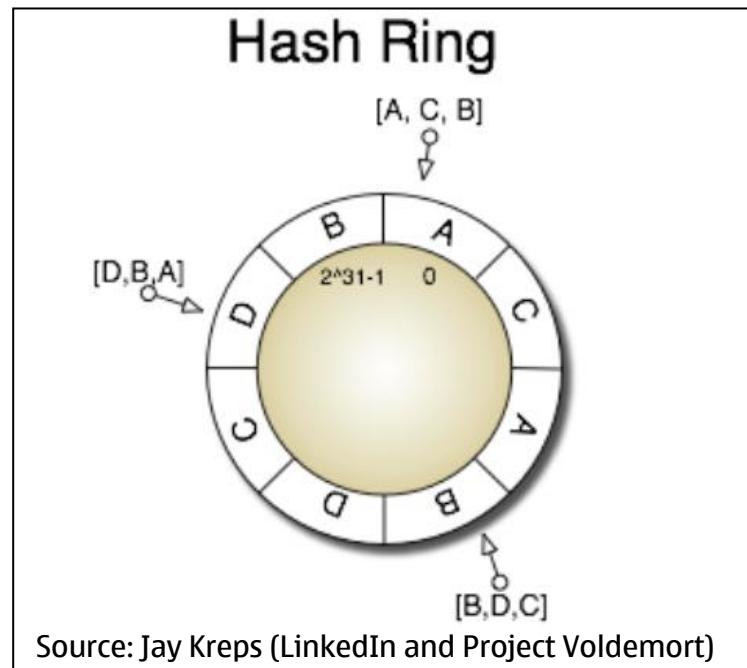
Table 1: Summary of techniques used in *Dynamo* and their advantages.

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Source: Verner Vogels, Amazon

Project Voldemort's Interpretation: Partitioning

- ▶ Key domain is partitioned using consistent hashing:
 $\text{hash}(\text{key}) = \text{shard_id}$
- ▶ Each server node is a “master node” for several partitions
- ▶ Partitions don't change, partition ownership can change
- ▶ Calculate “master” partition for a key
- ▶ Allocate shards in a way that nodes appear in different places on the hash ring



Project Voldemort's Interpretation: Partitioning

- ▶ Parameters: Replication factor N , # of blocking reads R , # of blocking writes W
- ▶ Replicated copies are on different nodes
- ▶ $R+W > N$ allows for quorum resolution
- ▶ Preference list: Next N adjacent partitions in the ring belonging to different nodes

$N-R-W$



- ▶ Thus, “5-3-3” configuration means
 - 5 replicated requests to insert, read, update, or delete key-values
 - A quorum of 3 request results is needed for success
 - Reads and writes both tolerate 2 simultaneous node failures

Entry Time Ordering in Distributed Systems

- ▶ Leslie Lamport (1978). "Time, clocks, and the ordering of events in a distributed system". *Communications of the ACM* 21 (7): 558–565
- ▶ Wikipedia→Lamport Timestamps:
 - In a distributed system, it is difficult to synchronize time across entities within the system;
 - Use the concept of a logical clock based on events through which they communicate.
 - If two entities do not exchange any messages, then they probably do not need to share a common clock; events occurring on those entities are termed as concurrent events.
 - On the same local machine we can order the events based on the local clock of the system.
 - When two entities communicate by message passing, then the send event is said to 'happen before' the receive event, and the logical order can be established among the events.

Project Voldemort's Interpretation: Use Vector Clocks for Versioning

▶ Wikipedia → Vector Clock

- Vector clocks is an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations.
- The vector clocks algorithm was independently developed by Colin Fidge and Friedemann Mattern in 1988

▶ [Jay Kreps (LinkedIn and Project Voldemort)]

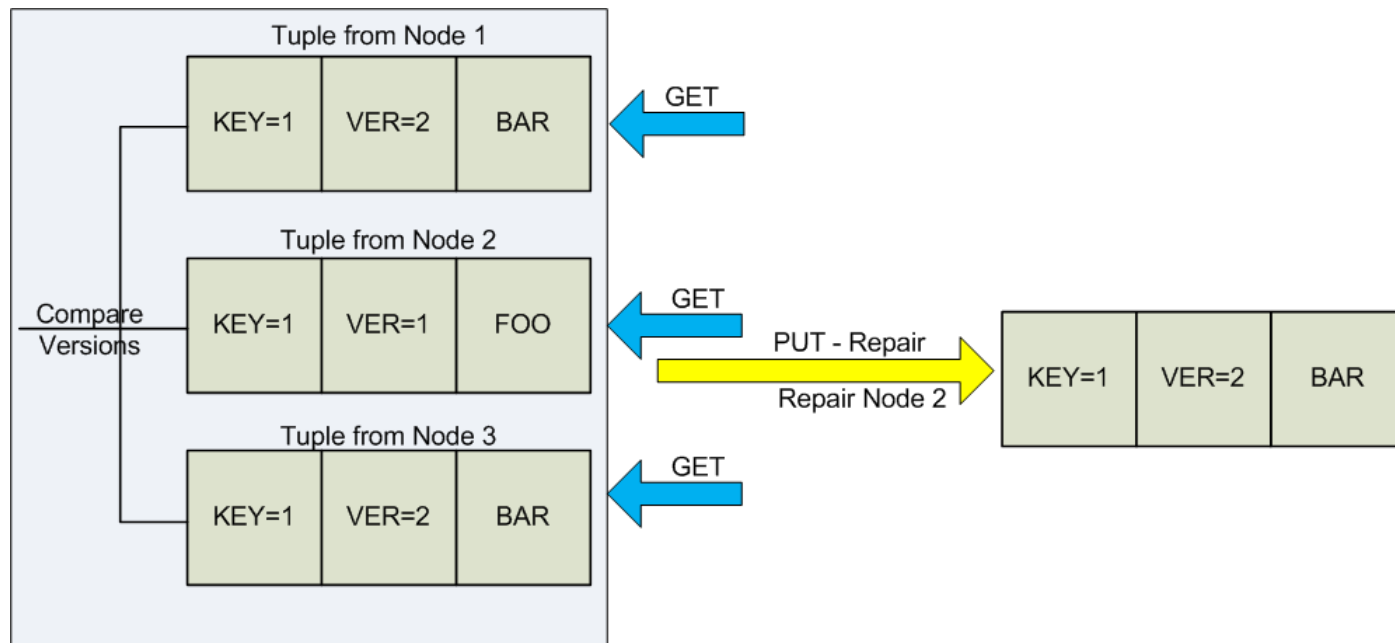
- In Voldemort, a vector clock is a tuple $\{t_1, t_2, \dots, t_n\}$ of counters
 - Each value update has a master node; when data is written with master node i , it increments t_i .
 - All the replicas will receive the same version
 - Allows for consistency resolution between writes on multiple replicas

Concurrent Versions and Inconsistency Resolution

- ▶ In failure or stress scenarios (e.g., network partition), multiple **concurrent versions** of a value may exist simultaneously.
- ▶ Concurrent versions can occur when updates to the same key happen on different nodes in the system.
- ▶ **Inconsistency Resolution** is the process of merging concurrent versions into a single unified value for the client.
- ▶ A simple example of Inconsistency Resolution is "last one wins" –the version with the most recent timestamp is kept.
- ▶ Eventual consistency may be achieved using Read Repair ...

Inconsistency Resolution Technique: Read Repair

- ▶ **Read Repair** is a technique for pushing missed updates to nodes using a lazy consistency protocol (also known as Optimistic replication)
- ▶ When a key is accessed, the version of the value retrieved from each node is compared.
- ▶ Any server that does not have the latest value is updated with the newer value.
- ▶ Read repair is used to make the nodes in the cluster eventually consistent.



A MySQL Practitioner's Journey to Understanding and Using NoSQL

Adapting MySQL for NoSQL DB as its persistent store

Client API

- ▶ Data is organized into “STORES”; typically a “STORE” is a DB table
- ▶ Six operations:
 - PUT (Key, Value)
 - PUT_IF_ABSENT(Key, Value)
 - GET (Key)
 - GETALL(Keys)
 - DELETE (Key, Version)
 - TRUNCATE()

Storage Engine API

- ▶ Each store is backed by an initialized “Storage Engine”
- ▶ On the server side, “Value” becomes a “Versioned” Value – comprised of a Vector Clock (Version) and an actual Value
- ▶ Storage Engine API:
 - PUT(Key, Versioned Value) – same as PUT(Key, Version, Value)
 - PUT_IF_ABSENT(Key, Versioned Value) – same as PUT_IF_ABSENT(Key, Version, Value)
 - GET (Key)
 - GETALL({Keys})
 - DELETE (Key, Version)
 - TRUNCATE()
 - CLOSE()

What Does This All Mean for a MySQL Store?

- ▶ Each record must include KEY, VERSION, and VALUE
- ▶ Voldemort uses binary type KEYS
- ▶ VERSION needs to be a binary type, as it is a complex variable length structure that needs to be serialized
- ▶ VALUE is a binary large object
- ▶ (KEY, VERSION) tuples are either unique or primary key
- ▶ If the storage engine stores BLOBs as files, then too many files and too many full cycle I/O operations – better to incorporate in DB file format
- ▶ No need for global transaction management
- ▶ No need for XA transactions
- ▶ Perhaps, no need for nested transactions? It depends (will discuss later)
- ▶ No need for table locking in case of a transactional MySQL storage engine – fully delegate transaction management to the Engine

MySQL as a Persistent Store for Key-Value Records

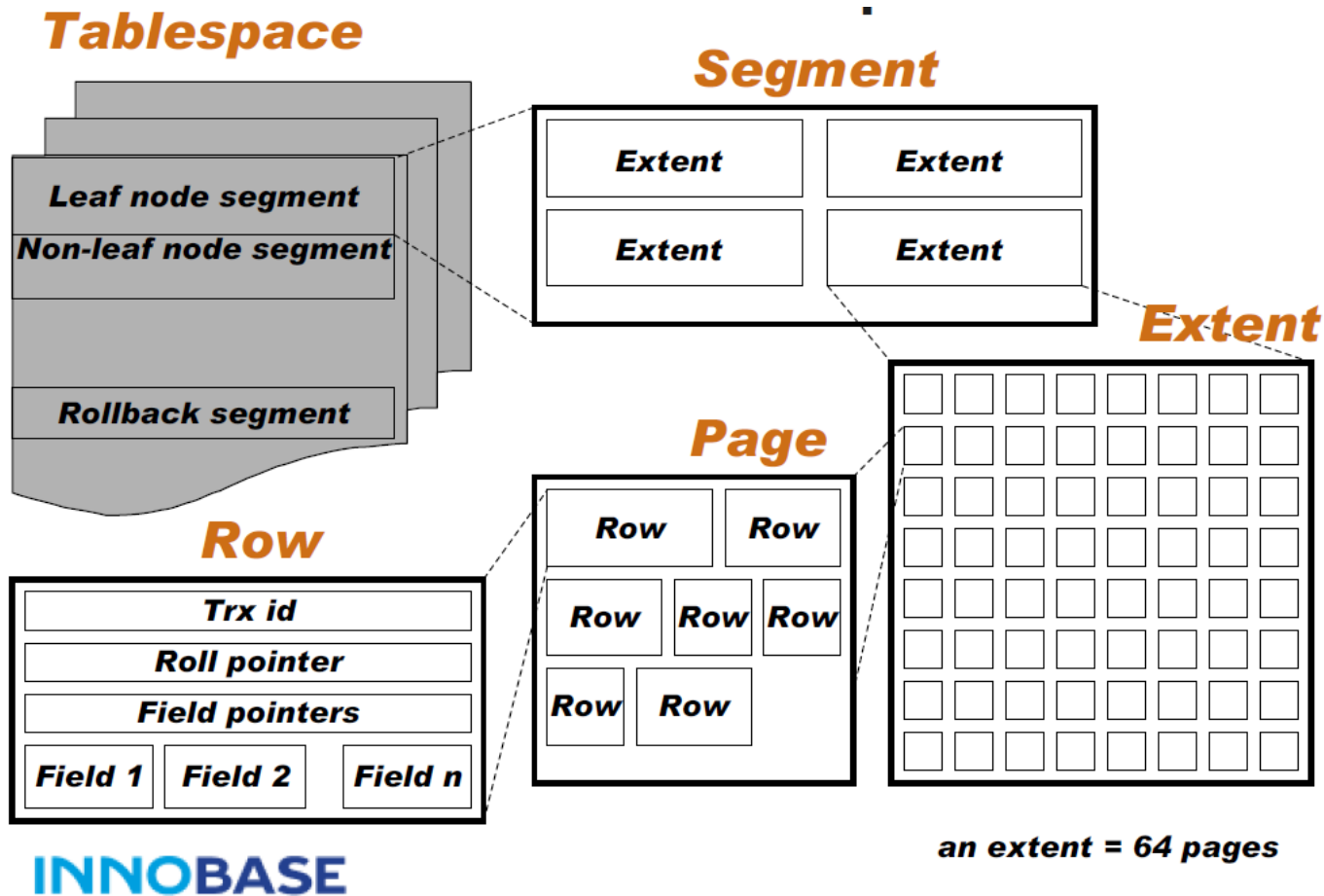
- ▶ Completely different SQL query processing focus:
 - Trivial queries against a single table, no joins, no subqueries, only one WHERE clause predicate – a lookup by a key or a primary key
 - The queries are always the same, but the query result sets are never reused
 - Result sets typically consist of a single record
 - Every record contains a BLOB
- ▶ Standard performance tuning recommendations are only partially valid
 - For example, with no complex query processing on a thread, you can benefit by configuring MySQL to use significantly more threads
- ▶ Standard benchmark used for tuning are not quite applicable to a key-value persistence store

Is InnoDB MySQL Engine Suitable for This?

- ▶ The answer is YES:
 - InnoDB is transactional, with full support for transactions
 - Row level locking
 - MVCC
 - Extensive and reliable crash recovery features
 - You can configure it to use a file per table
 - BLOBs are accommodated very efficiently within the file format
 - Generally concurrent transaction oriented and very tunable
 - InnoDB Plug-in with Google performance fixes handles massive concurrency levels
 - The entire record is stored on one page

InnoDB Page / Physical Storage Organization

- ▶ The following diagram is from '09 MySQL Conference InnoDB presentation by Heikki Tuuri and Calvin Sun; also featured in the MySQL University InnoDB internals talk by Calvin Sun



InnoDB Record Storage Formats

See online InnoDB Plug-in Documentation:

- ▶ http://www.innodb.com/doc/innodb_plugin-1.0/innodb-file-format.html
- ▶ http://www.innodb.com/doc/innodb_plugin-1.0/innodb-row-format.html

- ▶ Question: what storage format is more efficient? COMPACT? DYNAMIC? Antelope? Barracuda?

- ▶ It is more important to minimize the number I/O operations (entire record fits on one page) than to optimize index based query processing operations

Production Challenges for NoSQL Databases

- ▶ Putting a global private cloud based storage system in production
 - Totally different from a stateless service
 - Totally different from single server/cluster or even grid based systems
 - Backup and restore
 - Upgrades
 - Monitoring, provisioning, de-provisioning
 - Capacity planning
- ▶ Performance tuning
 - Performance is deceitfully high when data is in RAM
 - Need realistic tests: production-like systems, clusters, data, and load
- ▶ Operational advantages
 - No single point of failure
 - Predictable SLAs

A MySQL Practitioner's Journey to Understanding and Using NoSQL

Configuring/Tuning MySQL for NoSQL query workload

Tuning Connector/J JDBC Connections for Voldemort

▶ Sample settings that definitely make sense:

cachePrepStmts=true

cacheResultSetMetadata=true

prepStmtCacheSize=1024

useServerPrepStmts=true

cacheServerConfiguration=true

useLocalSessionState=true

enableQueryTimeouts=false

elideSetAutoCommits=true

alwaysSendSetIsolation=false

tcpKeepAlive=true

tcpTrafficClass=24

autoReconnect=true

autoReconnectForPools=true (To preserve server side PreparedStatements)

maxReconnects=216000 (Reconnects are attempted every 2 seconds)

allowMultiQueries=true

maxAllowedPacket=134217728 (Ability to store values up to 128M in size)

Turn Off MySQL Table Locking and XA Support

`skip-external-locking`

`skip-locking`

`innodb_support_xa = false`

This is almost a no-brainer:

- External locking makes no sense, as we:
 - use eventual consistency mechanisms,
 - employ only InnoDB storage engine,
 - don't use global transaction manager
- Regular table locking can also be turned off:
 - we only use statement level transactions
 - since InnoDB provides row level locking and MVCC

Do We Really Need Binary Logging?

- ▶ Arguably, binary logging is not needed, as in case of server crash, we can rely on eventual consistency mechanisms, such as read repair, for data recovery
- ▶ Arguments in favor of turning OFF binary logging:
 - Much better INSERT, REPLACE, and UPDATE performance in terms of throughput
 - In our tests, throughput (# requests per second) we have seen 2–3X difference
- ▶ Arguments in favor of turning ON binary logging:
 - Absolutely required to provide “point in time” recovery
 - In case of InnoDB file corruption, restoring from backup results in stale data, eventual consistency mechanisms might take too long to close the gap
 - In case of high update/insert/delete request volume or percentage, the shards containing most recent record replicas may become single points of failure
 - Binary logging tangibly increases the probability that InnoDB may automatically recover after a server crash
- ▶ A valid question to ask: Which of two options will provide better performance:
 - Configure Voldemort cluster replication as 3–2–2 AND turn binary logging ON
 - Configure Voldemort cluster replication as 5–3–3 AND turn binary logging OFF (in this case, there will be no shard level single points of failure)

Reduce Transaction Isolation Level, etc.

- ▶ The default transaction isolation level may be set to one of the following levels: READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, SERIALIZABLE
- ▶ As we rely on Eventual Consistency, we can set this to the lowest level possible
- ▶ Binary Logging necessitates at least READ-COMMITTED
- ▶ Therefore:
 - If Binary Logging is turned OFF, then use
transaction_isolation = READ-UNCOMMITTED
 - If Binary Logging is turned ON, then use
transaction_isolation = READ-COMMITTED
- ▶ In either case, transactional consistency guarantees can be weakened:
innodb_flush_log_at_trx_commit = 0
 - If set to 1, InnoDB will flush transaction logs to the disk at each commit, provide full ACID behavior
 - Value 0 means that the log is only written to the log file and the log file flushed to disk approximately once per second
 - Value 2 means the log is written to the log file at each commit, but the log file is only flushed to disk approximately once per second

Make Sure There are Enough Connections and Threads

thread_cache_size

- ▶ The minimum rational setting is the number of NIO channels plus 2.
- ▶ Performance should improve with higher settings, e.g., 4x (# of NIO channels)

thread_concurrency

- ▶ It makes sense to set it higher than thread_cache_size, as, presumably, threads spend much time sleeping while InnoDB does the nutty-gritty work

max_connections

- ▶ Always set this to a high enough number – up to 1000 or more

wait_timeout = 432000 (This is a sample setting)

- ▶ The shown setting allows for client auto-reconnecting for up to 120 hours.
- ▶ This server side setting corresponds to maxReconnects MySQL Connector/J setting on the JDBC client side.
- ▶ The JDBC client tries to reconnect every 2 seconds (this time slice is not configurable). We need 1800 reconnect attempts to last us through one hour.

Make Sure There are Enough InnoDB Threads

`innodb_thread_concurrency`

- Number of threads allowed inside the InnoDB kernel. Arguably, this value should be greater than *thread_pool_size*
- Although the documentation states that too high value may lead to thread thrashing, on a system with 8 cores, 64 seems to be a good number to try

`innodb_write_io_threads=16` or maybe even higher

`innodb_read_io_threads=4` or 8

- In case of a key-value database, there is no value in pre-fetching data, hence, we don't need too many read threads

Maximize InnoDB Memory, Shrink Unused Memory Areas

- ▶ On a hardware server with 16GB of main memory, try setting `innodb_buffer_pool_size = 10240M`
- ▶ The following memory is allocated by MySQL per thread and is completely wasted in our case, therefore, we should use the lowest settings:
 - `join_buffer_size` – use the lowest allowable value, as we have no joins to process
 - `key_buffer_size` – use the lowest allowable value, as we have no MyISAM indices
 - `read_buffer_size` – set to a low value, e.g., 1M, as we do not scan MyISAM tables
 - `read_rnd_buffer_size` – set to a low value, e.g., 1M, as we do not read MyISAM sorts
 - `bulk_insert_buffer_size = 0M` – we do not insert data into MyISAM tables
 - `myisam_sort_buffer_size` and `myisam_max_sort_buffer_size` – set to the lowest values

Other Performance Tuning Tweaks to Try on Linux

[mysqld_safe]

nice = -20

malloc-lib = tcmalloc

- ▶ Try giving MySQL higher priority on a dedicated hardware server
 - Pitfalls: May deadlock MySQL server or even crash the hardware server if MySQL settings are too aggressive
 - May make performance worse, unless Voldemort JVM is given same priority
- ▶ Thread-Caching Malloc library – one of Google performance fixes
 - Pitfalls: Makes a difference only when the number of threads is fairly low or when the value BLOBs are large
 - With BLOB values and very high concurrency of requests, configuring InnoDB with a lot of threads and not using tcmalloc library might provide better performance results

[mysqld]

memlock

- ▶ Instructs MySQL to use the memlock() function call to keep MySQL process locked in memory and to avoid potential swapping out in case of high memory pressure.

A MySQL Practitioner's Journey to Understanding and Using NoSQL

Experimenting with MySQL schema and table design

Want to Make it Better? Developers Welcome

- ▶ Project Voldemort uses Apache License
- ▶ Want to roll up sleeves? Here are some ideas.
 - Improve the existing MySQL persistent store implementation or come up with a better one (a JDBC application):
 - Rewrite connection pooling (the build-in version uses a simple *org.apache.commons.dhcp.BasicDataSource* based pooling design).
 - Make a more efficient use of prepared statements.
 - Experiment with SQL schema, table, and transaction design:
 - Use a different PRIMARY KEY, e.g., an AUTO_INCREMENT field.
 - Add extra fields to speed up queries, e.g., for searching by “key_”.
 - Replace transaction blocks with Java tricks.
 - Exotic idea: put each shard in a separate table.

◦

In Search of a Better Alternative to Replace Commons DBCP Connection Pooling

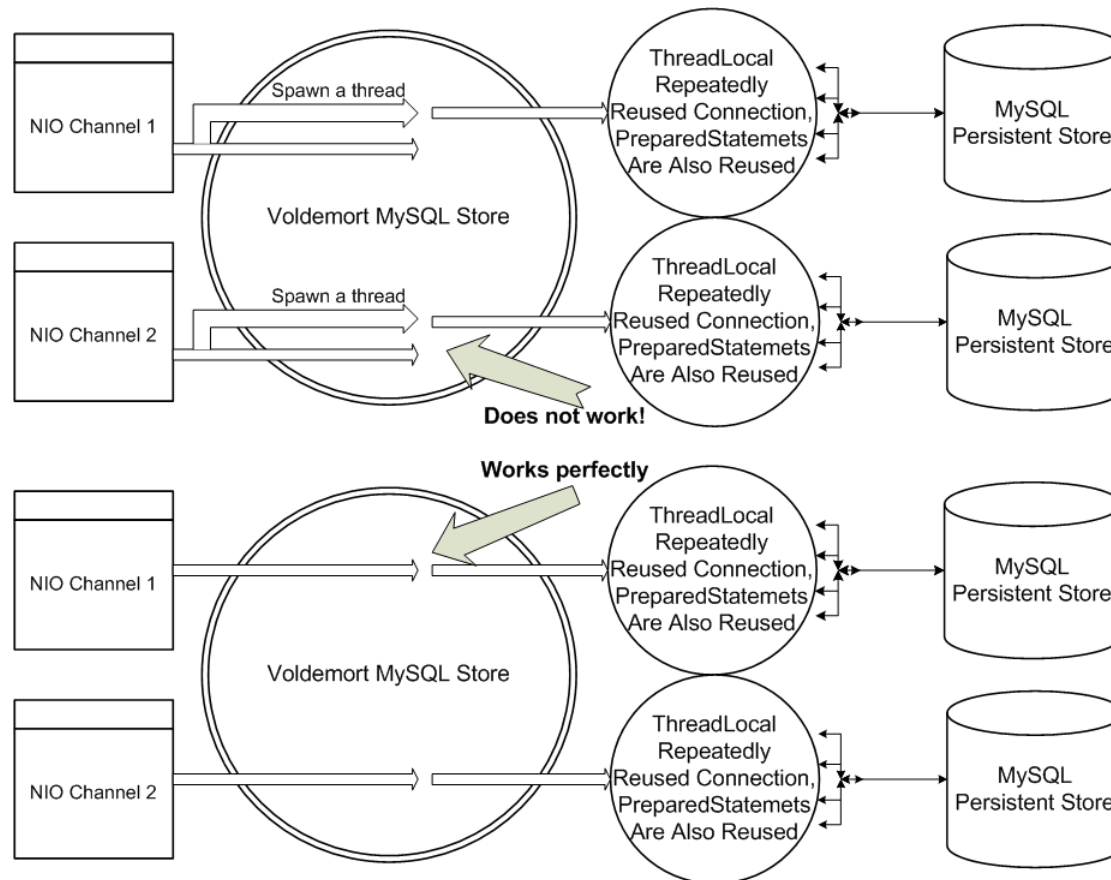
- ▶ Connection pool variants from Apache Commons DBCP
 - A lot of locking and blocking
 - Multi-level “DELEGATE” pattern causes data copying, inevitably too many try-catch blocks, lacking *PreparedStatement* reuse, difficult to configure.
- ▶ Alternatives:
 - ThreadLocal* wrapper containing a JDBC Connection container object
 - In each instance of a MySQL pluggable persistence store object, allocate a *ThreadLocal* container object that wraps around a *java.sql.Connection*.
 - There will be one JDBC Connection container per Java NIO thread
 - Create, prepare and save as class variables all the necessary *PreparedStatement* objects; reuse these Connections and Prepared Statements

Develop own implementation of connection pooling

- Make sure prepared statements are really reused and not closed unnecessarily
- Take advantage of Connector/J features
- Make everything non-blocking and non-locking as much as possible

In Search of a Better Alternative to Replace Commons DBCP Connection Pooling

- ▶ A ThreadLocal wrapped JDBC Connection container works the fastest
- ▶ However, it assumes that all processing is done “inline” on the respecting NIO thread and that NIO threads block on DB access without spawning new threads



In Search of a Better Voldemort MySQL Store Database, Table, and Transaction Design

- ▶ The built-in MySQL persistence store uses the following table definition:

```
create table mysql
( key_          varbinary(200) not null,
  version_     varbinary(200) not null,
  value_       blob,
  primary key(key_, version_)
) engine = InnoDB
```

- ▶ PUT operations: AUTOCOMMIT is off, each transaction blocks contains
 - A SELECT by key_ to see if this might be in fact an UPDATE or a read-repair
 - Iterate over the Result Set, DELETE records with verifiably earlier version_ field values
 - INSERT a new record
 - COMMIT or ROLLBACK
- ▶ Index operations are slow because of PK width
- ▶ Slow DELETES of individual records while iterating result sets
- ▶ Maximum size of BLOB values: 65535 bytes

More Unusual Ideas: Table Per Shard

- ▶ Idea: why not store each shard in a separate MySQL table
<shard_prefix>_<shard_id>
- ▶ “Table per shard” positives:
 - Better I/O parallelism, lower disk contention, could use more I/O threads in InnoDB.
 - More granular backup, restore, easier to migrate shards to another node as part of Voldemort cluster rebalancing
 - Smaller database file corruption footprint, should there be file corruption after a crash
- ▶ “Table per shard” negatives:
 - All pooled connections have to process database requests against all tables.
 - At least *times (number of shards per node)* more Prepared Statements.
 - Everything is a federation, single administrative commands do not suffice any more.
 - For example, in order to drop all tables in a cluster with 32 shards and a common Voldemort table prefix “*voldemort*”.

```
for i in `seq 0 32`; do echo "drop table voldemort$i" | mysql -ujason; done
```
 - Same applies to TRUNCATE, BACKUP, etc.

A MySQL Practitioner's Journey to Understanding and Using NoSQL

Taking on Cassandra: use MySQL for range searches, etc.

Idea: MySQL could help Voldemort to Implement range searches

- ▶ Cassandra is not only a key–value store:
 - Supports “subkeys”
 - Allows range searches (by using order–preserving partitioners).
- ▶ Idea: develop such features in Voldemort using MySQL means:
 - Add necessary extra MySQL fields and indices.
 - Design and contribute corresponding Voldemort API extensions.

Idea: MySQL could help Voldemort to Close any Functional Gaps relative to Cassandra

Adding new features to a NoSQL database will be easy when a full-featured RDBMS is used for data persistence!



A MySQL Practitioner's Journey to Understanding and Using NoSQL

Questions?

Alex Esterkin

aesterkin@gmail.com

- Presently, Database Technologist at Nokia.
- Prior to this, Chief Architect at Infobright.
- Developed other database server products:
 - Postgres-derived MPP database server grid at Dataupia;
 - SybaseIQ analytical DW server ant Expressway Technologies/Sybase.
 - SQL features of Model204 at CCA.