Web Scale Data Management: An Historical Perspective

SYSTEMS RESEARCH AT HARVARD

Margo Seltzer

October 29, 2013

Outline

- In the beginning …
- The heyday of RDBMS
- The rebirth of key/value stores
- Key/Value stores today: NoSQL
- NoSQL & key/value use cases

In the Beginning

- Where beginning equals 1960's
- Computers
  - Centralized systems
  - Spiffy new data channels let CPU and IO overlap.
  - Persistent storage is on drums.
  - Buffering and interrupt handling done in the OS.
  - Making these systems fast is becoming a research focus.
- Data
  - What did data look like?

Organizing Data: ISAM

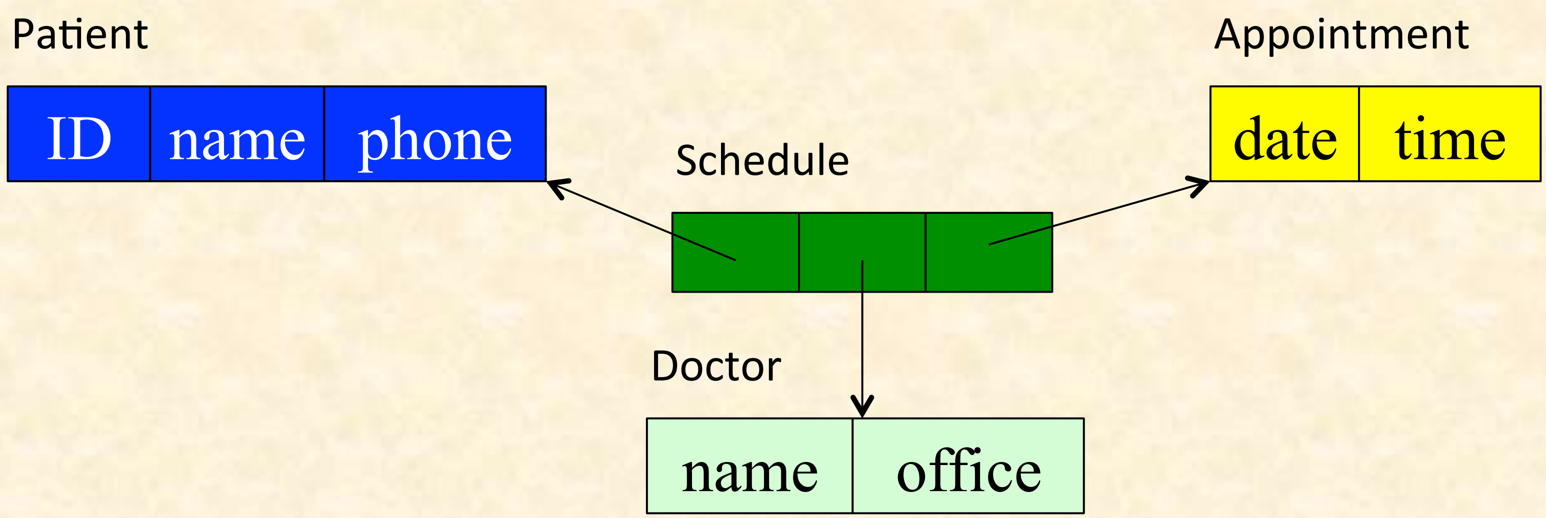
- Indexed sequential access method
  - Pioneered by IBM for its mainframes
  - Fixed length records
  - Each record lives at a specific location
  - Rapid record access even on a sequential medium (tape)
- All indexes are ‘secondary’
  - Allow key lookup, but …
  - Do not correspond to physical organization
  - Key is to build small, efficient index structures
- Fundamental access method in COBOL

Organizing Data: The Network Model

- Early data management systems represented data using something called a network model.
- Data are represented by collections of records (*today we would call those key/data pairs*).
- Relationships among records are expressed via links between records (today we can think of those links as pointers).
- Applications interacted with data by navigating through it:
  - Find a record
  - Follow links
  - Find other records
  - Repeat

The Network Model: Inside Records

- Records composed of attributes (think fields)
- Attributes are single-valued.
- Links connect exactly two records.
- Represent N-way relationships via link records



The Relational Model: The Competition

- The Network Model had some problems
  - Applications had to know the structure of the data
  - Changing the representation required a massive rewrite
  - Fundamentally: *the physical arrangement was tightly coupled to the application and the application logic*.
- 1968: Ted Codd proposes the relational model
  - Decouple physical representation from logical representation
  - Store "records" as "tables"
  - Replace links with implicit joins among tables
- The big question: could it perform?

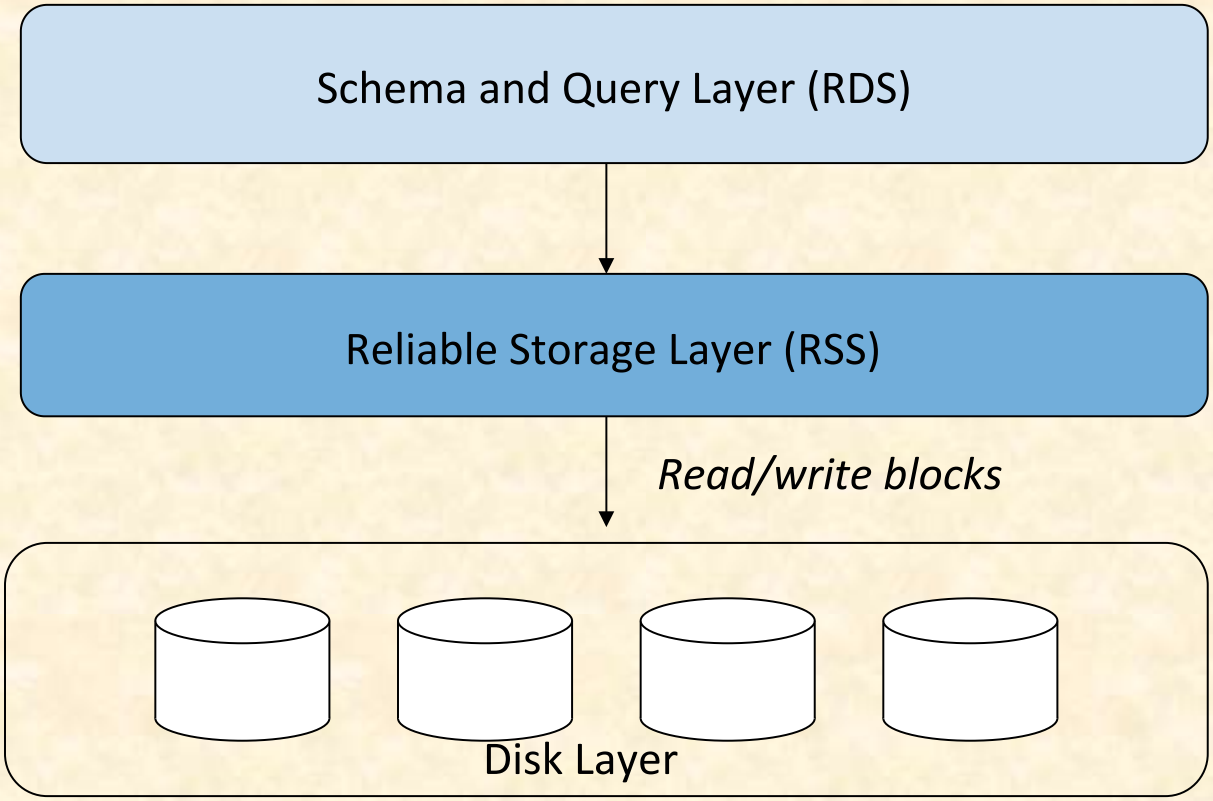
Outline

- In the beginning …
- The heyday of RDBMS
- The rebirth of key/value stores
- Key/Value stores today: NoSQL
- NoSQL & key/value use cases

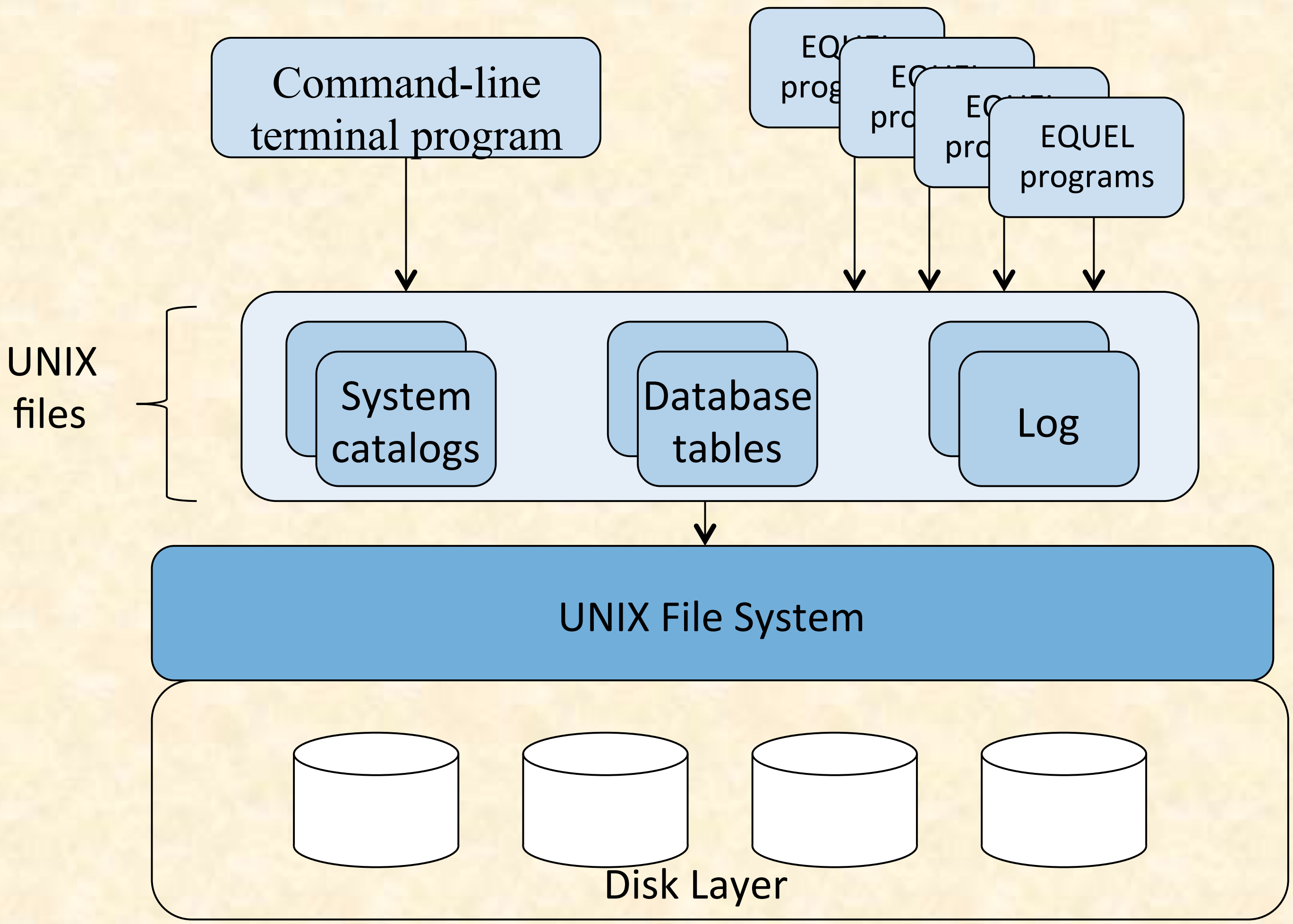
The heyday of RDBMS

- After Codd's paper, much debate ensued, but two groups set out to turn an idea into software.
  - IBM: System/R
  - U.C.Berkeley: Ingres
- Both were research projects to explore the feasibility of implementing the relational model.
- Both were hugely successful and had enormous impact.
- However, at their core, you'll notice something interesting …

The Design of System R



The Design of Ingres



The Key/Value Store Within

- All these relational systems and most data management systems have hidden deep inside some sort of key/value storage engine.
- For years, the “conventional wisdom” was to hide those KV stores deep underneath a query language and schema level.
- Major exception was COBOL, which continued to use ISAM.

RDBMS matured

- Lots of new features (SQL-86, SQL-89, SQL-92, SQL: 1999, SQL:2003, SQL:2008, SQL:2100?)
  - Triggers
  - Stored procedures
  - Report generators
  - Rules
  - …
- Incorporated new data models:
  - Object-relational systems
  - XML
- Became enormously large, complex systems requiring expert administration.

RDBMS: The Good and the Bad

- RDBMS Advantages:
  - Declarative query language that decoupled physical and logical layout.
  - (Relative) ease of schema modification.
  - Great support for ad hoc queries.
- RDBMS Disadvantages:
  - Pay for the overhead of query processing even if you don't have complex queries.
  - Require DBA for tuning and maintenance.
  - Nearly always pay IPC to access database server.
  - Require schema definition.
  - Not great at managing hierarchical or other complex relationships.

Outline

- In the beginning …
- The heyday of RDBMS
- The rebirth of key/value stores
- Key/Value stores today: NoSQL
- NoSQL & key/value use cases

The Age of the Internet

- New classes of applications emerged
  - Search
  - Authentication (LDAP)
  - Email
  - Browsing
  - Instant messaging
  - Web servers
  - Online retail
  - Public key management

These Applications were Different

- They weren't supporting ad hoc query mechanisms
  - Query set specified by the application
  - Users interacted with the applications not the data
- They had fairly simple schemas
- No need for fancy reports
- Performance was critical

All the bells and whistles of the big relational players weren't delivering necessary functionality, but were costing overhead.

The Emergence of Key/Value Stores

- In 1997, Sleepycat Software, introduced the first commercial key value store (now Oracle Berkeley DB).
- Berkeley DB was/is:
  – Embedded: links directly in an application's address space
  – Fast: avoids query parsing and optimization
  – Scalable: runs from handheld to data center
  – Flexible: trade off functionality and performance
  – Reliable: recovers from application and system failure

Key/Value versus RDBMS

	Key-value Databases	Standard RDBMS
Pros	• Sometimes the "good enough" solution for simple data schemas • Much smaller footprint • Shorter execution path • More efficient, fewer OS resources • Usually no client/server • No application data mapping	• Rich SQL Support • Data typing • Data relationship management • Dynamic data relationships • Procedural languages (stored procedures) • Parallel query execution • Security, Encryption • Centralized management
Cons	• Application-centric schema control • High Availability often missing (not BDB) • Higher level abstraction APIs often missing (not BDB) • Limited RDBMS integration	• Much larger engine, higher overhead, client/server overhead, general purpose RDBMS solution • Higher administrative burden • Higher TCO

From local to distributed

- As the web matured, data volume and velocity and customer demand grew exponentially.
- Exceeded single system capacity.
- Enter a new era of scalability demands.
- Infrastructure migrates from large-scale SMP to clusters of commodity blades.

Outline

- In the beginning …
- The heyday of RDBMS
- The rebirth of key/value stores
- Key/Value stores today: NoSQL
  - Framing
  - Technology
- NoSQL & key/value use cases

Enter NoSQL

- Web service providers (e.g., Google, Amazon, Ebay, Facebook, Yahoo) began pushing existing data management solutions to their limits.
- Introduced sharding: Split the data up across multiple hosts to spread out load.
- Introduced replication:
  – Allow access from multiple sites
  – Increase availability
- Relax consistency: Didn't always need transactional semantics

What is NoSQL?

- Not-only-SQL (2009)
- While RDBMs have typically focused on transactions and the ACID properties, NoSQL focuses on BASE:
  - **B**asic **A**vailability: Use *replication* to reduce the likelihood of data unavailability and *sharding* to make any remaining failures partial
  - **S**oft state: Allow data to be inconsistent and relegate designing around such inconsistencies to application developers.
  - **E**ventually consistent: Ensure only that at some future point in time the data assumes a consistent state

Why NoSQL?

- Three common problems
  1. Unprecedented transaction volumes
  2. Critical need for low latency access
  3. 100% Availability
- Hardware shift: From massive SMP to blades
- Changing how we deal with the CAP theorem
  - The CAP theorem states that you can't have all three of:
    - **C**onsistency: All nodes see the same data at the same time.
    - **A**vailability: Every request receives a success/fail response
    - **P**artition tolerance: The system continues to operate despite arbitrary message loss.
  - RDBMs (transactional systems) typically choose CA
  - NoSQL typically chooses AP or CP

NoSQL Evolution

- 1997: Berkeley DB released providing transactional key/value store.
- 2001: Berkeley DB introduces replication for high availability.
- 2006: Google publishes Chubby and BigTable papers.
  - Each system provides scalable, data management for loosely coupled systems
- 2007: Amazon publishes Dynamo paper
  - DHT-based eventually consistent key/value store
- 2008+: Multiple OSS projects to produce BigTable/Dynamo knockoffs
  - HBase: Apache Hadoop project implementation of BigTable (2008)
  - CouchDB: Erlang-based document store, MVCC and versioning (2008)
  - Cassandra: Write-optimized, column-oriented, secondary indices (2009)
  - MongoDB: JSON-based document store, indexing, sharded (2009)
- 2009+: Commercialization
  - Companies emerge to support open source products (e.g., DataStax/Cassandra, Basho/Riak, Cloudera/HBase,10Gen/MongoDB)
  - Companies develop commercial offerings (Oracle, Citrusleaf)

NoSQL Technology

- Some form of distribution and replication:
  - Distributed hash tables (DHTs)
  - Key partitioning
  - Replication in underlying storage or in NoSQL
- Relational or key/value store for underlying storage engine.
- Second generation systems building custom write-optimized engines.
  - Log-structured merge trees (LSM)

NoSQL Design Space

- Local node storage system
- Distribution
- Data Model
- Consistency Model
  - Eventual consistency
  - No consistency
  - Transactional consistency

Local Storage Systems

- Native KVstore
  - Oracle NoSQL Database: Berkeley DB Java Edition
  - Basho: Riak's Bitcask, a log-structured hash table
  - Amazon: Dynamo, BDB Data Store, BDB JE (or MySQL)
- Log-Structured Merge Trees
  - LevelDB
- Custom
  - BigTable (and clones): Tablet servers on SSTables exploiting GFS-like systems.

Distribution

- Three main questions:
  - How do you partition data
  - How many copies do you keep
  - Are all copies equal
- How do you partition data
  - Key partitioning
  - Hash partitioning (often called sharding)
  - Geographic partitioning (also called sharding)

More Distribution

- How many copies and how?
  - Use the underlying file system (GFS, HDFS)
  - Three is good; five is better; for hot things sometimes go to many.
- Copy Equality:
  - Single-Master
    - MongoDB, Oracle NoSQL Database,
  - Multi-Master/Masterless
    - Riak, CouchDB, Couchbase

NoSQL Data Models

- Common to most offerings:
  - Denormalization: some data values are duplicated to provide superior query performance.
  - No joins
- Key/Value: Little or no schema, minimal range query support
  - Oracle NoSQL DB, DynamoDB, Couchbase, Riak
- BigTable Column Family
  - Hbase, Cassandra
- Document-stores
  - MongoDB, CouchDB

Data Model: Key/Value

- Keys and data are both opaque byte strings.
- Designed for point queries
  - Typically no notion of a range query
  - If iteration exists, it's usually not in key-order
- Examples:
  - DynamoDB
  - LevelDB
  - Riak
  - Tokyo/Kyoto Cabinet
  - Oracle NoSQL Database
- Extensions:
  - Oracle NoSQL: Major/Minor keys, batching
  - LevelDB: Atomic batches

Data Model: Column Family (1)

- Columns are grouped into *column families*.
- Column families:
  - Are typically stored together
  - Can have different columns for each row
  - Can have duplicate items in any column
- No schema or type enforcement
  - All data are treated as byte strings
- Indexed by rows
  - Rows are grouped into tablets
  - No secondary indexes

Data Model: Column Family (2)

Column Families

Timestamps

versions

Keys	Name		Address					
1	*First* Margo	*Last* Seltzer	*No* 3	*Street* Millstone Lane	City Lincoln	*State* MA	*Zip* 01773	2000
			No 394	*Street* East Riding Dr	*City* Carlisle	*State* MA	*Zip* 01741	1993
			PO 65		*City* Sonyea	*State* NY	*Zip* 14556	1961
3	*Title* Lady	*Last* Gaga	*City* Hollywood			*State* CA	*Zip* 90027	2008
4	*Last* Madonna		*New York*		*Los Angeles*		*London*	2000

Data Model: Document

- Key/Value store where the value is a document with structure.
- Document store understands the structure of documents:
  - JSON, XML, BSON, PDF, Doc
- Sometimes documents can have sub-documents.
- Key attribute of a document store is that it lets you search within documents as well as searching for documents.

NoSQL Consistency

- The CAP theorem (Eric Brewer): it is impossible for a distributed computer system to simultaneously provide:
  - Consistency: All nodes see the same data at the same time
  - Availability: Every request receives a response about whether it succeeded
  - Partition tolerance: The system continues to operate despite arbitrary message loss or failure of parts of the system
- Originally posed as a conjecture by Brewer, later precisely proven by Gilbert and Lynch.
- It's an interesting history to read!
- Implications for NoSQL: Pick 2
  - Some systems pick CP
  - Other systems pick AP
  - Few systems pick CA
- Why?

Forms of Consistency

- Consistent/Partitionable Systems
  – BigTable, Hbase, HypterTable,MongoDB, Redis, MemcacheDB
- Available/Partitionable Systems
  – Cassandra, SimpleDB, CouchDB, Riak, TokyoCabinet, Dynamo
- Eventual Consistency
  – There exist multiple definitions. The most popular one is due to Vogels: The storage system guarantees that if no new updates are made to the object eventually (after the inconsistency window closes) all accesses will return the last updated value.
  – You can be AP and eventually consistent.
- Variable: through careful configuration choose how consistent you want to be:
  – Riak: set "R" and "W" to determine consistency levels
  – Cassandra: read/write to one, quorum, all
  – Oracle NoSQL DB: Can be CP when setting ack policy for simple majority, else is AP.
  – Amazon Dynamo: Picking read/write quorums trades off performance and degree of inconsistency

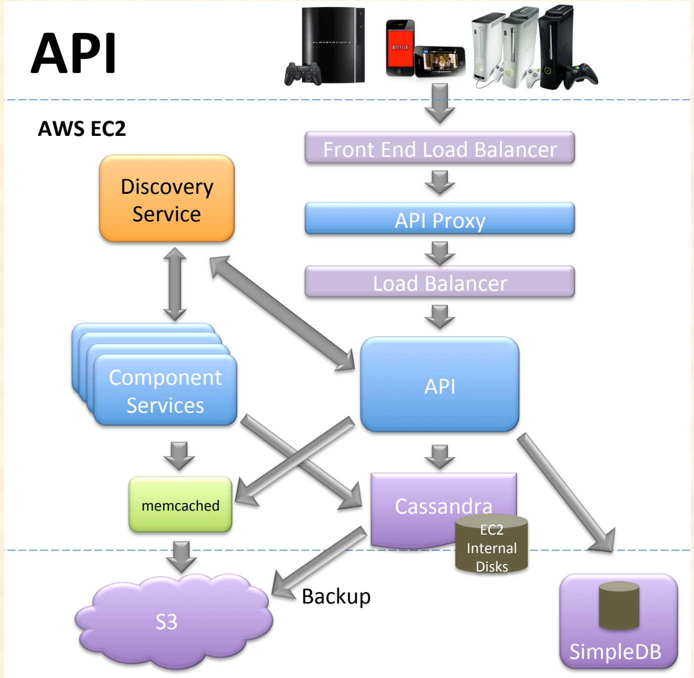
Outline

- In the beginning …
- The heyday of RDBMS
- The rebirth of key/value stores
- Key/Value stores today: NoSQL
- NoSQL & key/value use cases

Cassandra Use Case: Netflix

Adrian Cockcroft http://www.hpts.ws/sessions/GlobalNetflixHPTS.pdf

- In 2011 Netflix moved from centralized data centers to an out-sourced, distributed cloud-based system.
- Replaced centralized database with Cassandra for:
  - Customers
  - Movies
  - History
  - Configuration
- Why?
  - No need for schema changes
  - High performance
  - Scalability



HBase Use Case: Facebook Messages

Kannan Muthukkaruppan:
http://www.hpts.ws/sessions/StorageInfraBehindMessages.pdf

- Facebook integrated messaging, chat, email and SMS under a single new message framework.
- Why?
  - High write throughput
  - Good read performance
  - Horizontal scalability
  - Strong consistency
- What is in Hbase?
  - Small messages
  - Message metadata
  - Search index

MongoDB Use Case: Viper Media

http://nosql.mypopescu.com/post/16058009985/mongodb-at-viber-media-the-platform-enabling-free

- Viper uses MongoDB in a central service to which mobile clients connect to route messages to other clients.
- Why?
  - Scalability
  - Redundancy
- What is in MongoDB?
  - Variable length documents
  - Dictionary indexes

Beyond NoSQL

- Google's Spanner: a globally distributed SQL database with atomic transactions, synchronous replication, and consistency!
- Data is sharded
  - Replicated with Paxos state machines.
  - Paxos group leaders use two phase commit to enforce atomicity.
- Multiple consistency models
  - Snapshot reads
  - Read-only transactions
  - ACID transactions
- Enabling technology: TrueTime

Wrapping Up

- When should you be considering NoSQL?
  - Scalability
  - Low latency
  - Redundancy
  - No adhoc queries
  - Joins easily implementable in the application
  - Straight forward key structure
- A couple of sites I found pretty interesting:
  - List of NoSQL Databases: http://nosql-database.org
  - NoSQL Data Modeling: http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/
  - Recent Performance Comparison: http://www.networkworld.com/cgi-bin/mailto/x.cgi?pagetosend=/news/tech/2012/102212-nosql-263595.html