35+ Use Cases for Choosing Your Next NoSQL Database

Monday, June 20, 2011 at 8:50AM

Todd Hoff in nosql

We've asked What The Heck Are You Actually Using NoSQL For?. We've asked 101 Questions To Ask When Considering A NoSQL Database. We've even had a webinar What Should I Do? Choosing SQL, NoSQL or Both for Scalable Web Applications.

Now we get to the point of considering use cases and which systems might be appropriate for those use cases.

What are your options?

First, let's cover what are the various data models. These have been adapted from Emil Eifrem and NoSQL databases.

Document Databases

- Lineage: Inspired by Lotus Notes.
- Data model: Collections of documents, which contain key-value collections.
- Example: CouchDB, MongoDB
- Good at: Natural data modeling. Programmer friendly. Rapid development. Web friendly, CRUD.

Graph Databases

- Lineage: Euler and graph theory.
- Data model: Nodes & relationships, both which can hold key-value pairs
- Example: AllegroGraph, InfoGrid, Neo4j
- Good at: Rock complicated graph problems. Fast.

Relational Databases

- Lineage: E. F. Codd in A Relational Model of Data for Large Shared Data Banks
- Data Model: a set of relations
- Example: VoltDB, Clustrix, MySQL
- Good at: High performing, scalable OLTP. SQL access. Materialized views. Transactions matter. Programmer friendly transactions.

Object Oriented Databases

- Lineage: Graph Database Research
- Data Model: Objects
- Example: Objectivity, Gemstone
- Good at: complex object models, fast key-value access, key-function access, and graph database functionality.

Key-Value Stores

- Lineage: Amazon's Dynamo paper and Distributed HashTables.
- Data model: A global collection of KV pairs.
- Example: Membase, Riak
- Good at: Handles size well. Processing a constant stream of small reads and writes. Fast. Programmer friendly.

BigTable Clones

- Lineage: Google's BigTable paper.
- Data model: Column family, i.e. a tabular model where each row at least in theory can have an individual configuration of columns.
- Example: HBase, Hypertable, Cassandra
- Good at: Handles size well. Stream massive write loads. High availability. Multiple-data centers. MapReduce.

Data Structure Servers

- Lineage: ?
- Example: Redis
- Data model: Operations over dictionaries, lists, sets and string values.
- Good at: Quirky stuff you never thought of using a database for before.

Grid Databases

- Lineage: Data Grid and Tuple Space research.
- Data Model: Space Based Architecture
- Example: GigaSpaces, Coherence
- Good at: High performance and scalable transaction processing.

What should your application use?

- Key point is to rethink how your application could work differently in terms of the different data models and the different products. Right data model for the right problem. Right product for the right problem.
- To see what models might help your application take a look at What The Heck Are You Actually Using NoSQL For? In this

article I tried to pull together a lot of unconventional use cases of the different qualities and features developers have used in building systems.
- Match what you need to do with these use cases. From there you can backtrack to the products you may want to include in your architecture. NoSQL, SQL, it doesn't matter.
- Look at Data Model + Product Features + Your Situation. Products have such different feature sets it's almost impossible to recommend by pure data model alone.
- Which option is best is determined by your priorities.

If your application needs...

- **complex transactions** because you can't afford to lose data or if you would like a simple transaction programming model then look at a Relational or Grid database.
    - Example: an inventory system that might want full ACID. I was very unhappy when I bought a product and they said later they were out of stock. I did not want a compensated transaction. I wanted my item!
- **to scale** then NoSQL or SQL can work. Look for systems that support scale-out, partitioning, live addition and removal of machines, load balancing, automatic sharding and rebalancing, and fault tolerance.
- to **always** be able to **write** to a database because you need high availability then look at Bigtable Clones which feature eventual consistency.
- to handle lots of **small continuous reads and writes**, that may be volatile, then look at Document or Key-value or databases offering fast in-memory access. Also consider SSD.
- to implement social network operations then you first may want a Graph database or second, a database like Riak that supports

relationships. An in- memory relational database with simple SQL joins might suffice for small data sets. Redis' set and list operations could work too.

If your application needs...

- to operate over a **wide variety** of **access patterns** and **data types** then look at a Document database, they generally are flexible and perform well.
- powerful **offline reporting** with **large datasets** then look at Hadoop first and second, products that support MapReduce. Supporting MapReduce isn't the same as being good at it.
- to **span multiple data-centers** then look at Bigtable Clones and other products that offer a distributed option that can handle the long latencies and are partition tolerant.
- to build **CRUD** apps then look at a Document database, they make it easy to access complex data without joins.
- built-in **search** then look at Riak.
- to operate on **data structures** like lists, sets, queues, publish-subscribe then look at Redis. Useful for distributed locking, capped logs, and a lot more.
- **programmer friendliness** in the form of programmer friendly data types like JSON, HTTP, REST, Javascript then first look at Document databases and then Key-value Databases.

If your application needs...

- **transactions** combined with **materialized views** for **real-time** data feeds then look at VoltDB. Great for data-rollups and time windowing.
- **enterprise level support** and **SLAs** then look for a product that makes a point of catering to that market. Membase is an example.

- to log **continuous streams** of data that may have no consistency guarantees necessary at all then look at Bigtable Clones because they generally work on distributed file systems that can handle a lot of writes.
- to be as **simple as possible** to operate then look for a hosted or PaaS solution because they will do all the work for you.
- to be sold to **enterprise customers** then consider a Relational Database because they are used to relational technology.
- to **dynamically** build **relationships** between objects that have **dynamic properties** then consider a Graph Database because often they will not require a schema and models can be built incrementally through programming.
- to support **large media** then look storage services like S3. NoSQL systems tend not to handle large BLOBS, though MongoDB has a file service.

If your application needs...

- to **bulk upload** lots of data quickly and efficiently then look for a product supports that scenario. Most will not because they don't support bulk operations.
- an **easier upgrade path** then use a fluid schema system like a Document Database or a Key-value Database because it supports optional fields, adding fields, and field deletions without the need to build an entire schema migration framework.
- to implement **integrity constraints** then pick a database that support SQL DDL, implement them in stored procedures, or implement them in application code.
- a very deep join depth the use a Graph Database because they support blisteringly fast navigation between entities.
- to move behavior close to the data so the data doesn't have to be moved over the network then look at stored procedures of one kind or

another. These can be found in Relational, Grid, Document, and even Key-value databases.

If your application needs...

- to **cache or store BLOB data** then look at a Key-value store. Caching can for bits of web pages, or to save complex objects that were expensive to join in a relational database, to reduce latency, and so on.
- a **proven track record** like not corrupting data and just generally working then pick an established product and when you hit scaling (or other issues) use on of the common workarounds (scale-up, tuning, memcached, sharding, denormalization, etc).
- **fluid data types** because your data isn't tabular in nature, or requires a flexible number of columns, or has a complex structure, or varies by user (or whatever), then look at Document, Key-value, and Bigtable Clone databases. Each has a lot of flexibility in their data types.
- other business units to **run quick relational queries** so you don't have to reimplement everything then use a database that supports SQL.
- to operate in the cloud and automatically take full advantage of cloud features then we may not be there yet.

If your application needs...

- support for **secondary indexes** so you can look up data by different keys then look at relational databases and Cassandra's new secondary index support.
- creates an **ever-growing set of data** (really BigData) that rarely gets accessed then look at Bigtable Clone which will spread the data over a distributed file system.

- to **integrate with other services** then check if the database provides some sort of write-behind syncing feature so you can capture database changes and feed them into other systems to ensure consistency.
- **fault tolerance** check how durable writes are in the face power failures, partitions, and other failure scenarios.
- to **push** the **technological** envelope in a direction nobody seems to be going then build it yourself because that's what it takes to be great sometimes.
- to work on a **mobile platform** then look at CouchDB/Mobile couchbase.

Which is Better?

- Moving for a 25% improvement is probably not a reason to go NoSQL.
- Benchmark relevancy depends on the use case. Does it match your situation(s)?
- Are you a startup that needs to release a product as soon as possible and you are playing around with ideas? Both SQL and NoSQL can make an argument.
- Performance may be equal on one box, but what happens when you need N?
- Everything has problems, if you look at Amazon forums it's EBS is slow, or my instances won't reply, etc. For GAE it's the datastore is slow or X. Every product which people are using will have problems. Are you OK with the problems of the system you've selected?

Article originally appeared on High Scalability (http://highscalability.com/).