NoSQL

By Perry Hoekstra
Technical Consultant
Perficient, Inc.

perry.hoekstra@perficient.com

Why this topic?

- Client's Application Roadmap
  - "Reduction of cycle time for the document intake process. Currently, it can take anywhere from a few days to a few weeks from the time the documents are received to when they are available to the client."
- New York Times used Hadoop/MapReduce to convert pre-1980 articles that were TIFF images to PDF.

Agenda

- Some history
- What is NoSQL
- CAP Theorem
- What is lost
- Types of NoSQL
- Data Model
- Frameworks
- Demo
- Wrapup

History of the World, Part 1

Perficient

- Relational Databases – mainstay of business
- Web-based applications caused spikes
  - Especially true for public-facing e-Commerce sites
- Developers begin to front RDBMS with memcache or integrate other caching mechanisms within the application (ie. Ehcache)

Scaling Up

- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding

Scaling RDBMS – Master/Slave

Perficient

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves

Scaling RDBMS - Sharding

Perficient

- Partition or sharding
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Other ways to scale RDBMS

Perficient

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

What is NoSQL?

Perficient

- Stands for **N**ot **O**nly **SQL**
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins
- All NoSQL offerings relax one or more of the ACID properties (will talk about the CAP theorem)

Why NoSQL?

Perficient

- For data storage, an RDBMS cannot be the be-all/end-all
- Just as there are different programming languages, need to have other data storage tools in the toolbox
- A NoSQL solution is more acceptable to a client now than even a year ago
  - Think about proposing a Ruby/Rails or Groovy/Grails solution now versus a couple of years ago

How did we get here?

Perficient

- Explosion of social media sites (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Just as moving to dynamically-typed languages (Ruby/Groovy), a shift to dynamically-typed data with frequent schema changes
- Open-source community

Dynamo and BigTable

Perficient

- Three major papers were the seeds of the NoSQL movement
  - BigTable (Google)
  - Dynamo (Amazon)
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem (discuss in a sec ..)

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm
- Not a backlash/rebellion against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings

CAP Theorem

- Three properties of a system: consistency, availability and partitions
- You can have at most two of these three properties for any shared-data system
- To scale out, you have to partition. That leaves either consistency or availability to choose from
  - In almost all cases, you would choose availability over consistency

Availability

- Traditionally, thought of as the server/process available five 9's (99.999 %).
- However, for large node system, at almost any point in time there's a good chance that a node is either down or there is a network disruption among the nodes.
  - Want a system that is resilient in the face of network disruption

Consistency Model

Perficient

- A consistency model determines rules for visibility and apparent order of updates.
- For example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses.
  - Client B reads row X from node M
  - Does client B see the write from client A?
  - Consistency is a continuum with tradeoffs
  - For NoSQL, the answer would be: maybe
  - CAP Theorem states: Strict Consistency can't be achieved at the same time as availability and partition-tolerance.

Eventual Consistency

Perficient

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- Known as BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency), as opposed to ACID

What kinds of NoSQL

- NoSQL solutions fall into two major areas:
  - Key/Value or 'the big hash table'.
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
  - Schema-less which comes in multiple flavors, column-based, document-based or graph-based.
    - Cassandra (column-based)
    - CouchDB (document-based)
    - Neo4J (graph-based)
    - HBase (column-based)

Key/Value

Pros:

- very fast
- very scalable
- simple model
- able to distribute horizontally

Cons:

- many data structures (objects) can't be easily modeled as key value pairs

Schema-Less

Pros:

- Schema-less data model is richer than key/value pairs
- eventual consistency
- many are distributed
- still provide excellent performance and scalability

Cons:

- typically no ACID transactions or joins

Common Advantages

Perficient

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  - Down nodes easily replaced
  - No single point of failure
- Easy to distribute
- Don't require a schema
- Can scale up and down
- Relax the data consistency requirement (CAP)

What am I giving up?

Perficient

- joins
- group by
- order by
- ACID transactions
- SQL as a sometimes frustrating but still powerful query language
- easy integration with other applications that support SQL

Cassandra

Perficient

- Originally developed at Facebook
- Follows the BigTable data model: column-oriented
- Uses the Dynamo Eventual Consistency model
- Written in Java
- Open-sourced and exists within the Apache family
- Uses Apache Thrift as it’s API

Typical NoSQL API

Perficient

- Basic API access:
  - get(key) -- Extract the value given a key
  - put(key, value) -- Create or update the value given its key
  - delete(key) -- Remove the key and its associated value
  - execute(key, operation, parameters) -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc).

Cassandra and Consistency

Perficient

- Talked previous about eventual consistency
- Cassandra has programmable read/writable consistency
  - One: Return from the first node that responds
  - Quorom: Query from all nodes and respond with the one that has latest timestamp once a majority of nodes responded
  - All: Query from all nodes and respond with the one that has latest timestamp once all nodes responded. An unresponsive node will fail the node

Cassandra and Consistency

Perficient

- Zero: Ensure nothing. Asynchronous write done in background
- Any: Ensure that the write is written to at least 1 node
- One: Ensure that the write is written to at least 1 node's commit log and memory table before receipt to client
- Quorom: Ensure that the write goes to node/2 + 1
- All: Ensure that writes go to all nodes. An unresponsive node would fail the write

Some Statistics

Perficient

- Facebook Search
- MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms
- Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms

Don’t forget about the DBA

Perficient

- It does not matter if the data is deployed on a NoSQL platform instead of an RDBMS.
- Still need to address:
  - Backups & recovery
  - Capacity planning
  - Performance monitoring
  - Data integration
  - Tuning & optimization
- What happens when things don’t work as expected and nodes are out of sync or you have a data corruption occurring at 2am?
- Who you gonna call?
  - DBA and SysAdmin need to be on board