Busy Developer's Guide to NoSQL

Ted Neward

Neward & Associates

http://www.tedneward.com | ted@tedneward.com

Credentials

Who is this guy?

- CTO, iTrellis (http://www.itrellis.com)

  ask me how we can help your project, your team or your firm
- Microsoft MVP (F#, C#, Architect); JSR 175, 277 EG
- Author

  Professional F# 2.0 (w/Erickson, et al; Wrox, 2010)
  Effective Enterprise Java (Addison-Wesley, 2004)
  SSCLI Essentials (w/Stutz, et al; OReilly, 2003)
  Server-Based Java Programming (Manning, 2000)
- Blog: http://blogs.tedneward.com
- Writing: http://www.newardassociates.com/writing.html
- Twitter: @tedneward
- For more, see http://www.newardassociates.com/about.html

Objectives

- What is NoSQL?
- Why and when should you consider using it/them?
- How do you use it/them?
- Take a look at (a few, of many possible) examples

'NoSQL'

What the heck is a 'NoSQL', anyway?

"NoSQL"

"NoSQL" isn't really a great categorization

- Very rarely is "not X" a useful descriptor
- A better categorization/taxonomy is required
- Thus, I choose to eschew "NoSQL" as a useful term

Conceptuals

"Each node in a system should be able to make decisions purely based on local state. If you need to do something under high load with failures occurring and you need to reach agreement, you've lost."

--Werner Vogels, CTO, Amazon

Conceptuals

The problem is one of load/scale and contention

- Repeat after me: Contention is the enemy of scalability
- And scale is the issue of the Internet

Conceptuals

With the advent of the Web, we changed our enterprise apps

- Before, enterprise apps were internal
  - **Known user base, known loads, known scale**
  - **This user base was not likely to change without huge warning**
- With the Web app, we began to project our enterprise apps out into the Internet
  - **This meant an unknown and unpredictable user base**
  - **With that came an unknown and unpredictable load and scale**

Conceptuals

Scale and load began to take down the existing infrastructure

- Traditionally-managed RDBMS'es simply couldn't keep up
- ACID has its uses... but we found the edge really quickly
- Distributed Two-Phase Commit Transactions

Theory

"In theory, there's no difference
between theory and practice
In practice, however...."

--(various)

Conceptuals

CAP Theorem

- Consistency: all database clients see the same data, even with concurrent updates
- Availability: all database clients are able to access some version of the data
- Partition Tolerance: the database can be split over multiple servers
- "Pick two"

RDBMS goes for C + A
Most "NoSQL" databases go for A + P

Conceptuals

Data "shapes"

- Groupings of relations, tuples, and relvars
  - **If you don't know what these are, you don't know your relational theory**
  - **strongly-typed, enforced by the database**
- Object databases
  - **capturing the object graphs that appear in O-O systems**
  - **strongly-typed, defined by an O-O language (not external schema)**
- Key-value systems
  - **CRUD based solely on the primary key; no joins**
  - **weakly- or untyped**
- Document-oriented
  - **collections of named fields holding data (or more collections)**
  - **weakly- or untyped**

Conceptuals

Data "shapes" (continued)

- Graph databases
  - **capturing not just graph structures, but the "arcs" between nodes**
  - **graph-based query API/language**
- Column-oriented/tabular
  - **columns-and-tables, but no relations/relvars**
- Hierarchical databases
  - **generally, these are XML stores**
- Distributed peer stores
  - **low-level storage API**

Practice

"In theory, there's no difference
between theory and practice
In practice, however...."

--(various)

Examinations

The "NoSQL"s we'll be looking at:

- db4o
- CouchDB
- MongoDB
- Cassandra
- Neo4J
- All of these are open source and generally business-friendly

db4o

Object-oriented

db4o

db4o: object-oriented

- Java/CLR objects
- "native queries" (using Java/CLR language)
- strongly-typed to the object types
- much(!) easier refactoring for code evolution
- http://www.db4o.com
- Upshot: easy-to-use storage for non-integration databases

db4o

Opening the database

```
ObjectContainer db = null;
try {
  db = Db4o.openFile("mydata.db4o");

  // . . .
}
catch (Exception ex) {
  // Do more than just log the exception, please
}
finally {
  if (db != null)
    db.close();
}
```

db4o

Storing an object

```
// Create the new Person
Person p = new Person("Matthew", "Neward", 13, new Address(...));

// Store
db.set(p);
db.commit();
```

db4o

Fetching an object (prototype-based)

```
// Fetch all the Persons
ObjectSet persons =
  // use empty/default object (all fields hold null/0 values)
  db.get(new Person());
  // or could use type parameter:
  // db.get(Person.class);
while (persons.hasNext())
  System.out.println(persons.next());

// Fetch all the Newards
ObjectSet newards = db.get(new Person(null, "Neward", 0, null);
while (newards.hasNext())
  System.out.println(newards.next());
```

db4o

Updating an object

```
// Fetch Matthew Neward (note the absence of error-checking)
Person matthew = (Person)
  db.get(new Person("Matthew", "Neward", 0, null).next();

// Happy Birthday!
matthew.setAge(matthew.getAge() + 1);

// Store
db.set(matthew);
db.commit();
```

MongoDB

Document-oriented

MongoDB

MongoDB: document-oriented

- BSON documents
- JavaScript query language
- map/reduce, server-side execution
- master/slave replication; sharding
- drivers for most languages
- C++ implementation
- http://www.mongodb.org
- Upshot: useful for most things a small "dynamically-typed" RDBMS would be useful for

Basics

MongoDB server holds 0..n databases

- "database" = named instance of collections
- "collection" = named instance of documents
- "document" = collection of name-value pairs
- documents generally take the form of JSON-like data
- think "Map<name, value>" where values can be one of:
  - **integer, double**
  - **boolean**
  - **strings, dates, regular expressions**
  - **array**
  - **object ID's (OIDs)**
  - **binary data**
  - **code (!)**

Basics

MongoDB development generally consists of a few basic things:

- Connect to a local server, or to a server running on a different machine
- Authenticate to a Mongo database (username/password)
- Examine the root of a Mongo database as a collection (of collections)
- Get (or create) a collection
- Return all elements in a collection
- Inserting a document into a collection

  Uniqueness is determined by silently-added "_id" field to the document

Basics

Of course, MongoDB also supports querying for documents

- Find a single document matching specific criteria
  - **document passed as query**
  - **"_id" is perfectly acceptable as criteria**
  - **returns null on failure**
- Find a set of documents, either by query/predicate or complex query

  first is a "query by example" query; any objects which match all of the criteria in the prototype will be returned
  second is a "query by document"; pass in a document that contains some simple expressions for comparison

Basics

More querying options:

- Find documents via predicate clause

  can also use "dot notation" to query elements contained from toplevel document into subobjects

- Find documents via code block

  takes a Java/ECMAScript code block, executes it on the server, and returns the list of objects that are yielded from the code block's execution

Basics

- Operators

  $lt, $gt, $lte, $gte $ne: less-than, greater-than, less-than-or-equal, greater-than-or-equal, not-equals

  $in, $nin: specify array of possible matches ("IN"); "!IN"

  $all: like $in, but must match all values

  $size: specify size of an array

  $exists: existence (or lack) of a field

  $where: specify a "WHERE"-style clause

  { $where : "this.field > 3" }

- Combine operators to specify range

  { "field" : { $gt : 1, $lt : 5 } } == 1 < field < 5

CouchDB

Document-oriented REST

CouchDB

CouchDB: document-oriented

- JSON documents
- JavaScript map/reduce
- replication in multiple forms
- MVCC (no-lock) writes
- no drivers; everything via REST protocol
- "Couch apps": HTML+JavaScript+Couch
- Erlang implementation
- http://couchdb.apache.org
- Upshot: best with predefined queries, accumulating data over time
- Upshot: Couch apps offer an appserver-less way to build systems

Conceptuals

No locks; MVCC instead

- Multi-Version Concurrency Control
- documents are never locked; instead, changes are written as new versions of a document on top of old document

  each document has a revision number+content-hash
  advantage: read requests already in place can finish even in the face of a concurrent write request; new read requests return the written version
  result: high parallelization utility
- conflicts mean both versions are preserved

  leave it to the humans to figure out the rest!

API

CouchDB's principal I/O is HTTP/REST

- GET requests "retrieve"
- PUT requests "store" or "create"
- POST requests "modify"
- DELETE requests "remove"
- this isn't ALWAYS true

  for the most part, though, it holds
- the CouchDB programmer's best friend: curl

  or any other command-line HTTP client
  good exercise: write your own!

API

- Pre-requisite: HTTP/1.1 RFC

  curl is a no-abstraction HTTP client

- curl primer:
  - -X: HTTP verb to use
  - -v: Verbose output
  - -H: header (in "Header: value" format)
  - -d {data}: send {data} as part of request
  - --data-binary {data}
  - @{file} means use file contents as data

API

CouchDB's principal data is the document

- schemaless key/value store

  (essentially a JSON object)

- certain keys (_rev, _id) reserved for CouchDB's use

  _id is most often a UUID/GUID
  fetch one (if you like) from http://localhost:5984/_uuids
  database will attach one on new documents

- _rev is this document's revision number

  formatted as "N-{md5-hash}"
  OK to leave blank for new document...
  ... but must be sent back as part of the stored document
  assuming the _rev's match, all is good
  if not, "whoever saves a changed document first, wins"
  (essentially an optimistic locking scheme)

API

```
{
   "_id": "286ccf0edf77bfb6e780be88ae000d0b",
   "firstname": "Ted",
   "lastname": "Neward",
   "age": 40
}
```

assuming the above is stored in person.json, then...

```
curl -X PUT http://localhost:5984/%1 -d @%2 -H "Content-Type:
application/json"
```

API

Documents can have attachments

- attachments are binary files associated with the doc

  essentially URLs within the document
- simply PUT the document to the desired {_id}/{name}
- provide the document's {_rev} as a query param

  after all, you are modifying the document
- add ?attachments=true to fetch the attachments as binary (Base64) data when fetching the document

API

Documents can be inserted in bulk

- send a POST to http://localhost:5984/{dbname}/_bulk_docs
- include array of docs in body of POST

  if updating, make sure to include _rev for each doc
- "non-atomic" mode: response indicates which documents were saved and which weren't

  default mode
- "all-or-nothing" mode: all documents will be saved, but conflicts may exist

  pass "all_or_nothing : true" in the request

API

Retrieving an individual document is just a GET

GET http://localhost:5984/{database}/{_id}

API

Retrieving multiple documents uses Views

- views are essentially predefined MapReduce pairs defined within the database
- create a "map" function to select the data from a given document
- create a "reduce" function to collect the "map"ped data into a single result (optional)
- views are "compiled" on first use

API

Views can be constrained by query params

- ?key=... : return exact row
- ?startkey=...&endkey=... : return range of rows

Cassandra

Columnar-oriented clusters

Cassandra

Cassandra: column-oriented

- writes are faster than reads!
- "strange" data model
  - **Cluster: the nodes in a logical Cassandra instance; can contain multiple Keyspaces**
  - **Keyspace: namespace for ColumnFamilies (often one per app)**
  - **Column: triplet of name, value and timestamp**
  - **ColumnFamilies: contain multiple Columns, referenced by row keys**
  - **SuperColumns: Columns that themselves have subcolumns**
- binary protocol (Thrift), language drivers
- Java implementation
- http://cassandra.apache.org
- Upshot: best for write-mostly systems

Cassandra

A poorly-chosen name, perhaps...

Overview

What is Cassandra?

"Apache Cassandra is an open-source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, column-oriented database that bases its distribution design on Amazon's Dynamo and its data model on Google's Bigtable. Created at Facebook, it is now used at some of the most popular sites on the web."

Overview

Cassandra's Buzzword Bingo

- Open-source
- Distributed: runs on multiple machines, appearing as one
- Decentralized: no one node is a "master" node
- Elastically scalable: seamless scale up
- High-availability: always accessible, never "out"
- Tunably consistent: user-selectable consistency levels
- Column-oriented: shape of the data

Installation

Getting it up and running

Installation

Installing Cassandra: Several paths

- Install binaries from DataStax
  - **http://www.datastax.com**
  - **Community edition (free)**
  - **Enterprise edition (commercial)**
  - **Easy Windows install; others, not a big deal**
- Install binaries from Apache

  http://cassandra.apache.org
- Build from source
  - **REAL programmers do this**
  - **REAL programmers also know how to program in Java ;-)**
  - **... because the source is in Java**

Installation

DataStax is a commercial supporter of Cassandra

- Lots of the docs and support comes from them
- They distribute a "Community Edition" (free)
  - **Includes their Cassandra ring-monitoring tool**
  - **Installs Cassandra as a (Windows) service**
- Contact them for more info

I don't work for them or endorse them

Installation

Apache distribution is binary tarballs

- ... meaning, if you don't know what a tarball is, don't go this route
- built straight from source repo

no support tools

- requires all dependencies to be installed manually
- (basically, one step from building from source)

Installation

Hello, Java!

- Cassandra is written in Java
  - **this means a JDK will need to be on your system**
  - **DataStax will put one on for you**
  - **Apache will not--you will need to**
- Cassandra wants Oracle JDK, not OpenJDK

  don't ask why; they just do

Installation

Once installed, verify

- "nodetool": command-line tool to examine nodes
- "cqlsh": Python shell for CQL (later)
- "cassandra-cli": Java legacy CLI tool

Installation

Once installed, tune and inspect

- cassandra.yaml is the centerpiece
- logs and configrs are in /var or C:\var
- remember to tune Cassandra and the JVM

Installation

Clusters

- Cassandra's natural state is in a cluster
- This is non-trivial on a single machine
  - **Solution: collection of laptops**
  - **Solution: collection of VMs**

Architecture

Cassandra is not your normal database

Architecture

Cassandra is a "ring" storage

- No master node

  no single point of failure

- Even distribution of data
- Nodes can be multiple-oer-machine

Cassandra's Data Model

'I don't think we're in RDBMS-sas anymore, Toto...'

Data Model

Columns, Column Families, Super Columns, oh my!

- Cassandra is a "column-oriented" data model
  - **CQL (later) will make it look more SQL-ish**
- This is not a relational data model
  - **despite the similar shape and terminology**
  - **despite the "Query Language"**
  - **despite how badly you want it to be**

Data Model

Intrinsically schemaless

- meaning, the only structure is what Cassandra knows
- rows in a "table" need not be similarly-structured
  - **this permits easy extension and enhancement**
  - **this also permits more bug potential**
- (cue "static- vs dynamic-typing debate" here)

Data Model

Technically, there are 2 data models

- The original (Thrift-based) data model
  - **Columns, ColumnFamilies, SuperColumns, SuperColumnFamilies**
  - **Very non-relational**
  - **Very confusing to some**
- The newer, Cassandra Query Language (CQL) model
  - **Abstracts on top of the Thrift model above**
  - **Provides relational-ish kinds of structure**
  - **Still schemaless**
  - **Provides "extensions" to the basic model**

Data Model

The original Cassandra data model

- Column: 3-part tuple, the atom of storage
- SuperColumn: collection of columns in a row
- Rows: of columns or SCs identified by a RowKey
- Column Family: container for rows of columns
- SuperColumnFamily: container for rows of SCs

Data Model

Column

- three-part tuple of name, value, timestamp
  - **name: name by which to grab the value**
  - **value: data storage (hex/binary storage)**
  - **timestamp: date/time of write**

    often ignored/unused

Data Model

Super Column

- a "super column" is a grouping of columns
  - **this is not a row!**
  - **this is a collection of columns within a row**
  - **imagine a column with "subcolumns"**
- basis for other CQL types (later)

Data Model

Rows

- row is uniquely identified by a RowKey
  - **unique across the keyspace (database)**
  - **RowKeys can be composite keys**
  - **(for almost all data types)**

Data Model

Column Family is sort of a table

- a collection of rows
- essentially, a column-of-subcolumns

Data Model

Super Column Family

- a collection of Super Columns
- sort of a "map of maps"
- again, not a row, but a column

Data Model

Collections of those Table things: Keyspace

- Keyspace is roughly a database instance
- Some advocate one keyspace per application
- No real need to do this (except for lexical scope)

Data Model

Old-school keyspaces defined in several ways

- Your code
- Via cassandra-cli (script or by hand)
- Configuration (cassandra.yaml)

Data Model

Cassandra Query Language (CQL)

- abstraction on top of the earlier model
- designed to look/feel more SQL-ish
- CQL 3 == Cassandra 1.2.x+
- CQL 2 == Cassandra 1.0.x+
- there are some differences between 2 and 3

Data Model

CQL 3 DDL syntax

- tables, columns (with subcolumns), etc
- columns have CQL data types

Data Model

CQL Data Types

- **ascii (strings): US-ASCII character string**
- **bigint (integers): 64-bit signed long**
- **blob (blobs): Arbitrary bytes (no validation), expressed as hexadecimal**
- **boolean (true/false)**
- **counter (integer): Distributed counter value (64-bit long)**
- **decimal (integers, floats): Variable-precision decimal**
- **double (integers): 64-bit IEEE-754 floating point**
- **float (integers, floats): 32-bit IEEE-754 floating point**
- **inet (strings): IP address string in IPv4 or IPv6 format**

Data Model

CQL Data Types

- **int (integers): 32-bit signed integer**
- **text (strings): UTF-8 encoded string**
- **timestamp (integers, strings): Date plus time, encoded as 8 bytes since epoch**
- **uuid (uuids): Type 1 or type 4 UUID in standard UUID format**
- **timeuuid (uuids): Type 1 UUID only (CQL 3)**
- **varchar (strings): UTF-8 encoded string**
- **varint (integers): Arbitrary-precision integer**

Data Model

CREATE KEYSPACE

- CREATE KEYSPACE keyspace_name WITH properties

Data Model

USE

- USE keyspace_name

Data Model

CREATE TABLE

- CREATE TABLE keyspace_name.table_name (column_definition, column_definition, ...) WITH property AND property ...

column_definition = column_name cql_type
| column_name cql_type PRIMARY KEY
| PRIMARY KEY (partition_key)
| column_name collection_type

Data Model

CREATE INDEX

- CREATE INDEX identifier? ON table_name '(' identifier ')'

Data Model

INSERT

- INSERT INTO table (identifer, identifer, ...) VALUES (value, value, ...) USING TIMESTAMP integer AND TTL integer

Data Model

UPDATE

- UPDATE table USING TIMESTAMP integer AND TTL integer SET identifier = value, identifier = value, ... WHERE predicate

Data Model

Collections

- Cassandra "values" can include "collection" types
- these are "columns" made up of multiples of data
- yes, just like a list or map or set collection in C#/Java/C++
- in fact...

Data Model

CQL Collection Types

- list: A collection of one or more ordered elements
- map: A collection of one or more timestamp, value pairs
- set: A collection of one or more elements

Data Model

SELECT

- SELECT identifier, identifier, ... FROM table WHERE predicate

  (also supports ORDER_BY and LIMIT)

Data Model

Other SQL-like CQL commands

- ALTER TABLE
- DROP TABLE, DROP KEYSPACE, DROP INDEX
- DELETE
- BATCH (for batch execution)

Data Model

Defining a keyspace using CQL is done in several ways

- Your code
- Through cqlsh (script or by hand)

Accessing

How do we use it?

Client Access

Cassandra has two basic APIs

- one is the Apache Thrift RPC-based one
  - **legacy**
  - **more widespread support**
- the other is native/CQL access
  - **future direction**
  - **definitely easier to grok for RDBMS users**
  - **Java and .NET drivers**

Client Access

Apache Thrift

- low-level RPC API
- similar to RMI or .NET Remoting
- cross-platform
- baseline level for access
- lots of OSS layers on top of this

Client Access

API is extremely low-level

- you need to know the Cassandra low-level data model
- you need to feel comfortable with Thrift (not hard)
- you need to be able to navigate with cursors and such
- better off with a "high-level client"

Client Access

"High-level" clients

- Libraries which encapsulate the Thrift API
- compared to CQL, these are more "mid-level" now
- they protect you against the Thrift/API details, but you're still working with the original (C/CF/SC/SCF) data model
- "high-level" clients for practically every platform

Client Access

CQL drivers

- some higher-level CQL-based drivers are available

  basically Java and .NET

- these permit CQL access and manipulation
- some have abstractions on top of that

  such as DataStax's .NET driver, which has Linq2CQL

Neo4J

Graph-oriented

Neo4J

Neo4J: graph-oriented

- data model is basically "just" nodes and arcs
- graph API (Visitor) for navigation/query
- Java implementation
- http://www.neo4j.org
- Upshot: perfect for graph analysis and storage

Wrapping up

What have you learned?

Summary

- This is obviously a high-level view
  - further investigation is necessary!
  - prototypes are necessary!
  - allowing yourself time to fail is necessary!

Questions