NoSQL

with HBase and Hadoop

BeJUG - 17/6/2010

http://www.flickr.com/photos/wolfgangstaudt/2215246206/

OUTER THOUGHT

IIC » TECHNOLOGIEPARK 3 » B-9052 ZWIJNAARDE (GENT) » www.outerthought.org

Who am I

- Steven Noels - stevenn@outerthought.org
- Outerthought : scalable content applications
- makers of Daisy, Lily and Kauri : open source internet/Java/REST/content apps

1. Intro
2. Theory
3. Technology
4. Experiences

An evolution driven by pain.

History

hierarchical databases
IMS
XMLDB
OODBMS
1. standardization
RDBMS
2. simplification
MICROSOFT
ADO.NET
STEP BY STEP
RAILS
HIBERNATE
django
pear

History

4. rethinking
the problem
RDBMS
NOSQL
caching
denormalisation
sharding
replication ...
3. pain

Four Trends

- Trend 1 : Data Size
- Trend 2 : Connectedness
- Trend 3 : Semi-structure
- Trend 4 : Architecture

Trend 1: Data size

Data source: IDC 2007

3

Trend 2: Connectedness

Trend 3: Semi-structure

- Individualization of content
  - In the salary lists of the 1970s, all elements had exactly one job
  - In the salary lists of the 2000s, we need 5 job columns! Or 8? Or 15?
- All encompassing "entire world views"
  - Store more data about each entity
- Trend accelerated by the decentralization of content generation that is the hallmark of the age of participation ("web 2.0")

Trend 4: Architecture

1980s: Mainframe applications

Trend 4: Architecture

1990s: Database as integration hub

Trend 4: Architecture

2000s: (moving towards) Decoupled services
with their own backend

http://bigdatamatters.com/bigdatamatters/2010/04/high-availability-with-oracle.html

Enter NoSQL

It's a Cambrian Explosion

Buzz-oriented development

Image credit: http://browsertoolkit.com/fault-tolerance.png

Common themes

- SCALE SCALE SCALE
- new datamodels
- devops
- N-O-SQL
- The Cloud :
  technology is of no interest anymore

New Data

- sparse structures
- weak schemas
- graphs
- semi-structured
- document-oriented

NoSQL

- Not a movement.
- Not ANSI NoSQL-2010.
- Not one-size-fits-all.
- Not (necessarily) anti-RDBMS.
- No silver bullet.

NoSQL = pro Choice

NoSQL = toolbox

NOSQL, if you need ...

- horizontal scaling (*out* rather than *up*)
- unusually common data (aka free-structured)
- speed (especially for writes)
- the bleeding edge

SQL/RDBMS, if you need ...

- SQL
- ACID
- normalisation
- a defined liability

Some RDBMS bashing

» sparse and dynamic tables

id	content	field1	field2	field3	field4	field5	field6
561	foo	val561					
565	bar		val565				
584	baz				val584		
589	boo			val589			
654	...						val654
678						val678	
698					val698		
710		...					
725				...			

Some RDBMS bashing

» solution

id	content	field1	field2	field3	field4	field5	field6	...
561	foo	val561						
565	bar		val565					
584	baz				val584			
589	boo			val589				
654	...						val654	
678						val678		
698					val698			
710		...						
725				...				
...								

```
mysql> desc thefields;
+---------------------+---------------+------+-----+---------+-------+
| Field               | Type          | Null | Key | Default | Extra |
+---------------------+---------------+------+-----+---------+-------+
| doc_id              | bigint(20)    | NO   | PRI | NULL    |       |
...
| fieldtype_id        | bigint(20)    | NO   | PRI | NULL    |       |
...
| stringvalue         | varchar(255)  | YES  | MUL | NULL    |       |
| datevalue           | datetime      | YES  | MUL | NULL    |       |
| datetimevalue       | datetime      | YES  | MUL | NULL    |       |
| integervalue        | bigint(20)    | YES  | MUL | NULL    |       |
| floatvalue          | double        | YES  | MUL | NULL    |       |
| decimalvalue        | decimal(10,5) | YES  | MUL | NULL    |       |
| booleanvalue        | char(1)       | YES  | MUL | NULL    |       |
...
+---------------------+---------------+------+-----+---------+-------+
25 rows in set (0.00 sec)
```

More RDBMS bashing

- replication and failure recovery
  - (when working on a budget)
- application-level partitioning logic

1 . Intro

2 . Theory

3 . Technology

4 . Experiences

Academic background

- Amazon Dynamo
- Google BigTable
- Eric Brewer CAP theorem

Shameless plug

a NOSQL summer

Papers

A seasonal, worldwide reading club for databases, distributed systems & NOSQL-related scientific papers.

Amsterdam
Berlin
Boston
Bucharest
Budapest
Cheltenham
Chicago
Denver
Ghent
Groningen
London
Los Angeles
Madrid
Malmö
Memphis

Currently featuring 417 participants in 27 cities on 3 continents, studying & discussing 29 papers!

A NOSQL Summer is a network of local reading groups, that will decipher & discuss NOSQL-related articles, from late June to early September 2010. Each group sets its own meeting pace (usually once a week or once every two weeks) and select which papers are up for discussion.

At every cycle, members read the selected paper at home and then meet up for an hour or so to discuss, debate and answer their own questions.

We then encourage you to produce an annotated version of the paper, or short summary that we can then publish here for the rest of world to peruse.

nosqlsummer.org

Shameless plug

Papers

A seasonal, worldwide reading club for databases, distributed systems & NOSQL-related scientific papers.

San Francisco (53)
Ghent (36)
Seattle (30)
Paris (27)
London (25)
Bucharest (23)
New York (22)
Vancouver (21)
Berlin (21)
Wellington (20)
Saint Louis (19)
Los Angeles (19)
Madrid (17)
Toronto (16)
Rome (16)
Stuttgart (15)
Malmö (15)
Budapest (13)
Boston (11)
Chicago (8)
Philadelphia (7)
Cheltenham (7)
Denver (5)
São Paulo (4)
Groningen (2)
Amsterdam (2)
Memphis (1)

Currently featuring 417 participants in 27 cities on 3 continents, studying & discussing 29 papers!

A NOSQL Summer is a network of local reading groups, that will decipher & discuss NOSQL-related articles, from late June to early September 2010. Each group sets its own meeting pace (usually once a week or once every two weeks) and select which papers are up for discussion.

At every cycle, members read the selected paper at home and then meet up for an hour or so to discuss, debate and answer their own questions.

We then encourage you to produce an annotated version of the paper, or short summary that we can then publish here for the rest of world to peruse.

Please note that, in most cities, you do not need to sign up to attend NOSQL Summer meetings. You just need to have read the paper planned for the week by your local chapter and show up at the designated meeting place!

Feel free to skip a meeting or jump in at any time. We're trying to make this low-maintenance and flexible, for everybody to get a chance to learn more about a fuzzy concept that's here to stay.

A few things you can do...

Propose a Paper
Propose a City
Follow us on Twitter
or Contact us

nosqlsummer.org

Amazon Dynamo

- coined the term ‘eventual consistency’
- consistent hashing

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.

Eventual Consistency Gone Wild

server 1
1.
2.
replica
server 2
1. update ACL: disallow mother from folder 'spring break'
2. upload spring break pictures
how is my boy doing on his spring break?

Consistent hashing

- a solution for naive mod *n* distributions
  - specifically in the case of adding or deleting nodes

Consistent hashing

Google BigTable

- multi-dimensional column-oriented database
- on top of GoogleFileSystem
- object versioning

CAP theorem

strong consistency
high availability
partition-tolerance

CAP

- **Strong Consistency:** all clients see the same view, even in the presence of updates
- **High Availability:** all clients can find some replica of the data, even in the presence of failures
- **Partition-tolerance:** the system properties hold even when the system is partitioned

Culture Clash

- ACID
  - highest priority: strong consistency for transactions
  - availability less important
  - pessimistic
  - rigorous analysis
  - complex mechanisms

- BASE
  - availability and scaling highest priorities
  - weak consistency
  - optimistic
  - best effort
  - simple and fast

spectrum

Availability ≠ total async !

The Enterprise Service Bus

bus =

congestion

Bus systems

- objects don't fit in a pipe
- object → message
- serialization / de-serialization cost
- message size
- queuing = cost

Use a mixture of both

» async + sync

stuff which matters !

2.1 Interlude

2.1 Interlude

2.1 Interlude

Processing large datasets :

Hadoop + Map/Reduce

Hadoop: HDFS + MapReduce

» single filesystem + single execution-space

M/R Execution

(c) Google

MapReduce example: WordCount

- Read a line of text, output every word

- Group all the values with each unique word

- Add up the values associated with each unique word

Copyright Concurrent, Inc. 2009. All rights reserved

Hadoop ecosystem

- Hadoop Common
- Subprojects
  - Chukwa: A data collection system for managing large distributed systems.
  - HBase: A scalable, distributed database that supports structured data storage for large tables.
  - HDFS: A distributed file system that provides high throughput access to application data.
  - Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying.
  - MapReduce: A software framework for distributed processing of large data sets on compute clusters.
  - Pig: A high-level data-flow language and execution framework for parallel computation.
  - ZooKeeper: A high-performance coordination service for distributed applications.
  - Mahout: machine learning libraries

Processing large datasets with MR

- Benefit from parallellisation
- Less modelling upfront (ad-hoc processing)
- Compartmentalized approach reduces operational risks
- AsterData et al. have SQL/MR hybrids for huge-scale BI

1. Intro
2. Theory
3. Technology
4. Experiences

We welcome the Polyglot Persistence overlords.

The NOSQL footprint

Categories

- key-value stores
- column stores
- document stores
- graph databases

Key-value stores

- Focus on scaling to huge amounts of data
- Designed to handle big loads
- Often: cfr. Amazon Dynamo
  - ring partitioning and replication
- Data model: key/value pairs

Key-value stores

- Redis
- Voldemort
- Tokyo Cabinet

Redis

- REmote DIctionary Server
- http://code.google.com/p/redis/
- vmware

Redis Features

- » persisted memcache, ‘awesome’
- » RAM-based + persistable
- » key → values: string, list, set
- » higher-level ops
  - » i.e. push/pop and sort for lists
- » fast (very)
- » configurable durability
- » client-managed sharding

Voldemort

- http://project-voldemort.com/
- LinkedIn

Voldemort

- persistent
- distributed
- fault-tolerant
- hash table

Voldemort

Logical Architecture
Client API
Conflict Resolution
Serialization
Routing & Read Repair
Faiover (Hinted handoff)
Storage Engine
(BDB/MySQL/Memory)
API: GET, PUT,
DELETE
Network Client & Server
(HTTP/Sockets/NIO)
(Optional)
?
?
Requests
Responses

Voldemort

routing logic moving up the stack,
smaller latency

Column stores

- BigTable clones
- Sparseness!
- Data model: columns → column families → cells
  - Datums keyed by: row, column, time, index
  - Row-range → tablet → distribution

Column stores

- BigTable
- HBase
- Cassandra

BigTable

- http://labs.google.com/papers/bigtable.html
- Google
- layered on top of GFS

HBase

- http://hadoop.apache.org/hbase/
- StumbleUpon / Adobe / Cloudera

HBase

- sorted
- distributed
- column-oriented
- multi-dimensional
- highly-available
- high-performance

- persisted
- storage system
- adds random access reads and writes atop HDFS

HBase data model

- Distributed multi-dimensional sparse map
- Multi-dimensional keys:
  `(table, row, family:column, timestamp) → value`

Row Key	Time Stamp	Column *"contents:"*	Column *"anchor:"*		Column *"mime:"*
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

- Keys are arbitrary strings
- Access to row data is atomic

Sample schema

Card Number
1233.45.33-23

TS	Card	Transaction	Owner	Support	Notes
t5	registerCode=<..> model=<model> pinHash=<md5(pin)> ValidFrom=<md5(f)> ValidTo=<...>				A new card was issued to the customer
t4		<DealerId>= <amount> Location=<..>	pinAttempts=<.>		A Transaction was made
t3				Reason=<...> <ReqId>= <status> Supporter=<..>	Support Request from Customer
t2			City=<...> Addressid=<..> src=byTel operator=<...>	Reason=<...> <ReqId>= Solved Supporter=<>	Address-Change by support
t1			Email= <md5(email)> src=web		Email-Change from WebSite

Card Number
1233.45.33-24

TS	Card	Transaction	Owner	Support	Notes
t5	registerCode=<..> model=<model> pinHash=<md5(pin)> ValidFrom=<md5(f)> ValidTo=<...>				A new card was issued to the customer
t4		<DealerId>= <amount> Location=<..>	pinAttempts=<.>		A Transaction was made
t3				Reason=<...> <ReqId>= <status> Supporter=<..>	Support Request from Customer

(c) eCircle

Storage architecture

© lars george

Cassandra

- http://cassandra.apache.org/
- Rackspace / Facebook

Cassandra

- Key-value store (with added structure)
- Reliability (identical nodes)
- Eventual consistent
- Distributed
- Tunable
  - Partitioning
  - Replication

Cassandra applicability

FIT	NO FIT
» Scalable reliability (through identical nodes)	» Flexible indexing
» Linear scaling	» Only PK-based querying
» Write throughput	» Big Binary Data
» Large Data Sets	» 1 Row must fit in RAM entirely

twitter
Mahalo
User profiles
Main real-time
data store
cassandra
Inbox search,
possibly more
facebook
Persistent cache
reddit
Cloud service,
monitoring&log storage
Main real-time
data store
rackspace
digg

User profiling system
Real-time data storage
and analytics platform
YAHOO!
NING
Near-duplicate document detection
Near-real time analytics
and reporting
HBASE
Versioned content
storage and retrieval
People search, analytics
twitter
Mahalo

Document databases

- ≈ K/V stores, but DB knows what the Value is
- Lotus Notes heritage
- Data model: collections of K/V collections
- Documents often versioned

Document stores

- CouchDB
- MongoDB
- Riak

CouchDB

- http://couchdb.apache.org/
- couch.io

CouchDB

- fault-tolerant
- schema-free
- document-oriented
- accessible via a RESTful HTTP/JSON API

CouchDB documents

```
{
  "_id": "BCCD12CBB",
  "_rev": "AB764C",
  "type": "person",
  "name": "Darth Vader",
  "age": 63,
  "headware": ["Helmet", "Sombrero"],
  "dark_side": true
}
```

CouchDB REST API

- HTTP
  - PUT /db/docid
  - GET /db/docid
  - POST /db/docid
  - DELETE /db/docid

CouchDB Views

- MapReduce-based
- Filter, Collate, Aggregate
- Javascript

map

```
function (doc) {
 for(var i in doc.tags)
   emit(doc.tags[i], 1);
}
```

reduce

```
function (Key, Values) {
 var sum = 0;
 for(var i in Values)
   sum += Values[i];
 return sum;
}
```

CouchDB

- be careful on semantics
  - replication ≠ partioning/sharding !
  - distributed database = distributable database
- sharded / distributed deployment requires proxy layer

MongoDB

- http://www.mongodb.org/
- 10gen

MongoDB

- cfr. CouchDB, really
- except for:
  - C++
  - performance focus
  - runtime queries (mapreduce still available)
  - native drivers (no REST/HTTP layering)
  - no MVCC: update-in-place
  - auto sharding (alpha)

Graph databases

- Focus on modeling structure of data - interconnectivity
- Scale, but only to the complexity of data
- Data model: property graphs

Graph databases

- Neo4j
- AllegroGraph (RDF)

Neo4j

- http://neo4j.org/
- Neo Technology

Neo4j

» data = nodes + relationships + key/value properties

Neo4j

- many language bindings, little remoting
- ‘whiteboard’ friendly
- scaling to complexity (rather than volume?)
- lots of focus on domain modelling
- SPARQL/SAIL impl for triple geeks
- mostly RAM centric (with disk swapping & persistence)

Bandwagonjumpers

- JCR / Jackrabbit
- GT/M
- RDF stores

Market maturization

Rise of integrators

- Cloudera (H-stack)
- Riptano (Cassandra)
- Cloudant (hosted CouchDB)
- (Outerthought: HBase)

VC capital

- Cloudera
- couch.io
- Neo
- 10gen
- many others

1 . Intro
2 . Theory
3 . Technology
4 . Experiences

the fireside conversations

http://www.flickr.com/photos/52641994@N00/516394238/

NOSQL applicability

- Horizontal scaling
- Multi-Master
- Data representation
  - search of simplicity
  - data that doesn't fit the E-R model (graphs, trees, versions)
- Speed

Tool selection

- be careful with the marketeese: smoke and mirrors beware!
- monitor dev list, IRC, Twitter, blogs
- monitor project 'sponsors'
- mix-and-match: polyglot persistency
- DON'T NOSQL WITHOUT INTERNAL SYS ARCHS & DEV(OP)S !

Our Context: Lily

- cloud-scalable content store and search repository
- successor (in many ways) of Daisy

Complexity

Complexity

Business Development 101

Solution

We Prefer Sophistication

» the challenge for us was to scale ...
without dropping features

The typical CMS ‘architecture’

database (+opt. filesystem) (+ opt. full-text indexes)

The typical CMS ‘architecture’

The typical CMS ‘architecture’

The typical CMS ‘architecture’

The typical CMS ‘architecture’

The typical CMS ‘architecture’

client (+cache)

even more cache

more cache

application | cache

database (+opt. filesystem) (+ opt. full-text indexes)

What we found hard to scale

- access control
- facet browsing
- all the nifty stuff people were using our software for
- ... anything that required random access to in-memory-cache data for computations

Beyond the ‘scaling’ problem

- three-prong data layer

- result set merging (between MySQL & Lucene)
  - happened in appcode/memory
- ‘transactions’, set operations = hard

Requirements, phase I

- *automatic* scaling to large data sets
- fault-tolerance: *replication*, automatic handling of failing nodes
- a flexible data model supporting *sparse data*
- runs on commodity hardware
- *efficient* random access to data
- open source, ability to participate in the development thus drive the direction of the project
- *some* preference for a Java-based solution

Requirements, phase II

- After careful consideration, we realized the important choices were also:
  - *consistency*: no chance of having two conflicting versions of a row
  - *atomic updates* of a single row, single-row transactions
  - bonus points for *MapReduce* integration
    - e.g. full-text index rebuilding

That brought us to HBase, which bought us:

- a *datamodel* where you can have column families which keep all versions and others which do not, which fits very well on our CMS document model
- *ordered tables* with the ability to do *range scans* on them, which allows to build scalable indexes on top of it
- HDFS, a convenient place to *store large blobs*
- Apache license and community, a familiar environment for us

» OK, so now we had a data store !

» However, content repository = store **+ search**

ouch!

That was easy ! (however ...)

Search ponderings

- CMS = two types of search
  - structured search
    - numbers, strings
    - based on logic (SQL, anyone?)
  - information retrieval (or: full-text search)
    - text
    - based on statistics

Search ponderings

» All of that, at **scale**

Structured Search

- HBase Indexing Library
  - idea from Google App Engine datastore indexes
  - http://code.google.com/appengine/articles/index_building.html

Full-text / IR search

- Lucene?
  - no sharding (for scale)
  - no replication (for availability)
  - batched index updates (not real-time)

Beyond Lucene

- Katta
  - scalable architecture, however only search, no indexing
- Elastic Search
  - very young (sorry)
- hbasene *et al.*
  - stores inverted index in HBase, might not scale all *features*
- **SOLR**
  - widely used, schema, facets, query syntax, cloud branch

More info: http://lilycms.org/lily/prerelease/technology.html

?

HBASE

+

Apache
Solr

=

Easy ! Or ?

Remember distribution ?
Remember secondary indexes ?

→ Need for reliable queuing

Connecting things

- we needed a **reliable** bridge between our main storage (HBase) and our index/search server(s) (SOLR)
  - indexing, reindexing, mass reindexing (M/R)
- we need a **reliable** method of updating HBase secondary indexes
- all of that eventually to run distributed
- distribution means coping with failure

Solution

- ACMEMessageQueue ? Bzzzzzt.
  We wanted ~~fault-safe~~ HBase persistence for the queues.
  Also for ease of administration.
- → WAL & Queue implemented on top of HBase tables

WAL / Queue

- WAL
  - **guaranteed execution of synchronous actions**
  - call doesn't return before secondary action finishes
  - e.g. update secondary actions
  - if all goes well, size = #concurrent ops
  - will be useful/made available outside of Lily context as well!
- Queue
  - **triggering of async actions**
  - e.g. (re)index (updated) record with SOLR back-end
  - size depends on speed of back-end process

The Sum

- Lily model (records & fields)
- mapped onto HBase (=storage)
- indexed and searchable through SOLR
- using a WAL/Queue mechanism implemented in HBase
- runtime based on Kauri
- with client/server comms via Avro

LILY

LILY

Architecture

LILY Architecture

Roadmap

Nearly there!

- June 7-8: release of learning material (architecture, model, API, Javadoc)
  - www.lilycms.org
  - bit.ly/lilyprerelease
- Tomorrow: WAL/queue
- Mid July = 'proof of architecture' release
- from there on, ca. 3-monthly releases leading up to Lily 1.0

bit.ly/lilyprerelease

lilycms.org/lily/prerelease.html

OUTER THOUGHT

HOME | PRE-RELEASE | ROADMAP

TECHNOLOGY | ARCHITECTURE | REPOSITORY MODEL | API TUTORIAL | HBASE MAPPING | JAVADOC

LILY PRE-RELEASE DOCUMENTATION

We're shooting for a first runnable Lily Content Repository release by mid July. That release will be a 'proof of architecture' and still have many rough edges. However, what will be available by then is:

- a runnable Lily setup
- the basic client/server infrastructure
- the content repository model implemented in HBase
- a mapping between HBase-stored Lily records and SOLR indexing/schema configuration
- a WAL/queue implementation, backed by HBase, that serves as a robust mechanism to pass information between HBase and SOLR

In the mean time, we hope you can still be a bit patient, or start learning about Lily by reading this prerelease documentation:

- **The technology choices** we made
- **The Lily architecture**
- **The repository model** Lily provides
- **API Tutorial**
- **HBase mapping**
- **Javadoc**

In case you have any questions, get in touch with us via **Twitter** or **mail**.

License

» Apache

Business model

- Consulting, mentoring, turn-key projects
  - audience: developers
- Strong focus on partner relations
  - targeting vertical markets
  - geographic coverage
  - SaaS offerings
- Markets: media, finance, insurance, govt, heritage ... LOTS of semi-structured data
- Not: OLAP

Reading material

- Amazon Dynamo, Google BigTable, CAP
- http://nosql.mypopescu.com/
- http://nosql-database.org/
- http://twitter.com/nosqlupdate
- http://highscalability.com/

Questions?

http://www.flickr.com/photos/leehaywood/4237636853/

Thanks for your attention !

» stevenn@outerthought.org

» @stevenn