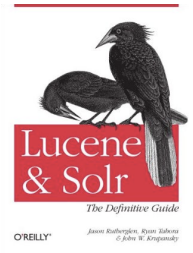
Search and Real Time Analytics on Big Data

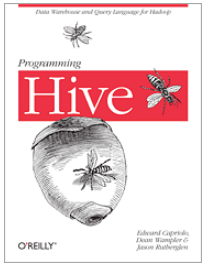
Ryan Tabora - Jason Rutherglen

Who are we?

Ryan Tabora

- Big Data Consultant
- Co-author of Lucene and Solr: The Definitive Guide from O'Reilly
- Think Big Analytics





Jason Rutherglen

- Co-author of *Programming Hive* and *Lucene and Solr: The Definitive Guide* from O'Reilly
- Search, mobile, Hadoop, cryptography, natural language processing, security, Hive
- Works on Datastax Enterprise

Ryan + Jason

The Plan

1. Search with Big Data
2. Product Landscape
3. Lucene and Solr Deep Dive
4. Break
5. Example Use Case
6. Scaling Search
7. Search with NoSQL
8. Performance Tuning

Ryan

About the Exercises

- Unix, Java, UI Based Exercises
- All Java projects are Mavenized (Maven 3.0 needed)
- Can be downloaded from S3 https://s3.amazonaws.com/thinkbig-academy/Strata2013/RealTimeSearchAndAnalytics-master.zip
- View the README files for detailed instructions
- Most exercises are intended to be run on the student's local environment

Ryan

Realtime Analytics

Search and Real Time Analytics on Big Data

Jason

What is Near Real Time Search?

- Low latency
  - Query Response
  - Data Availability
  - End-to-end response
- Could be nanoseconds, milliseconds, seconds, or minutes depending on your problem.

Jason

What is Big Data?

- The Buzz
  - “Big Data is the frontier of a firm's ability to store, process, and access the data it needs to operate effectively, make decisions, reduce risks, and serve customers.” [1]
  - Of course: The V’s (Volume, Velocity, Variety)
- The Reality
  - Data so big or analysis so compute intensive that traditional approaches can’t scale well or cheaply enough.

[1] http://blogs.forrester.com/mike_gualtieri/12-12-05-the_pragmatic_definition_of_big_data

Jason

Real World Example: Ticker Data

- Details about every trade
- Tick data generated real time and is quantitatively query-able
- Too big to query on in real time? Not anymore!

Jason

image src: http://25.media.tumblr.com/tumblr_mcmh3o1ktz1qj5rqko1_500.jpg

Tick Data Analytics - Moving Average

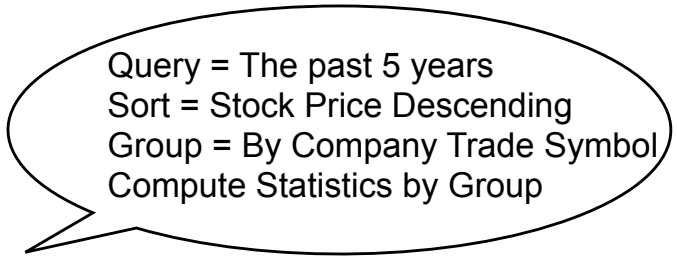
- Computing the moving stock price average in real time
- Comparing multiple moving averages for different stock_symbols
- Requires statistical analysis, group by companies, and faceting features

Jason

img src: http://www.trading-plan.com/images/moving_average_1.gif

Tick Data Analytics - Ad Hoc Searches

- Read latest ticks for a given company
- Query ticks for companies in specific verticals during large events such as press releases
- Compute deviation of stock data over 5 years for groups of companies



Jason

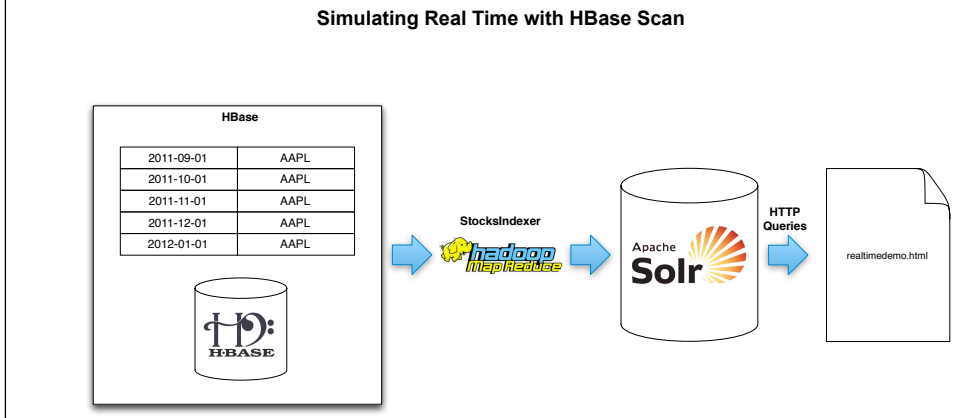
Real Time Demo Data

```
exchange,
stock_symbol,
date,
stock_price_open,
stock_price_high,
stock_price_low,
stock_price_close,
stock_volume,
stock_price_adj_close

NASDAQ,ABXA,2009-12-09,2.55,2.77,2.50,2.67,158500,2.67
NASDAQ,ABXA,2009-12-08,2.71,2.74,2.52,2.55,131700,2.55
NASDAQ,ABXA,2009-12-07,2.65,2.76,2.65,2.71,174200,2.71
NASDAQ,ABXA,2009-12-04,2.63,2.66,2.53,2.65,230900,2.65
NASDAQ,ABXA,2009-12-03,2.55,2.62,2.51,2.60,360900,2.60
```

Ryan

Real Time Demo Flow



Ryan

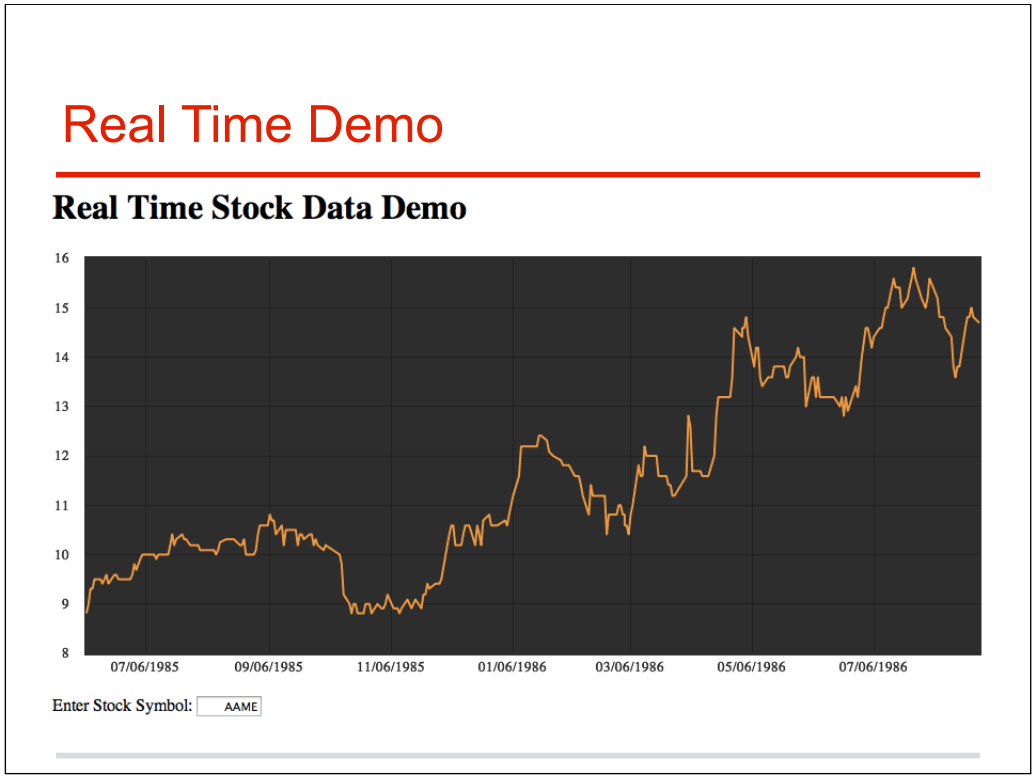
Real Time Demo Queries

Query = Stock Symbol:[User Input]
Sort = Descending Date
Response Format = JSON

Every 5ms via HTML GET

Ryan

Real Time Demo



Ryan

Real Time Search Demo

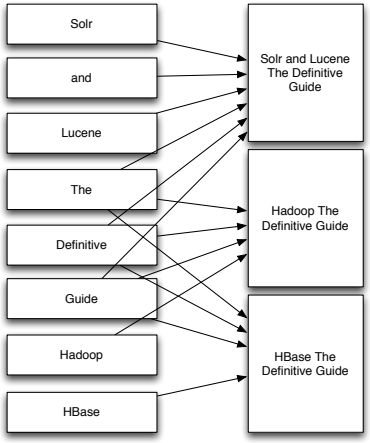
/07- real-time

Instructor Only
(But the code is there if you really want to run it!)

Ryan

What Does Search Mean?

- Querying terms that are not the unique identifier or key
- Inverted Index

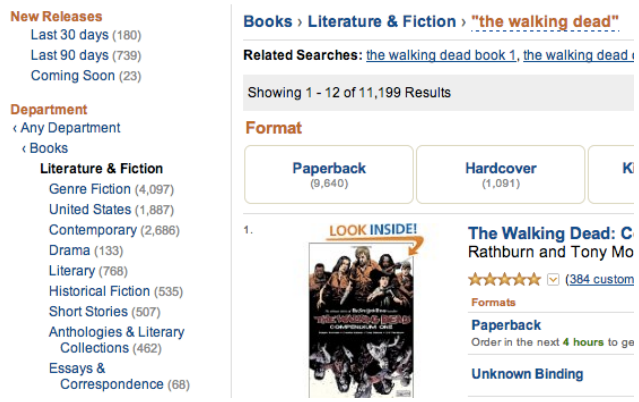


Ryan

An inverted index is like the index in the back of the book. Its a list of terms that point to pages in the book.

What Can You Do With Search?

- Facets
- Amazon, CNET, etc

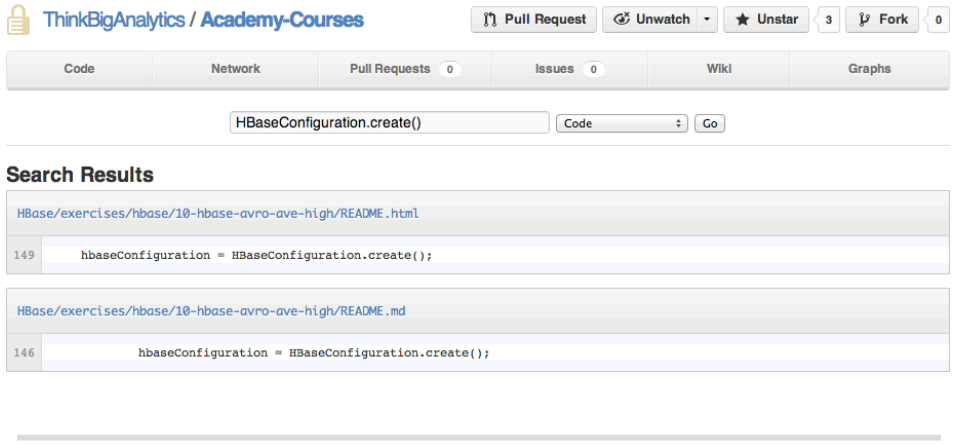


Ryan

Faceting is an extremely powerful form of search that many take for granted
CNet whom you will learn helped create Solr pioneered faceting

What Can You Do With Search?

- Text Search
- Github code search, Google, etc



Ryan

Tokenized search is just a small piece of search, there is much more you can do with text based search. More on this later.

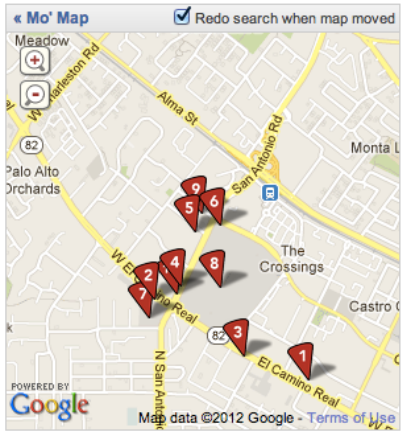
What Can You Do With Search?

- Image
- Google, Tineye

Ryan

What Can You Do With Search?

- Geospatial
- Google, Yahoo, Yelp



Ryan

Moving the map in Yelp to query which restaurants are where

Where is Search in NoSQL?

- Many popular NoSQL datastores have limited or no search capability at all
- HBase scan/get by rowkey
- Cassandra secondary indexing

Ryan

That leads us to our next slide, what can we use to integrate search?

We Want Our SQL Back!

- Hive provides SQL like queries over data in HDFS (and others) via MapReduce
- Hive allows users to JOIN data
- But Hive is batch oriented

Ryan

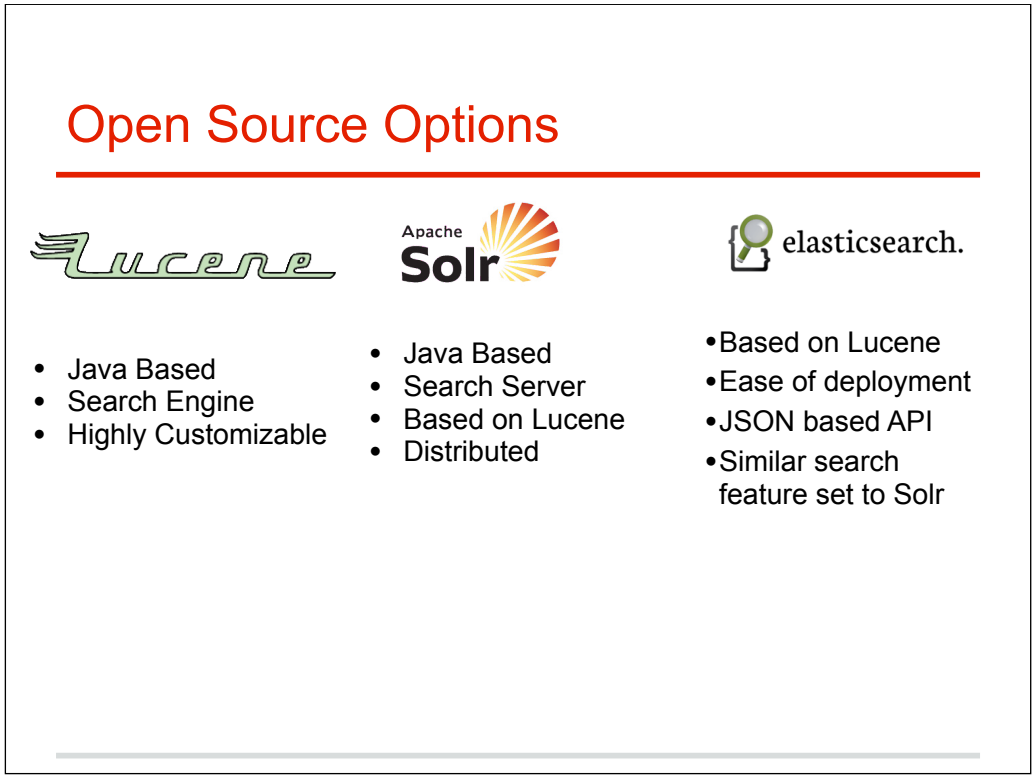
Product Landscape

Search and Real Time Analytics on Big Data

So lets talk about some of the search technologies that might help us get there.

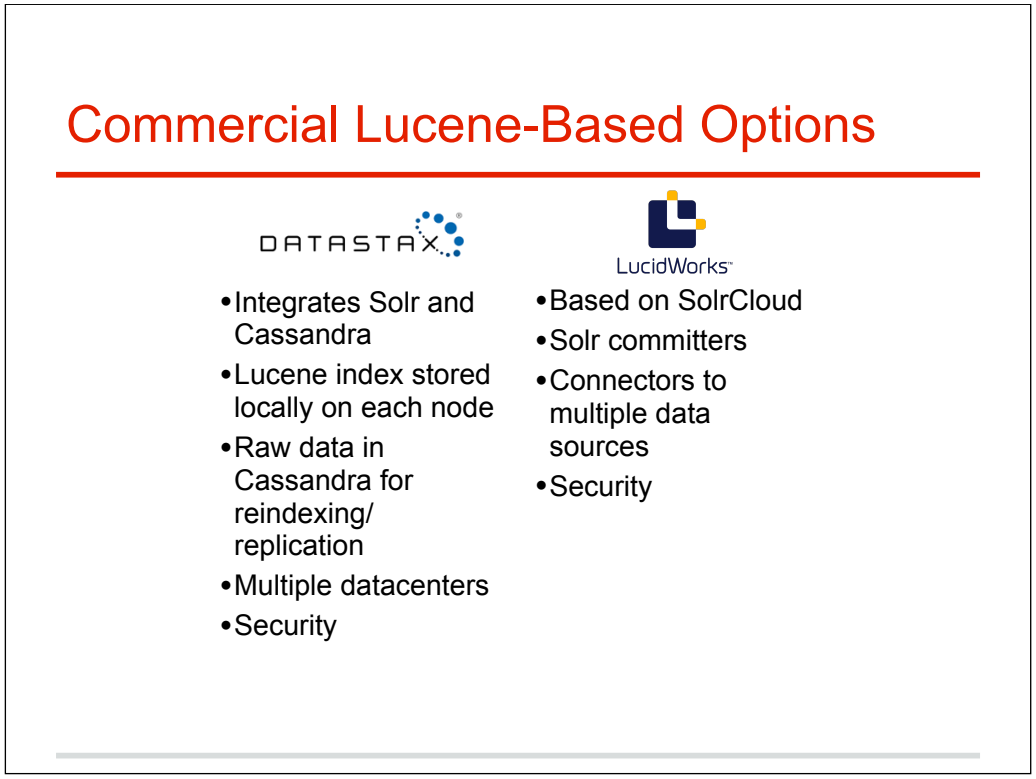
Jason

We can broadly categorize the search landscape into two groups. Lucene based and non-Lucene based. In this talk we are going to focus on the Lucene based solutions.



Jason

Elastic Search is the main competitor for Solr, and its founded on bringing up a distributed cluster with as much ease as possible. With SolrCloud, Solr and ElasticSearch have very similar feature sets.



Jason

Some projects we've heard of:
Cloudera + Solr
MapR + LucidWorks
Greenplum/Pivotal Labs+ Solr

Storing the Data

- Distributed big data processing framework
- Batch Oriented
- MapReduce

- NoSQL Datastore
- Based on Hadoop
- Master slave architecture
- Real time random data access
- Lookup by rowkey only
- Efficient scans
- MapReduce

- NoSQL Datastore
- Peer to peer architecture
- Real time random data access
- Secondary indexing
- User configurable CAP tradeoffs

Jason

Lucene and Solr Deep Dive

Search and Real Time Analytics on Big Data

First - Lucene

- High performance inverted index
- Java based
- Embeddable library
- Collection of jar files

Ryan

Solr is an Apache licensed search server with Lucene at the core.
Where Lucene is a search library with no dependencies, Solr has dependencies on other libraries.
The default method of running Solr is as a J2EE web application, however Solr may also be embedded in other Java applications.

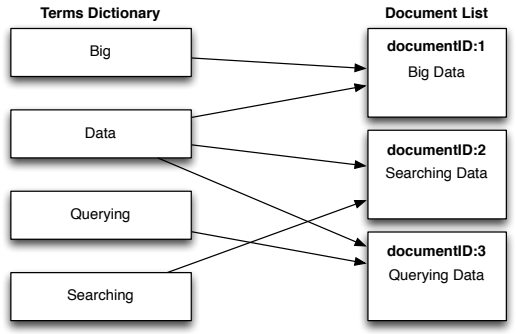
History of Lucene

- Started by Doug Cutting in 1999, Apache Lucene later became a top-level Apache project in February of 2005.
- Used as a part of Nutch (web crawler).

Ryan

Indexing Basics - Lucene Segments

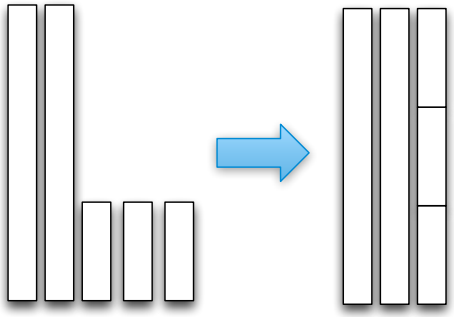
- Lucene stores the index in discrete units called segments
- Each segment is a complete index in itself
- Segments contain an inverted index



Ryan

Lucene File System

- Log structured merge tree
- Written once and immutable
- Segments merge as the index grows



Ryan

http://www.youtube.com/watch?v=YW0bOvLp72E&feature=player_embedded

Lucene Documents

- Essentially a collection of fields
- Field consists of a Field Type, Name, and Value
- Field Types include...
  - IntField
  - ByteDocValuesField
  - TextField
  - StringField
  - StoredField
  - ...

Ryan

Expert: directly create a field for a document. Most users should use one of the sugar subclasses: `IntField`, `LongField`, `FloatField`, `DoubleField`, `ByteDocValuesField`, `ShortDocValuesField`, `IntDocValuesField`,`LongDocValuesField`, `PackedLongDocValuesField`, `FloatDocValuesField`, `DoubleDocValuesField`, `SortedBytesDocValuesField`, `DerefBytesDocValuesField`, `StraightBytesDocValuesField`,`StringField`, `TextField`, `StoredField`.
A field is a section of a Document. Each field has three parts: name, type and value. Values may be text (String, Reader or pre-analyzed TokenStream), binary (byte[]), or numeric (a Number). Fields are optionally stored in the index, so that they may be returned with hits on the document.

Analyzers

- Convert text into tokens
- Records the position of each token
- Filters tokens as per configuration/design
- Can be applied when text is indexed or queried
  - Ex. Indexed as lower cased, queries lower cased at query time

Ryan

Analyzers Components

- Character Filters
  - Transformations before tokenizing
- Tokenizer
  - Breaks text into terms
- Token Filters
  - Transformations on the output of the tokenizer

Ryan

Tokenizer: Breaks the long string of input into discrete chunks or terms.
Character filters: Performs character transformations on the raw input string before it is tokenized.
Token Filter: Performs one transformation on the stream of tokens output by the tokenizer, each executed in the order specified.

Analyzers Use Cases

- Stemming - beyond simple plurals, including identification of root words
- Stop word removal - to reduce the size of the index and improve matching of similar text
- Eliminating accent marks for non-English text
- Arbitrary transformations using regular expressions
- Splitting terms based on
  - Embedded punctuation
  - Case changes
  - Changes between letters and digits

Ryan

Provided Filters and Tokenizers

Some simple ones are:

- WhitespaceTokenizer
- StopFilter
- LowerCaseFilter
- StandardTokenizer

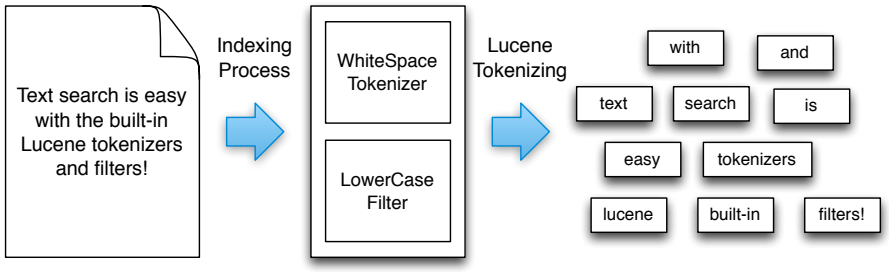
Ryan

CommonGrams

Construct bigrams for frequently occurring terms while indexing. Single terms are still indexed too, with bigrams overlaid. This is achieved through the use of `PositionIncrementAttribute.setPositionIncrement(int)`. Bigrams have a type of `GRAM_TYPE`
Example:

- input:"the quick brown fox"
- output:|"the","the-quick"|"brown"|"fox"|
- "the-quick" has a position increment of 0 so it is in the same position as "the" "the-quick" has a term.type() of "gram"

Analyzers Example



Ryan

Lucene IndexWriters and Directories

- IndexWriters create the index (Lucene Segments)
- Directories represent the location of the Lucene index
  - FSDirectory
  - RAMDirectory
  - NRTCachingDirectory
- All IO goes through Directory

Ryan

NRT Caching Directory: This class is likely only useful in a near-real-time context, where indexing rate is lowish but reopen rate is highish, resulting in many tiny files being written. This directory keeps such segments (as well as the segments produced by merging them, as long as they are small enough), in RAM.

Query Types

QueryName	Description
TermQuery	Matching a term
BooleanQuery	AND, OR NOT Functionality
WildcardQuery	Searching with W*LDCARD*
PhraseQuery	Searching for a Sequence of Terms
PrefixQuery	Searching for Pre*
FuzzyQuery	Searching for Like Terms
RegexpQuery	Regular Expression Matches
NumericRangeQuery	Self Explanatory
...	...

Ryan

Lucene Exercise

/01-lucene-basics

Jason

Hang on...

That was pretty advanced for such a simple query....

...isn't there an easier way to do this?

Ryan

Ryan

Distributed Search, Facets, Schemas, Group by (features Lucene does not have built in)

History of Solr

- Created at CNET in 2004, and graduated from Apache incubation status in 2007.
- March 2010 Lucene and Solr were merged as Apache projects

- Ryan

Core Solr Features

- Schema
- Extensions to Lucene Query Language
- Geospatial Search
- Advanced Text Analysis
- Web Administrative GUI
- Distributed Search
- JSON, XML, CSV support
- REST API

Ryan

Automatic, manual, and configurable relevancy boosting, including complex function queries
Schema for declaring and managing data and document structure, fields, and field types
Typed dynamic fields in addition to fully-typed schema
Optional “passthrough” so that undeclared data can be supported, if needed for the application
Multiple indexes (collections or tables) in a single server Sorting
Faceting
Highlighting of document snippets
Result Grouping and field*based collapsing of search results Spellcheck
Autocomplete
More Like This (Find Similar)
Debugging
Statistics
Term analysis
Extraction of data from rich text documents (Office, PDF, web pages) using Apache Tika
Extensive caching support
Powerful text analysis for both indexing and query
Distributed indexing and queries with partitioning/shards and replication for scal- ing
Geospatial search
Clustering of results
Multiple query formats for varying application requirements
Extensive support for non*European languages
Web*based administrative console with development and debugging features
Automatic but configurable support for advanced Lucene features

Solr Benefits

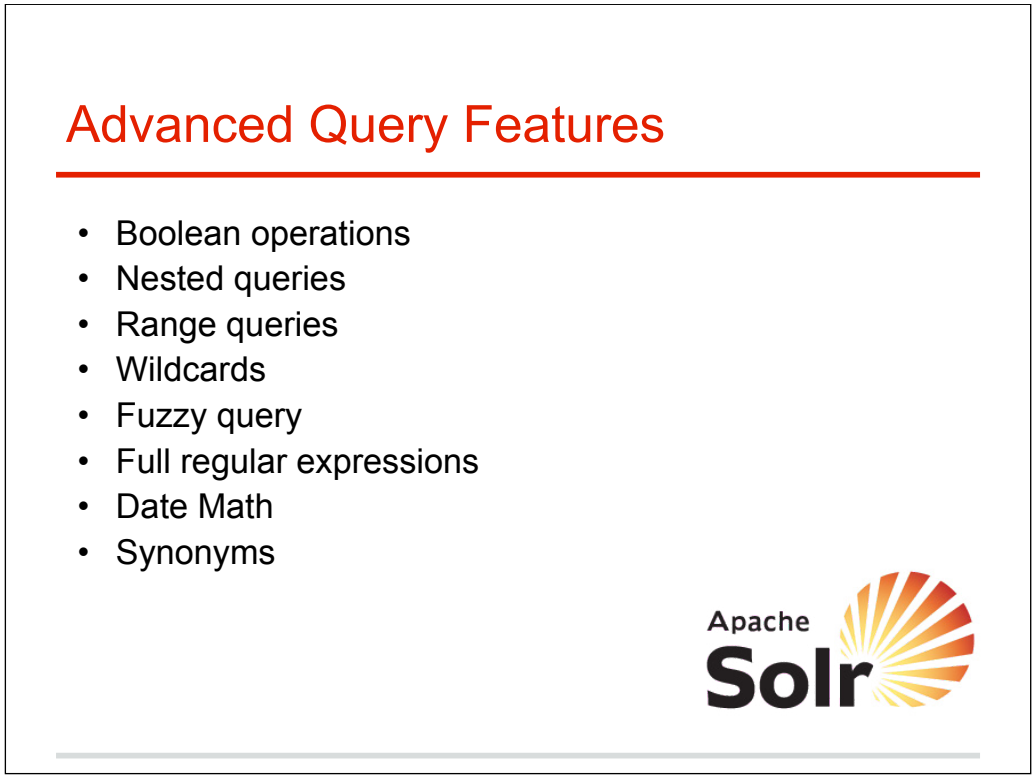
- Open Source
- Cheap (free)
- It scales to billions of documents
- Optimized for high-volume Web traffic
- Fast (extensive performance optimizations)
- Java Based
- Commercial vendors providing training, support, and consulting for corporate customers

Ryan

Think Big supports/trains Solr!


Who Uses Solr?
reddit
StubHub!
cnet
comcast
Pinterest
NETFLIX
THE WHITE HOUSE
WASHINGTON
Instagram

Ryan

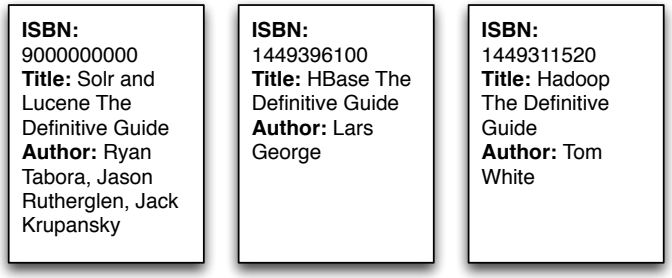


Ryan

Boolean operations - AND, OR, NOT, +, -, ()
Nested queries
Range queries, including date range
Numeric as well as raw string and tokenized text
Wildcards
Fuzzy query
Full regular expressions
Phrases, with optional “slop”
Stemming/plurals
Stopword removal
Accent and diacritical mark removal
Synonyms
Date math
Ability to explain how a document score was derived
Debugging

Solr Documents

- Lucene Indexes Solr Documents
- Documents consist of fields
- Fields consist of a name and one or more values



Ryan

Schema Type Options

- DynamicFields
  - Flexible schema
- CopyFields
  - Different analyzers for same field
- Field types
  - Strings
  - Integers
  - Dates
  - Trie fields
  - ...

Ryan

The schema also provides the ability to copy fields so that the same data can be indexed in different ways, or to be able to more efficiently search a number of fields by automatically copying them into a single search field.

Solr also permits dynamic fields, allowing the developing to automatically associate various field types with dynamic field names based on prefix and suffix patterns. The developer can choose whether to allow such fields, as well as whether to allow dynamic fields with arbitrary names.

The Solr schema file for a collection primarily details the fields and their field types. In other words, what does the data look like and how is it organized.

Solr comes with a number of built-in data types, including strings, integers, floating point, date, boolean, and text. There are specialized forms for many of the built-in types.

Developers can also add their own field types by developing plug-ins in Java.

The text field type (actually, a whole family of types) is special in that a variety of transformations are typically needed to permit efficient and flexible searching of text. These transformations are performed by specialized processing sequences called analyzers, and are typically composed of a tokenizer and a sequence of filters.

The schema will also declare which field is to be used as a unique key for the collection. The default is "id".

Advanced Schema Types

- Custom Field Types
- Text Fields
- Analyzers
  - Tokenizers
  - Filters

Ryan

Changing the Schema

- Adding fields is okay
- Changing existing fields requires a complete reindex
  - Think new analyzers, tokenizers, filters
- Can be costly
  - Time to develop custom reindexing application
  - Time to actually perform the reindexing

Ryan

Solr Schema Exercise - Types

```
<types>
...
    <fieldType name="string"
class="solr.StrField"/>

        <fieldType name="date"
class="solr.TrieDateField"/>

    <fieldType name="boolean"
class="solr.BoolField"/>
...
 </types>
```

/02-solr-schema/schema.xml

Ryan

Solr Schema Exercise - Types

```
<types>
...
  <fieldType name="edgetext" class="solr.TextField"
positionIncrementGap="100">
    <analyzer type="index">
      <tokenizer class="solr.KeywordTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.EdgeNGramFilterFactory"
minGramSize="1" maxGramSize="25" />
    </analyzer>
    <analyzer type="query">
      <tokenizer class="solr.KeywordTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
    </analyzer>
  </fieldType>
...
 </types>
```

/02-solr-schema/schema.xml

Ryan

Solr Schema Exercise - Fields I

```
 <fields>
...
   <field name="name" type="string"
indexed="true" stored="true"/>

   <field name="description" type="text"
indexed="true" stored="true"/>
...
 </fields>
```

/02-solr-schema/schema.xml

Ryan

Solr Schema Exercise - Fields II

```
<fields>
  ...
   <dynamicField name="*_s" type="string"
indexed="true" stored="true" />

   <dynamicField name="*_i" type="int"
indexed="true" stored="true"/>
  ...
</fields>
```

/02-solr-schema/schema.xml

Ryan

Solr Schema Exercise - Fields III

```
<fields>
...
  <field name="text" type="text"
indexed="true" stored="false"
multiValued="true"/>
...
</fields>
...
<copyField source="id" dest="text"/>
<copyField source="name" dest="text"/>
<copyField source="description" dest="text"/>
```

/02-solr-schema/schema.xml

Ryan

Solr as a Database?

- Indexed vs. Stored
  - Retrieving stored fields can be expensive
- Stores the raw data alongside the index
- Storing entire text blocks could impact query performance

Ryan

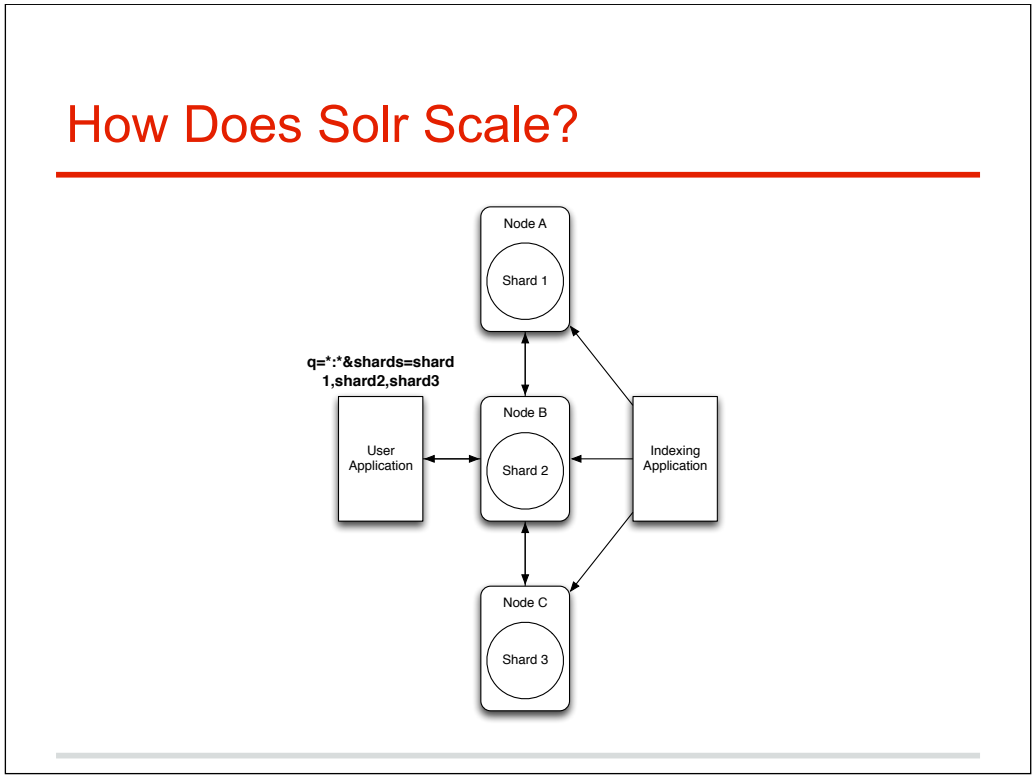
Solr will store the actual raw data alongside the index if you want, however this can greatly increase your storage profile as well as your response times (sending more data over the wire)

There are tradeoffs, storing little bits of the raw data can be good. Storing entire text blocks could be detrimental to your application.

Scaling Search

Search and Real Time Analytics on Big Data

Ryan



Ryan

The index is broken into shards. Essentially these are slices of the index.

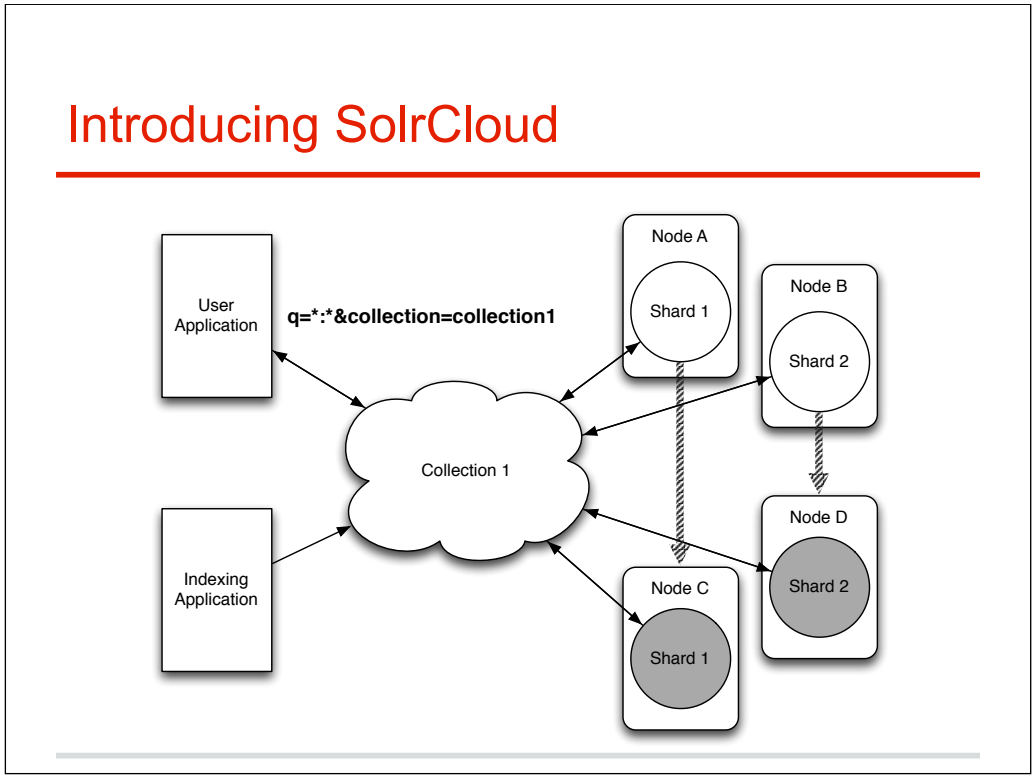
When users create distributed queries, the query is sent to all shards that make up the logical index. Query results are merged and returned to the client.

Many Solr clusters in production look this way.

Core Pre-Cloud Issues

- Manually managing shard creation
- Manually managing replication
- Manually managing index partitioning
- Manually managing query balancing

Ryan



Ryan

SolrCloud focuses on handling the sharding logic automatically. Instead of querying a set of shards, we now query a collection. Replicas are created automatically.

Core Cloud Features

- Automatically generates replica cores
- Automatically partitions your index
- Handles syncing index between cores and their replicas
- Load balances queries to cores and their replicas
- Centralized schema management across cores
- Integrating ZooKeeper
- Introduces an optional transaction log for write durability (and real time gets)

Ryan

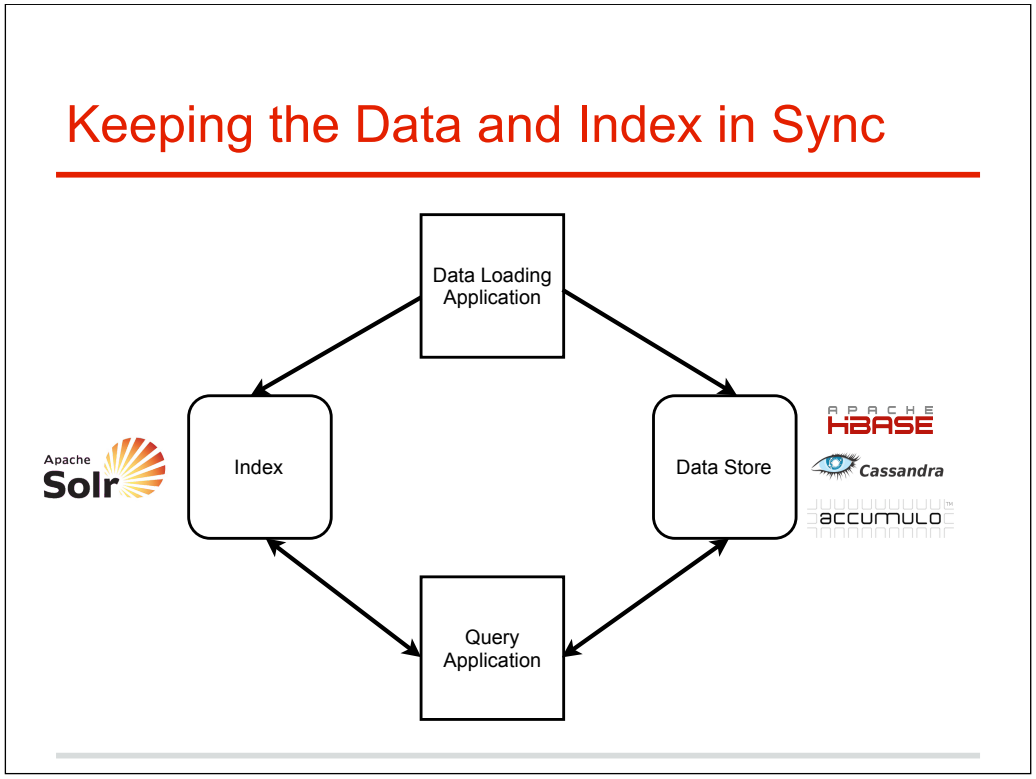
Distributed Solr Limitations

- Fixed number of shards (Adding more nodes)
- Join
- IDF
- QueryElevationComponent
- MoreLikeThis
- Still a work in progress

Ryan

Search with NoSQL

Search and Real Time Analytics on Big Data

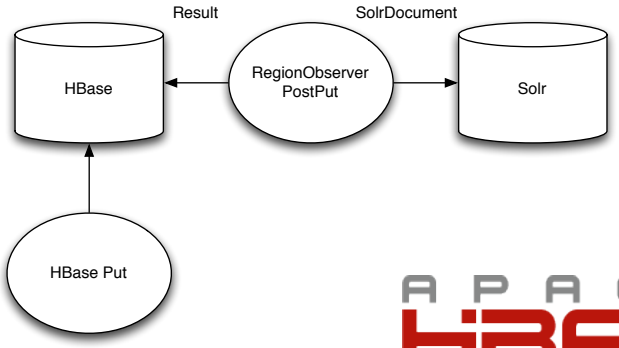


Ryan

What happens when you have updates/deletes outside of the data loading application?

Consider HBase + Solr

- Coprocessors
- Essentially like triggers/storedprocs



APACHE HBASE

Ryan

HBase and Solr Desired Features

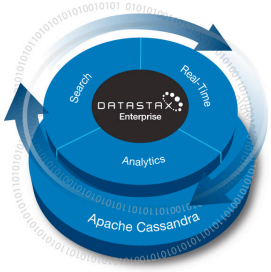
- Storing raw fields in HBase, indexing in Solr
- Updates to HBase trigger updates in Solr (and vice versa)
- Building Lucene index in Hadoop (SOLR-1301)
- Syncing Solr shards with HBase regions
- Shard creationg/balancing with region splitting
- Mapping HBase qualifiers to Solr types
- Reindexing with MapReduce on HBase

Ryan

Consider Cassandra + Solr

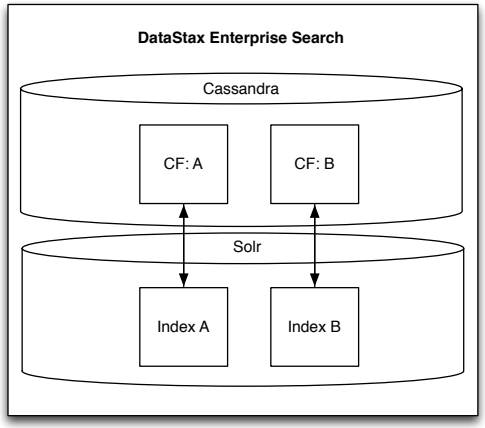
The work has already been done!

DATASTAX



Jason

DataStax Architecture



DATASTAX

Jason

SolrCloud vs DataStax

- Open Source
- Zookeeper
- Not meant for data storage
- Consistency, Persistence

DATASTAX

- Multiple datacenters
- Peer Architecture
- Cassandra is a proven NoSQL data store
- Availability, Persistence (tunable)
- Reindexing
- No fixed shard count

Jason

Starting Up Your Own Solr Instance

/03-installing-solr

Ryan

Solr UI

- Ping
- Schema
- Solrconfig
- Analysis
- Creating/Dropping Cores

Ryan

Exploring the Solr UI

/04-solr-ui

Ryan

How to Index Documents

- Manually build the Lucene Index
- Use Solr APIs like SolrJ and submit SolrDocuments

```
SolrDocument solrDoc = new SolrDocument();
solrDoc.addField("id","1234");
solrServer.add(solrDoc);
solrServer.commit();
```

Ryan

Loading Solr

/05-solr-index

Ryan

Solr Query Features

- Search on any number of fields with boolean logic (AND, OR + -)
- Sort results per field similar to SQL
- Range queries
- Phrase queries
- Regular expression queries
- Query boosting (DisMax)

Jason

Basic Queries

- Select All Query
  - q=*:*
  - **SQL:** SELECT * FROM core
- Single term query
  - q=name:ryan
  - **SQL:** SELECT * FROM core WHERE name = ‘ryan’
- Multiple Fields
  - q=(+first_name:ryan +last_name:tabora)
  - **SQL:** SELECT * FROM core WHERE first_name = ‘ryan’ AND last_name = ‘tabora’

Jason

And Or Not Logic

- **And** logic - Name field containing both tokens
  - q=name:(+rick +grimes)
  - **SQL:**... WHERE name = ‘rick’ AND name = ‘grimes’
- **Or** logic - Subject field containing either token
  - q=subject:(pirates zombies)
  - **SQL:**... WHERE subject = ‘pirates’ OR subject = ‘zombies’
- **Not** logic - Query for the test_results field that do not included the token pass
  - q=-test_results:pass
  - **SQL:**... WHERE NOT test_results = ‘pass’

Jason

Range Queries

- Query a **numerical range**
  - salary:[100000 TO 150000]
  - **SQL:** ... WHERE salary >= 100000 AND salary <= 150000
- Query **from** a start date **to anything beyond** it
  - date:[1999091091T23:59:59.999Z TO *]
  - **SQL:** ... WHERE date >= 1999091091T23:59:59.999Z

Jason

Range Queries (Continued)

- Query for **anything lower** than a number
  - stock_price_close:[* to 10]
  - **SQL:**... WHERE stock_price_close <= 10
- Query for any document **without a value** for the field
  - -amount_paid:[* TO *]
  - **SQL:** WHERE amount_paid IS NULL

Jason

Sort By

- Get the **latest** critical documents
  - q=priority:critical&sort=true&sort.field=date desc
  - **SQL:** WHERE priority = ‘critical’ ORDER BY date DESC
- Get a list of students sorted **alphabetically**
  - q=role:student&sort=true&sort.field=last_name asc
  - **SQL:** WHERE role = ‘student’ ORDER BY last_name ASC
- Get the **highest** traded stock **values** for the day
  - q=stock_symbol:NYSE&sort=true&sort.field=stock_price_close desc
  - **SQL:** WHERE stock_symbol = ‘NYSE’ ORDER BY stock_price_close DESC

Jason

Group by (and group sorting)

- Query on all disk **devices grouped by server**_id
  - q=device_type=disk&group=true&group.field=server_id
  - **SQL:** WHERE device_type = ‘disk’ GROUP BY server_id
- Query on all **companies sorted alphabetically**, and **documents sorted by date**
  - q=state:wisconsin&group=true&group.field=company_name&group.sort=date desc&sort=company_name desc
  - **SQL:** WHERE state = ‘wisconsin’ GROUP BY company_name ORDER BY company_name, date DESC

Jason

Group by StatsComponent

- StatsComponent with facets enables group by with aggregations
- Does not support ordering of the grouping
- Query on all stock prices **grouped by** stock_symbol with all aggregations including average, sum, count
  - stats=true&stats.field=stock_price_open&stats.facet=stock_symbol&q=*:*
  - **SQL:** FROM stocks GROUP BY stock_symbol
  - **SQL:** SELECT stock_symbol, SUM(stock_price_open) FROM stocks GROUP BY stock_symbol
  - **SQL:** SELECT stock_symbol, COUNT(stock_price_open) FROM stocks GROUP BY stock_symbol

Jason

Filter Queries

- Cached bit sets
- No score calculated
- Good for queries that are reused such as types or **access controls**
  - q=product_name:necronomicon&fq=customer_id:s-mart

Jason

Phrase Query

- Search for “big” and “data” within 4 words of each other
  - “big data”~4

Jason

Prefix Queries

- Find all monster types starting with DRA
  - q=monster_type:DRA*
  - Results = DRAGON, DRACULA, etc.
  - **SQL**: ... WHERE monster_type LIKE ‘DRA%’
- Queries cannot begin with an asterisk

Jason

Regular Expressions

- Use forward slash to demarcate a regex query
- Match on a five digit zip code
  - body:/[0-9]{5}/

Jason

Facets

- Intersection count of another query
- Commonly seen on shopping and other web sites
- Solr supports multi-select faceting
- Range faceting

Jason

Facets Parameters

- Facet = true
- Facet.field = fields comma separated
- Facet.query = query to facet on
- Facet.method = enum, fc, fcs

Jason

Highlighting

- Highlighting re-analyzes each document
- Fast vector highlighter is faster however it requires more storage

Jason

Highlighting Parameters

- hl = true
- hl.fl = fields comma separated
- hl.useFastVectorHighlighter = true/false

Jason

Debug Query

- Pass in debug=true
- Provide info about timing of components
- Debug info about the query
- Debug info about the result scoring

Jason

Auto Suggest

- Use SpellCheckComponent
- Spellcheck/suggest is built from an existing index
- Can be set to automatically rebuild the suggest index on commit

Jason

Prefix Auto Suggest

- It is recommended to use FSTLookup or WFSTLookup
- They are more memory efficient

Jason

Auto Suggest Parameters

- Spellcheck = true
- Spellcheck.dictionary = suggest
- Spellcheck.onlyMorePopular = true
- Spellcheck.count = 5 (number of returned suggestions)
- StringField = UTF8Type

Jason

AutoSuggest by Popular Queries

- Prefix based auto-suggest can be limiting
- Use EdgeNGramFilterFactor to query within terms
- Sort Results by a hit count field

Jason

Dismax Query Parser

- Dismax query parser provides query time field level boosting granularity, with less special syntax
- Dismax generally makes the best first choice query parser for user facing Solr applications
- Boosting is the ability to increase the relevance of terms from specific fields over others

Jason

Perform some queries yourself!

/06-sql-to-solr

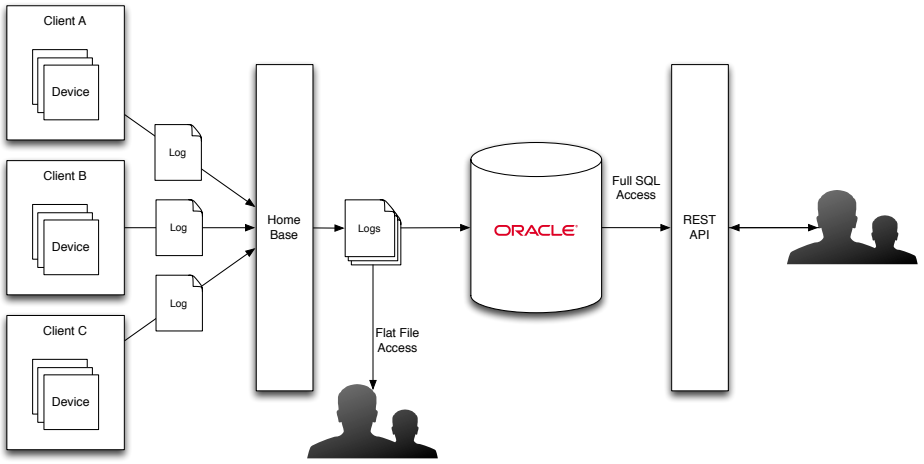
Ryan

Example Use Case

Search and Real Time Analytics on Big Data

Ryan

Use Case: Device Data



Ryan

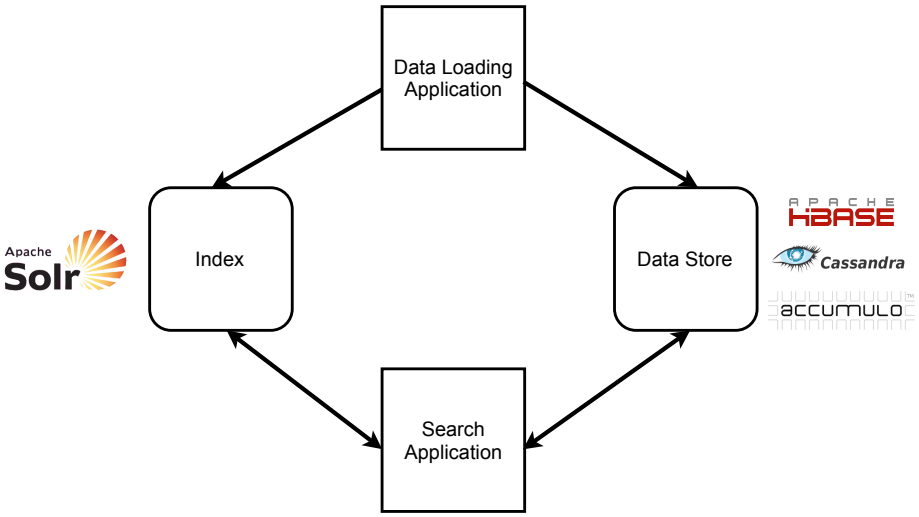
Billions of incoming logs increasing greatly over time

Each log is a significant file size (<10MB)

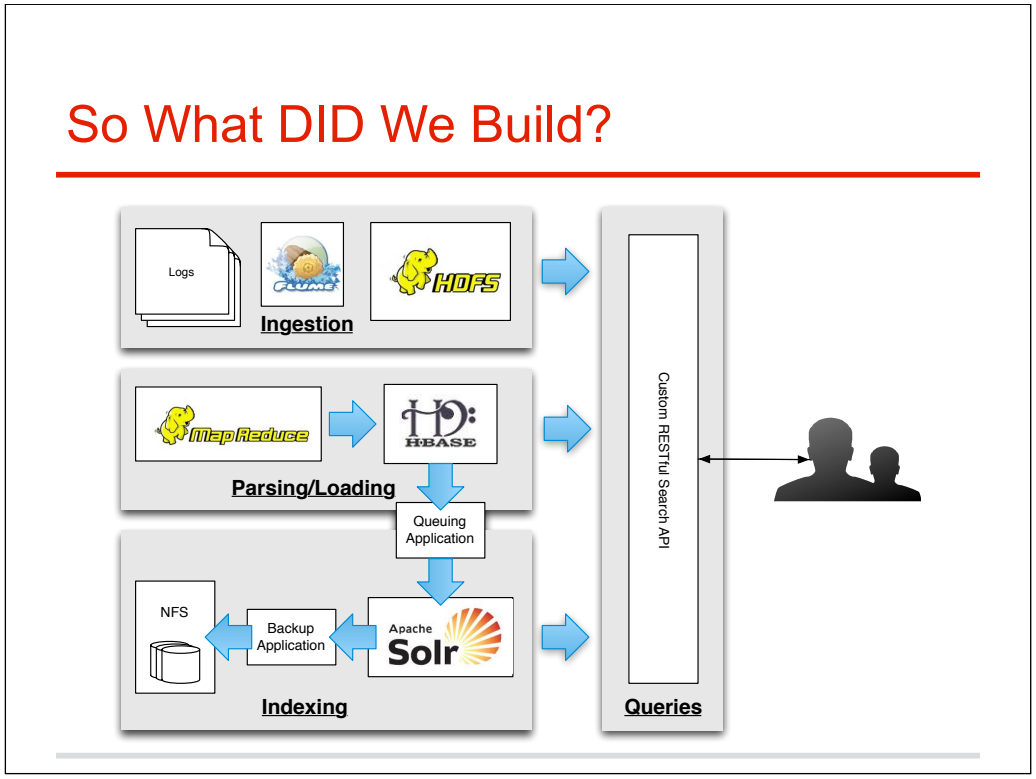
Required to index many attributes for each log

Required to store parsed and raw log data

So What Do We Need To Build?



Ryan



Ryan

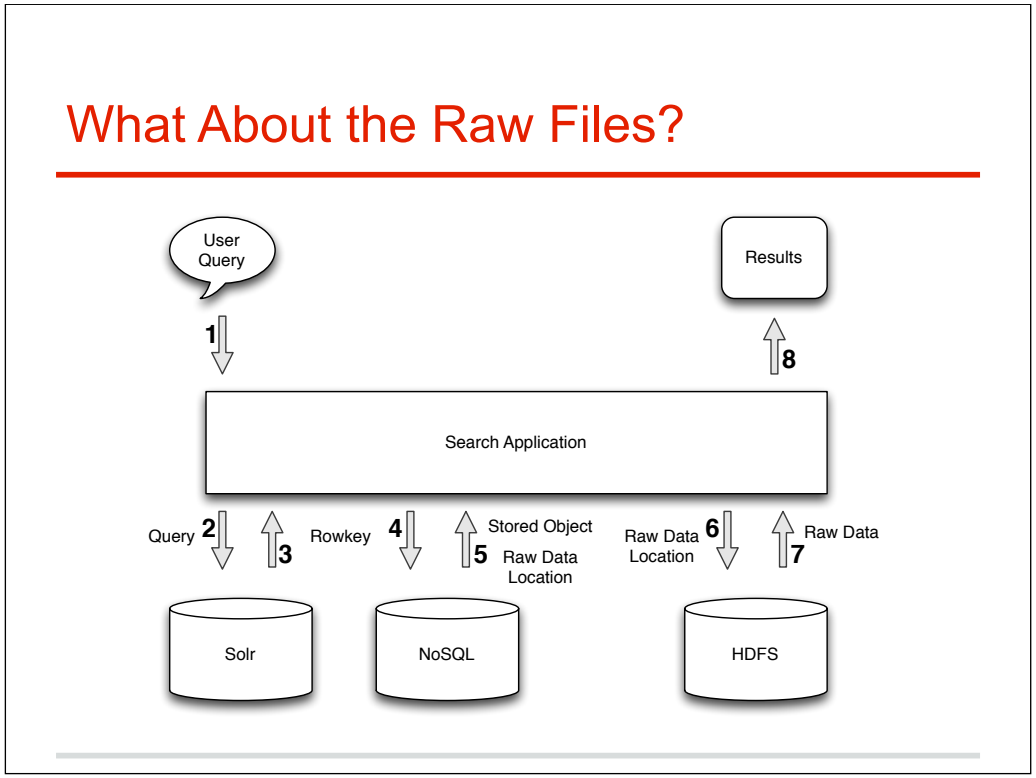
Disclaimer: This was designed and built PRIOR to the announcement of SolrCloud/DSE 2.0.

After those, we really didnt need the queueing application or the backup application

The Search Application

- Searching on log subject lines across install base
- Searching on latest logs across all machines for a given customer that have created support tickets
- Retrieving the raw log data for a given section in a log for a cluster

Ryan



Ryan

. User Query
+
The user would send an HTTP formatted query to the REST API.

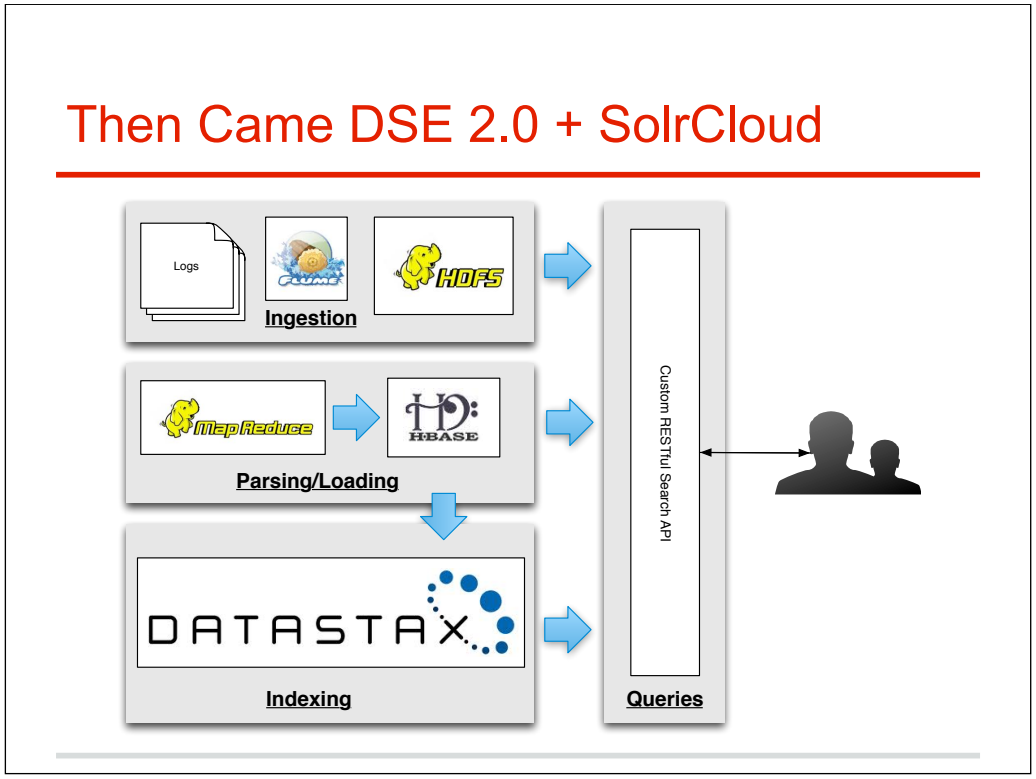
. Solr Query
+
The user defined query string would then be translated into a Solr query. Each API was very unique so a generic Solr class was required that would take in a generic set of attributes and create the proper Solr query from it.

. Rowkey
+
The SolrDocuments contained in the Solr response included the unique rowkey that identified ASUPs in the HBase schema.
. HBase Query
+
The REST API would then gather all of the rowkeys contained in the Solr response and use those to query HBase.

. Stored Object/Raw Data Location
+
The HBase response included not only the stored object for each ASUP but also a pointer to the location of the raw ASUP file located in HDFS.

. Read HDFS
+
The location of the raw ASUP in HDFS was used to read the file from HDFS.

. Formatted Results
+
The REST API arrage the stored fields from Solr, the objects from HBase, and raw data from HDFS in an XML formmated response.



Ryan

After the previously described architecture was finalized, DataStax announced DataStax Enterprise 2.0 which had integrated Solr into Cassandra. The team eventually decided to move forward with DataStax Enterprise for several reasons:

* Commercially Supported - Apache SolrCloud is still relatively new and not supported. Standard sharding through Apache Solr required a great amount of custom code, which is not always very supportable.
* Failure Tolerance via Cassandra - Since the indexed data is stored in Cassandra, you can lose a node without losing any data. In standard Apache Solr, losing a node meant losing a portion of your index. Dozens of nodes = dozens of single points of failure.
* Automatic Reindexing from Stored Data - DataStax Enterprise can automatically reindex based off of the data stored in Cassandra. This meant they could change the schema whenever they wanted and reindex automatically.
* Ease of Adding Nodes to the Cluster - Adding a Solr Shard is as easy as adding a node to Cassandra. No need to manually reindex or manually load balance. It is all taken care of within DataStax.
* Complete support for Solr - DataStax Enterprise supports all of the features included in Solr 4.0. Code developed against Apache Solr would be 100% compatible with DataStax Enterprise Solr.
* Data Storage - If necessary the raw data could be stored in Cassandra, a proven and scalable NoSQL database.

With DataStax Enterprise 2.0, the custom developed and complex HBase Queue Table could be entirely replaced. The durability was supported by Cassandra internals and there was no longer any need to manually manage shards. The query component of the REST API did not need to change as all of the Solr APIs were completely supported. It was for the most part as simple as removing some code, installing/configuring DataStax, and pointing the REST API to a new Solr host. The final architecture is depicted below.

Performance Tuning

Search and Real Time Analytics on Big Data

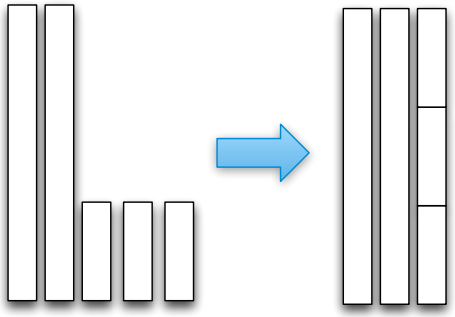
Near Real Time Search

- Hard Commit: Performs fsync to disk
  - Slower availability in query
  - Greater reliability if node goes down
- Soft Commit: Does not fsync, straight to memory
  - Near real time indexing
  - Only reliability up to latest hard commit
- For real time use cases, we usually soft commit frequently (sub second) and hard commit every few minutes.

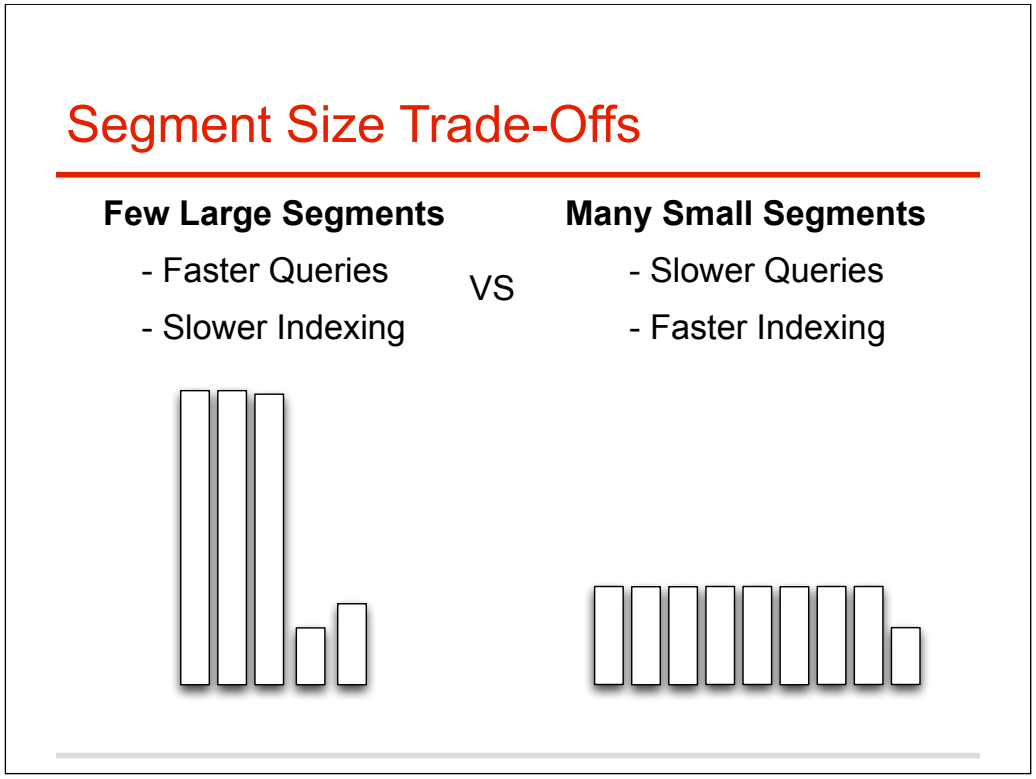
Jason

Remember Segments?

- Log structured merge tree
- Written once and immutable
- Segments merge as the index grows



Jason



Jason

You can set the Lucene segment merge ratio in the SolrConfig file
Few larger segments mean you dont have to query as many segments, but you will be constantly merging
Many small segments mean you will not have to merge as often, but your queries will have to iterate over many segments

Optimizing

- Merges all of the Lucene indexes to one segment
- Rewrites the entire index, careful!
- Let Lucene handle segment merging instead

Jason

System IO Cache

- Most Lucene operations rely on the operating system IO cache to keep the index effectively ‘in-ram’
- Lucene relies on fast random access which ram provides

Jason

Conclusions

Search and Real Time Analytics on Big Data

Jason

Solr and Lucene

- Advanced text search and more
- Real time analytics
- Rich SQL like functionality
- Excellent as a secondary indexing system
- Ability to scale
- Open source
- Integration with NoSQL is already happening

Jason

Questions?

Search and Real Time Analytics on Big Data

Jason

Thank You!

Search and Real Time Analytics on Big Data

ryan.tabora@thinkbiganalytics.com
jason.rutherglen@datastax.com