Solr Primer
Day 1

Discount
Tires

jeffnb@gmail.com

Course Outline [Day 1]

- Introduction to Solr
- Solr Overview
- Solr Admin
- Searching
- Schema

Course Outline [Day 2]

- Indexing
- Configuration
- SolrJ
- Optimization
- SolrCloud

Part 0: Solr

Solr: The Database Issue

- Filtering simple things

```
SELECT * FROM movies WHERE Year = 1995;
```

- A little more complicated looking in multiple fields

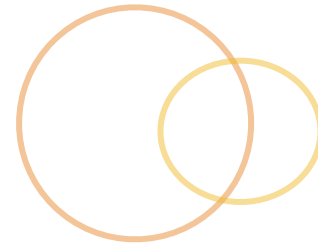
```
SELECT * FROM movies WHERE Title like '%Toy%'
   OR Plot like '%Toy%';
```

- Even more so if we need to check many fields

```
SELECT * FROM movies WHERE Title like '%Tom%'
   OR Plot like '%Tom%' OR Actors like '%Tom%'
   OR Director like '%Tom%';
```

Solr: The Database Issue

- Text field searching is slow
- No “best” matches
- “Try” != “Tries”




- What?
  - Enterprise Level Search Engine
- Why?
  - Extremely fast
  - Extremely flexible
  - Extremely powerful
  - Extremely scalable

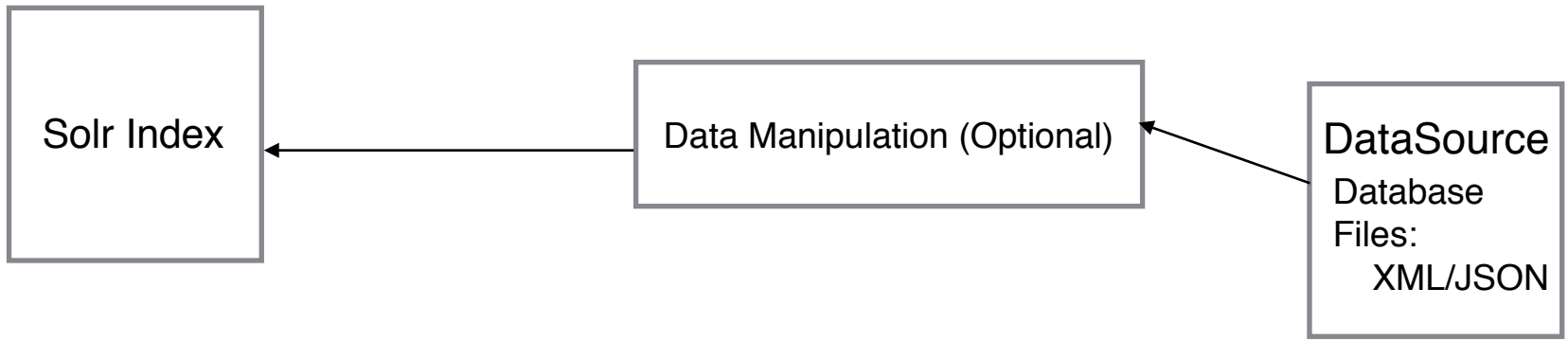
Solr: What Can it Do

- Searching
- Faceting
- Tokenizing
- Spell Checking
- Replicating
- Similarity-ing

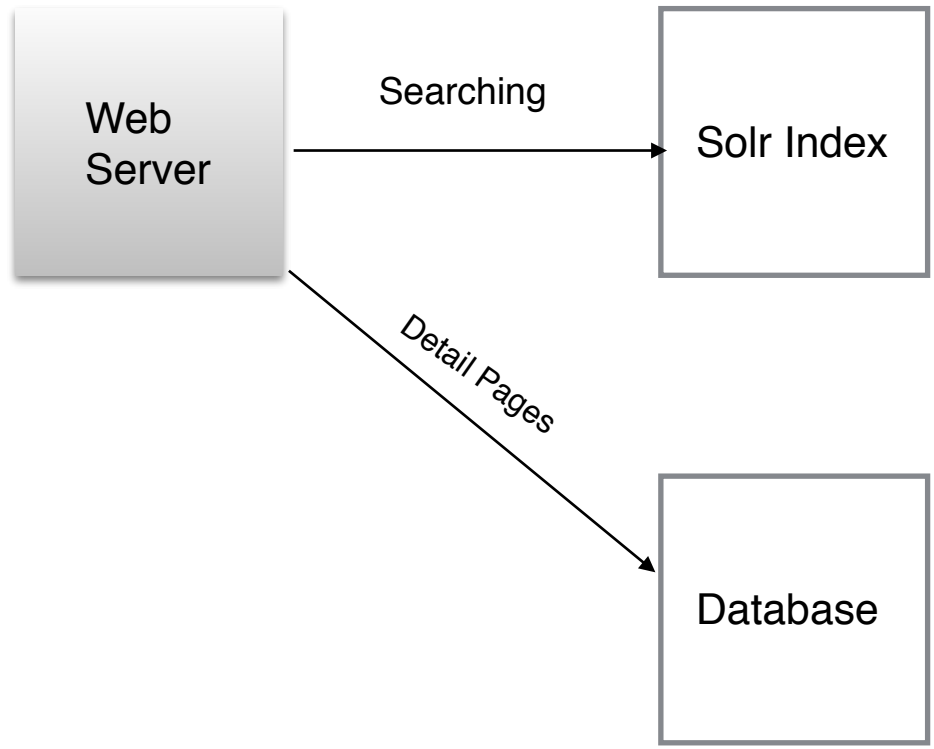
Solr: History

- Built on Lucene
- Versions up with Lucene

Solr: Indexing



Solr: Searching



Solr: Master/Slave

- Main server responsible for indexing
- Slave servers
- Replication handled within Solr Software

SolrCloud

- Built to simplify scalability
- Allows adding nodes to a cluster

Solr Admin

- Walkthrough

Part 1: Searching

Searching: The Basics

- q=<field>:<value>
- Wildcards: * and ?
- Fuzzy searches: toy~ or toy~1
- Proximity: “toy story”~10
- Required: +toy
- Ranges:
  - Year:[1995 TO 1999] - Inclusive
  - Title:{Aladdin TO Boy} - Exclusive
- Boosting: ^10

Lab: Searching

- Try to find the movies with Woody Harrelson
- Find your favorite movie
- Get movies that were made in 2010 and 2011
- Return movies that were about disasters in 1999
- Find movies that have the word “world” and may have “danger”
- Find movies with a fuzzy search for fight
- Find movies with Deep in the title or the Plot
- Same as previous but boost documents with it in the title by 10

Searching: Common Params

- sort DESC:ASC
- Field List: `fl=title,plot,year`
- Filter Query:
  - Applies filter without impacting the score
  - `fq=Year:1995&fq=actors:"Tom Hanks"`
- Write Type: `wt=json`
- Indent to pretty up the return: `indent=true`
- Start (offset): `start=10`
- Debug/ExplainOther

TF/IDF

- Term Frequency
  - Search: The Brown Cow
  - Documents more relevant where “the”, “brown” and “cow” occur the most
- Inverse Document Frequency
  - Adds weight to terms that are infrequent across all documents
  - “The” occurs many times across all documents
  - “brown” and “cow” get more weighting

Lab: Common Parameters

- Return json and indent it
- Get a list of movies with Title, Year, score and imdbRating for Adventure and show 50 records
- Sort the previous example on the highest rated and most recent
- Search for “batman” in plot and title but use fq to filter to the year 2013
- Use `debug=true` to parse through the matching of the previous example
- Play around with searching and see what else you can find

Solr: Facets

- Similar to group by
- Allows for facetted navigation
- Field must be indexed
- Facets on tokens not stored data
- `facet=true` Turns on the facet system
- `facet.field` Picks a field to facet on
- `facet.limit` Sets a max amount of facet values
- `facet.mincount` Sets minimum doc count for a facet

Solr: Facets

- `facet.prefix` Simply filters facets to prefix
- `facet.contains` (contains.ignorecase): Filters facets to ones that contain the string
- `facet.offset`: Starts the facets after the offset
- `facet.range`: Which field to do a range facet
- `facet.range.start (end)`: Start and end of range
- `facet.range.gap`: The amount of each range
- All properties can be set on a per field basis with `f.<fieldname>.<parameter>`

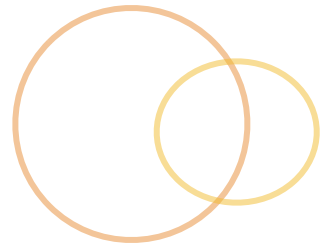
Lab: Facets

- Facet on Year and genres
- Facet on actors but limit to only 5 facets
- Facet on Year. Limit them to years with more than 1000 movies in the data set
- Facet on all Baldwin brothers using contains
- Facet on Year in ranges of 5 years miscount of 1

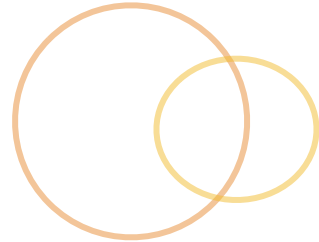
Discussion: Searching

- How do you want to search?
- What bits of information do you need?
- What ways can you incorporate searching into your business?

Part 2: Schemas and Indexes



Fields



- Fields store data
- Fields have a type
- Parameters
  - stored: values can be retrieved
  - multiValued: Can store multiple values
  - indexed: Can be searched/faceted
  - required: Document will error if value missing
  - type: The type of field
  - docValues: Orders values to be more efficient

Basic FieldTypes

- Data types
- No tokenization
- string: most common
- int: solr.TrieIntField
- long: solr.TrieLongField
- float: solr.TrieFloatField
- double: solr.TrieDoubleField
- date: solr.TrieDateField

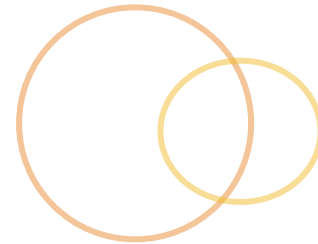
Lab: Basic Fields

- `bin/solr create_core -c movies -d <path>/movies-start`
- Open file: `server/solr/movies/conf/managed_schema`
- Find the id field in the file and add fields under. Add remaining fields with built in types from the following: Country, Rated, Language, imdbVotes, Type, Poster, Metascore, Year, actors, genres, directors, writers, Runtime, imdbID, Released, imdbRating

Text FieldTypes

- The magic is here
- Many pre configured including text_general
- Have several steps in processing data
- Are the core power of solr

Analyzers



- 2 Times Run
  - Index: Documents going in
  - Query: Searching existing
- Contain steps to process data
- First Step Tokenizer

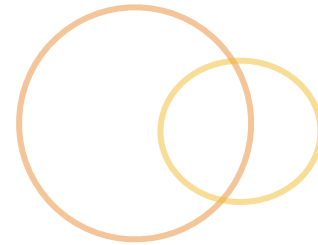
Tokenizers

- Tokenizers take input and break it into “tokens”
  - The Quick Brown Fox Jumps Over the Lazy Dog
  - “The”, “Quick”, “Brown”, “Fox”, “Jumps”, “Over”, “the”, “Lazy”, “Dog”
- Can also produce 1 token

Tokenizer Overview

- Standard: Splits based on whitespace and punctuation
- Classic: Standard but doesn't support Unicode Annex
- Keyword: Keeps entire input as a single token
- Letter: Tokens are strings of contiguous letters
- LowerCase: Delimits on non-letters and lowercases
- N-Gram: generates n-gram tokens
- Path Hierarchy: Replaces a delimiter with another value building up a path
- Regular expression: Tokenizes based on regular expression as a delimiter
- Url Email: Splits on white space and tokens preserving email and urls
- Whitespace: Splits on whitespace only

Filters



- Alter tokens
- Produce more tokens
- Suppress tokens
- Dozens exist

Filters: Heavily Used

- Lowercase Filter: Lowercases all tokens
- Stop Filter: Takes a file and removes all tokens that match words in the file
- Stemmers (porter/snowball): Remove word conjunctions and plurization
- Synonym Filter: Token matched to file and all synonyms added as tokens
- Trim Filter: Simply trims white space
- Pattern Replace Filter: Replaces a regular expression match with a string

Lab: Text Fields

- Create a field for Plot and Title
- Create a field type for Awards
- Create a field for awards using your own field type
- Use the analyzer admin to ensure these fields work

CopyField

- Shortcut to copy one field into another

```
<copyField source="<sourcefield>" dest="<destfield>"/>
```

- Useful for when facets and search are needed

Lab: CopyField

- Use copyField to copy the following to text:
  - actors, genres, directors, writers, Plot, Year, Title, Awards, language, Rated
- Create fields for (use a name like actors_text):
  - actors
  - genres
  - directors
  - writers
- Use copy field to copy the source fields above to the new _text versions

Dynamic Fields

- DynamicField allows wild carding field names

```
<dynamicField name="*_en"  type="text_en"
indexed="true"  stored="true" multiValued="true"/>
```

- Any field with _en at the end automatically populates into that field
- Can be referenced directly

Lab: DynamicFields

- Change all _text fields to simply copy into _en fields
- Explore a few of the other dynamic fields

Discussion

- Pick a feature Content, Product, Reviews
- What are major fields needed?
- What types are the fields?
- What analysis could be used?

Schema API

- Allows for schema changes dynamically
- Allows for adding, changing, deleting fields and field types
- Caveat: Changing a field does not change data
- Main entry point:
  - `http://<host:port>/solr/<core>/schema`
- Reading fields or a specific field:
  - `/solr/<core>/schema/fields/`
  - `/solr/<core>/schema/fields/<fieldname>`

Lab: Schema API

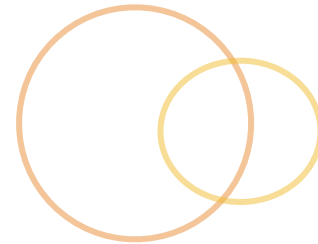
- Retrieve the list of the defined fields for the index
- Retrieve the Plot field specifically
- Add a new field RTReview (int) for storing rotten tomatoes
- Add a string field to store tags
- Add a field type to search tags should trim, lowercase, and stem them
- Create a copy directive to copy the first tags field into the new field type

Schemaless

- Solr can run in schema less (guessing) mode
- Data will determine the schema
- New fields are added automatically

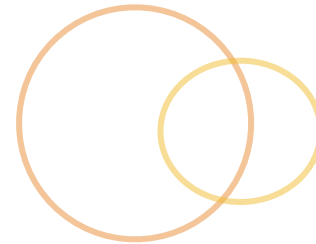
Part 3: Indexing

Indexing



- Process to add to the index
- Common ways:
  - `bin/post` file
  - Post request to update handler
  - Data Import Handler

Deleting



- Leaves document in index hidden
- Posting with body
  - {"delete": { "id":"12345" }}
  - {"delete": { “query”:”Title:Toy Story" }}

Committing

- Adding records keeps them in staging
- Commit makes the records update the index
- Similar to database transactions
- bin/post commits automatically
- Sending a commit:
  - http://localhost:8983/solr/movies/update?commit=true

Commit: Soft vs Hard

- Hard commits write to disk
- Soft commits:
  - Makes changes visible quicker
  - Changes not written
  - Crashes can lose data

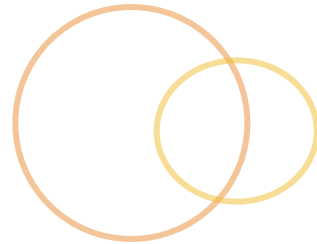
Optimizing

- Optimization cleans up the index
- Reduces amount of index files
- Commit does not remove deleted docs
- Optimize makes the index nice and clean
- Can speed up search times

Lab: Load the Data

- Moment of Truth
  - bin/post -c movies <path>/movie_data_cleaned.json
- Fix errors or ask questions
- Delete your least favorite movie
- Run an optimization if num docs and max docs

Part 4: Configurations



SolrConfig.xml

- Defines:
  - All Handlers
  - Commit Behavior
  - Caching
  - Default behaviors
  - Search Components installed
  - Schemaless configuration
  - Default and environmental variables
  - Listeners

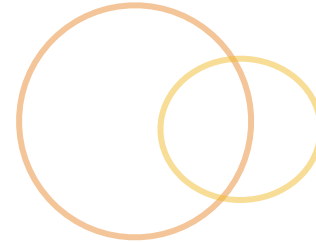
AutoCommit

- Allows commits to be run automatically
- Useful when trickle changes come in
- Helps avoid many small commits

AutoSoftCommit

- Commits will be visible
- Commits not written to disk
- Useful with AutoCommit
  - AutoCommit at longer intervals
  - AutoSoftCommit at short intervals

Caches



- Query
  - Stores document ids for common queries
  - Cuts down on multiple index calls to find docs
- Filter
  - Stores document ids for common filter queries
  - Speeds up common filters considerably
- Document
  - Stores commonly requested fields
  - Helps speed up document retrieval

Searchers

- Similar to a thread
- Searchers process the request
- Searchers have their own caches
- Have events:
  - Trigger when the first searcher starts
  - Trigger when new searchers start
  - Useful for warming with common queries

Lab: Commits and Caches

- Set up auto commit in the movies solrconfig file
  - Commit every 5 minutes
  - Soft commit every minute
  - Ensure new searcher is opened on commits
- Experiment tuning the caches
  - Enable a filter cache which auto warms with 200 entries
  - Enable a query cache with 20 entries auto warmed
  - Look at the admin Plugins->Cache and checkout the cache levels. Try queries and watch them change
- Run 3 queries on startup when searcher is warmed

Search Handlers

- End points for querying
- Several come with solrconfig
- Built in ones can be overridden
- New ones can be created (edismax from earlier)
- Specifies all behaviors
- `class="solr.SearchHandler"`

Search Handlers

- Defaults for queries
  - `<lst name="defaults">`
  - Specifies any query defaults
  - Useful for fl, rows and wt
- Appends
  - `<lst name=“appends">`
  - Allows values to queries
  - User cannot override
  - Example: `<str name="fq">inStock:true</str>`

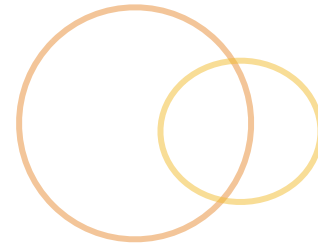
Search Handlers [cont]

- Invariants
  - `<lst name="invariants">`
  - Used to limit options on queries
  - Example: `<str name="facet.field">genres</str>`
  - Once specified changes are allowed query time for that name

Lab: Search Handler

- Create a new search handler for Action movies
- Specify these defaults:
  - fl with Title, Plot, Year, genres, imdbRating
  - rows of 50
  - wt of json
  - turn on indent
- Append genres_en:Action to all queries
- Allow faceting on Year and actors

eDismax



- Advantages
  - Intended to not throw errors
  - Syntax more forgiving
  - Can search over many fields
  - Useful for user searches
- Extremely powerful
- Flexible for different needs
- Run with: `defType=edismax`

eDismax parameters

- q - Term to search for
- qf
  - Fields to use to search with boosts

```
<str name="qf">
   Title^10 Plot^5
</str>
```

- mm - Minimum amount of terms to match
- pf
  - Boosts based on fields whole phrases appear in.
  - Same syntax as qf
- bq
  - Boosts based on a query
  - bq=genres:action^3.0

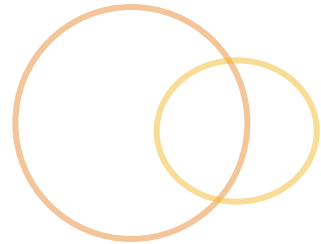
Lab: eDismax Handler

- Create a new search handler “edismax”
- For search fields:
  - Title and Plot are the highest boosted
  - actors, directors and writers have a lower boost
  - Year and Language are searched but not boosted
- All terms should match
- Boost (pf) Title and plot again
- Rows should be 30
- Fields returned should be Title, Plot and Year

Discussion

- Now that you have a grasp on query options
- What fields do you think are most important?
- What would you want to return by default?

Part 4: Optimization



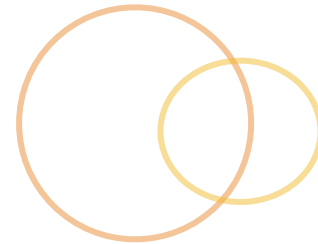
Overview

- Solr is big
- No silver bullet
- Should be based on user behavior
- Dependent on your architecture

System Level

- Memory
  - Do not give Solr all the memory
  - At least half should be left for the OS
  - Turn up OS file cache to get index in memory
  - Make sure open file limit is high enough
- Hardware
  - Master/Slave servers work much better on bare metal
  - SSDs are great speed improvements if possible
  - Ram can boost performance

Caching

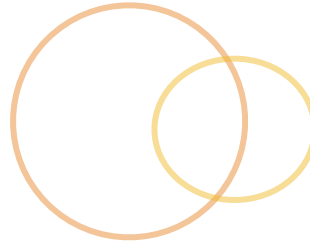


- Look at the logs
- Warm searchers with common queries
- Autowarm when you can
- Start with defaults and increase or decrease levels
- Balance memory with caches
- Use the cache admin to get hit rates

Distributed Indexes

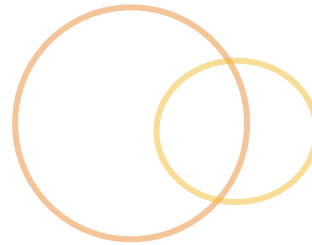
- Indexes slow with size
- Threshold highly variable (millions of records)
- Sharding: breaking up index
- Solr HIGHLY recommends not sharding with M/S setup
- If index becomes too large move to SolrCloud

Commits



- Commits slow throughput
- Searches are expensive to open
- Be practical about real time data needs
- Use soft commits to supplement hard commits
- Transaction logs will grow out of control without hard commits

Optimizes



- Slow throughput down
- Optimizing creates new files
- Optimizing merges files making index smaller
- Overall better performance when optimized
- Better run at low times

Replication

- Replication can cause new searchers
- Will temporarily slow searches down
- Understand real time needs

Garbage Collection

- Larger Solr instances
- Can stop the entire server for seconds
- Older versions of GC “stop the world”
- CMS better for 1.6
- G1 better

Part 5: SolrCloud

SolrCloud Overview

- Grew out of scalability troubles
- Closest to ElasticSearch methods
- Handles replication and sharding
- Uses ZooKeeper to manage nodes
- Allows new servers to plug in with ease
- Has collections not cores

Lab: SolrCloud

- Use bin/solr to create a movies collection
- Use 2 shards and 2 replicas
- Import movies data
- Explore diagrams in admin interface

Part 6: SolrJ

SolrJ Overview

- Included with solr
- Abstracts many of the communication details
- Adding documents easier with java based class
- Uses a fast java serialization
- Highly recommended for all java projects

Lab: SolrJ

- Use solrj-demo project
- Connect to your running solr instance
- Run a query to get all Avenger movies
- Delete a movie you hate by id
- Add a movie you enjoy from the last two years
- Run a commit to get the new movie to show

Thank you

Please fill out the surveys