Project Report

CS5604: Information Storage and Retrieval

Offered by Dr. Edward A. Fox
Spring 2015

Project IDEAL (Integrated Digital Event Archiving and Library)
Solr Team

Richard Gruss, Ananya Choudhury, Nikhil Komawar
{rgruss, ananya, komawar}@vt.edu
Department of Computer Science
Virginia Tech, Blacksburg
May 11, 2015

Abstract

The Integrated Digital Event Archive and Library (IDEAL) is a Digital Library project that aims to collect, index, archive and provide access to digital contents related to important events, including disasters, man-made or natural. It extracts event data mostly from social media sites such as Twitter and crawls related web. However, the volume of information currently on the web on any event is enormous and highly noisy, making it extremely difficult to get all specific information. The objective of this course is to build a state-of-the-art information retrieval system in support of the IDEAL project. The class was divided into eight teams, each team being assigned a part of the project that when successfully implemented will enhance the IDEAL project's functionality. The final product, which will be the culmination of these 8 teams' efforts, is a fast and efficient search engine for events occurring around the world.

This report describes the work completed by the Solr team as a contribution towards searching and retrieving the tweets and web pages archived by IDEAL. If we can visualize the class project as a tree structure, then Solr is the root of the tree, which builds on all other team's efforts. Hence we actively interacted with all other teams to come up with a generic schema for the documents and their corresponding metadata to be indexed by Solr. As Solr interacts with HDFS via HBase where the data is stored, we also defined an HBase schema and configured the Lily Indexer to set up a fast communication between HBase and Solr.

We batch-indexed 8.5 million of the 84 million tweets from HBase before encountering memory limitations on the single-node Solr installation. Focusing our efforts therefore on building a search experience around the small collections, we created a 3.4-million tweet collection and a 12,000-webpage collection. Our custom search, which leverages the differential field weights in Solr's edismax Query Parser and two custom Query Components, achieved precision levels in excess of 90%.

1)

Table of Contents

Table of Figures

Tables and Listings

1 Overview

1.1 Motivation

The Integrated Digital Event Archive and Library (IDEAL) project, an extension of an earlier project, CTRnet, collects, analyses and visualizes information pertaining to events with a significant human impact such as community activities, government activities, natural disasters, disease outbreaks, terrorist attacks, etc. The IDEAL project focuses on crawling the web for documents related to these events and provides analysis, visualization, archiving, and retrieval services. To date, the project has collected over 1 billion tweets and over ten terabytes of web pages. The information buried inside these tweets and web pages is crucial for event analysis and related research. But manual retrieval of such contents is extremely tedious, time consuming and error prone. Hence we need to develop a domain-specific automated search system that provides fast and accurate information storage and retrieval.

1.2 Management

The 2015 class of CS5604 was divided into teams that specialized in certain areas in the Information Storage and Retrieval domain. We were responsible for understanding and advising on all the tools and solutions around Solr. None of the teams have been dependent on the output of Solr. We worked closely with the Hadoop team, which focused on appropriate HBase table input and output implementation. As Solr was directly indexing from the HBase tables that held the analysis metadata from the other teams, we communicated with them to provide them with requirements of Solr indexing and advised them on inserting data into HBase.

The intra- and inter-team communication was mostly done via emails sent to the individuals that were working on specific issues. Demos and presentations were given in some of the classes. Other specifics of the project development were mentioned in the periodic reports as per the specifications given by the instructor. A github repository was set, however its adoption by the class was minimal. Reports, Google Docs, Piazza posts, and personal interactions were the main modes of communication with other teams for finalizing the Solr schema design. The HBase schema was finalized by consulting mainly with the Hadoop team.

The overall communication and management of the project in the class was thorough and close nit. The issues were resolved amicably, teams coordinated well to come to a common understanding, and solutions were developed by considering all the trade-offs.

1.3 Challenged Faced

During the implementation of this project, we encountered several challenges. We found solutions for some, mitigated a few issues and, unfortunately, due to dearth of necessary resources, we had to limit our progress because of one issue. Listed below are some significant hindrances we faced.

- Configuration issues: Like most of the technical documentation available on Solr, that for Lily and HBase are not up-to-date. We spent hours doing trial and error to configure and tweak these frameworks.

- Defining the schema for both HBase and Solr: We defined a draft of the schemas very early in the project. But creating a consolidated schema for Solr and then HBase required that several teams identify their respective final outputs. This required a significant number of iterations and much time.

- SolrCore or SolrCloud: Up until recently, we worked on the Cloudera VM in our local machines with the expectation of setting up the same configuration on the cluster. Due to a security issue revealed towards the end of the semester, we had to operate on the cluster with minimal control. The GTA helped us configure the cluster according to the restrictions. The apprehension of security loopholes as well as lack of documentation on the side effect of certain configurations significantly increased our communication barrier with the GTA and slowed our progress.

- OutOfMemoryError: We successfully indexed all of the 3.4 million tweets from the small collections and are being capped at 8 million tweets from the large collection due to resource constraints. The index for the large collection of 85 million tweets proved to be too immense for our Solr configuration, and we repeatedly encountered a "java.lang.OutOfMemoryError: GC overhead limit exceeded" error. We tried doubling the memory allocated to the Solr Java Virtual Machine from 1GB to 2GB, but this did not solve the problem. We also tried tightening up the index by not storing fields unless absolutely necessary, but this also had no effect. This issue can be fixed by creating multiple index shards, which is only permissible when the number of Solr nodes is greater than or equal to the required number of shards.

1.4 Solutions Developed

- Designed schema.xml for types of content -- tweets and webpages.
- Collaborated with Hadoop team to design an HBase schema.
- Used Nutch to get large collections on HDFS.
- Designed "morphline" configurations for Lily HBase Indexer [27].
- Used Lily HBase Batch Indexer to index data into Solr.
- Worked with GTA to fine tune the cluster to accommodate the size of the index.
- Developed a custom search handler.
- Configured Solr to use multiple search handlers as per the requirements of other teams like Social Networks, LDA, etc.
- Did a performance analysis of indexing, querying and read/write operations for size of the data.
- Documented and analyzed the results.

1.5 Future Work

For our current single node Solr setup, we see memory issues while attempting to index large amounts of data. Currently, the design and the (single shard) Solr support data of the size of approximately 8-9 million tweets. However, the large collection that we were provided has 85 million tweets. We would need to index webpages for the large collections as well, that would have a separate index. Also, in the future more metadata would be added to each of these fields. Hence, the index size is likely to grow over time. The first solution to consider for the future would be running a SolrCloud (multiple shards) over at least a dozen nodes.

The anticipation is that those may or may not be sufficient depending on the hardware support and various configurations. We need to consider better strategies to reduce the size of our indexes even for being able to handle one set of tweets and webpages with batch processing. Moreover, batch processing can be slow while running multiple shards and may result in serious network bottlenecks (again depending on the network capacity). If we wish to support dynamic updates that include index updates and non-null metadata in the analysis fields, a good approach would be to come up with a strategy to reduce the number of fields and the number of stored fields in the Solr schema (schema.xml for both tweets and webpages). String fields being more space-efficient than text fields [28], we should consider reducing text fields in our schema. Also, Solr comes with a lot of dynamic fields out of the box that may or may not be necessary. Some discussion on the ML [31] suggests that we need not have out of the box dynamic fields like "*_t_raw", "*_fs_raw", etc. As a part of the future work we may try removing them from Solr schema to see the performance gain in indexing and ensure that query performance is not degraded.

A random string prepended with a collection identifier provides a good compromise between the convenience of a random string, which can be generated asynchronously, with the utility of a natural key such as a collection name, which provides an additional human readable field for understanding a document Our document identifiers, therefore, look like "collection_UUID" (e.g., ebola_S—307834)

2 Literature Review

Since our team was primarily responsible for technical expertise in Solr and Lucene, we agreed that certain technology-specific external sources—Manning's *Solr in Action* and *Lucene in Action*—would provide the core of our reading. The textbook 'Introduction to Information Retrieval' [1] supplied the theoretical background on concepts related to indexing, similarity measures, and document ranking.

Basic text processing techniques such as tokenization, stopword removal, etc. must be applied before any raw text can be indexed. Section 2.2.1 in [1] discusses tokenization with several examples. Section 2.2.2 discusses briefly about stopwords. Although a large section of the book deals with index construction, Chapter 4 introduces the topic and discusses the concept of "inverted index". Section 4.4 discusses distributed indexing algorithms commonly used in web search engines. Chapters 6, 7 and 11 cover ranking functions and similarity measures. Parametric and zone indexes which allow us to retrieve documents by metadata are introduced in Section 6.1. Section 6.2 develops the idea of weighting the importance of terms in a document represented by the Vector Space Model, discussed in Section 6.3. This is an important topic from the perspective of Solr, where we can boost relevance of certain terms

using weights. Section 6.4 introduces the famous tf-idf weighting function. Section 7.1 describes a ranking algorithm, which is further illustrated in Section 11.2. It is essential that a search engine identifies the most relevant documents and rank-orders based on matching the query. Improving search engine recall by addressing issues like synonymy has been described in Chapter 9 along with query expansion under the topics 'relevance feedback' and 'query extension'.

Solr in Action [2] provides the knowledge and techniques necessary to get Solr up and running. It also gives information about the underlying Solr architecture and covers key concepts through several out-of-the-box features. We are also referring to the "Solr Reference Guide" [3] that comes with the bundle and the Solr wiki [4] to get the latest information on Solr configuration and features.

Lucene is the underlying Java search library for Solr. *Lucene in Action* [5] covers the overall architecture of Lucene and how it can be manipulated to get customized features using Solr.

Apache Mahout is a free implementation of machine learning primarily in the areas of clustering and classification. We might have to refer to Mahout resources if teams such as the Classification team, Clustering team, etc. plan to use it. Getting some knowledge on the framework will help us integrate Mahout with Solr/Lucene. We plan to use the book 'Mahout in Action' [12]. As we decided to use HBase on top of HDFS to interact with Solr and integrate HBase and Solr through the Lily Indexer, we referred to [18] for configuring Lily and [19] to work on HBase.

Besides all this, we referred to a few external resources to gain some more insight into the domain. [6] discusses the visualization aspects of search features like faceted search and hit highlighting. [9] introduces Earlybird, a core retrieval engine behind Twitter's search service. This is powered by the Lucene NRT search, but modified to accommodate Twitter's use case of real-time search. To understand how Solr can be configured to deal with ambiguous words and if time permits, enhance context based search in Solr, refer to [11]. We have also added a few extra resources [7, 8, and 10] into our bibliography section related to Solr/Lucene, which can support future reference.

3 Requirements

The following sets of requirements were accomplished over the course of this semester:

- Build a generic Solr schema for two different types of data sets, viz. tweets and webpages.
- Collaborate with all the teams to devise an HBase schema that the Lily indexer can use to index, and one that is compatible with Solr schema.
- Fine tune schema.xml, solrconfig.xml, solr.xml to accommodate the requirements of the metadata from other teams, handle size of the data and keep them flexible to support future enhancements without having to redevelop from scratch.
- Integrate HBase with the Lily indexer and Lily indexer with Solr.
- Configure Lily Batch indexer [27] to fetch data from HBase and index into Solr.
- Design the deployment of Solr to match the needs for scaling the concerned Big Data.
- Index the data into Solr and collaborate with other teams to do knowledge sharing. This will enable them to index data into Solr as and when new analysis is performed.
- Develop a custom query processor for Solr.

- Configure Solr to do performance analysis using different processors including the custom one.

4 Design

4.1 Conceptual Background

Solr
Apache Solr [3] [4] is a well-known open source search platform for searching data stored in HDFS in Hadoop. It is written in Java and runs in a servlet container such as Jetty or Tomcat. It extends Apache Lucene Library project and uses Lucene as its core for full-text search and indexing.

Cloudera Search, which comes with SolrCloud, increases capabilities of Solr's distributed search. As shown in Figure 4.1, in SolrCloud data is organized into multiple pieces, or shards, that can be hosted on multiple machines, with replicas providing redundancy for both scalability and fault tolerance. A ZooKeeper server manages the overall structure so that both indexing and search requests can be routed properly.

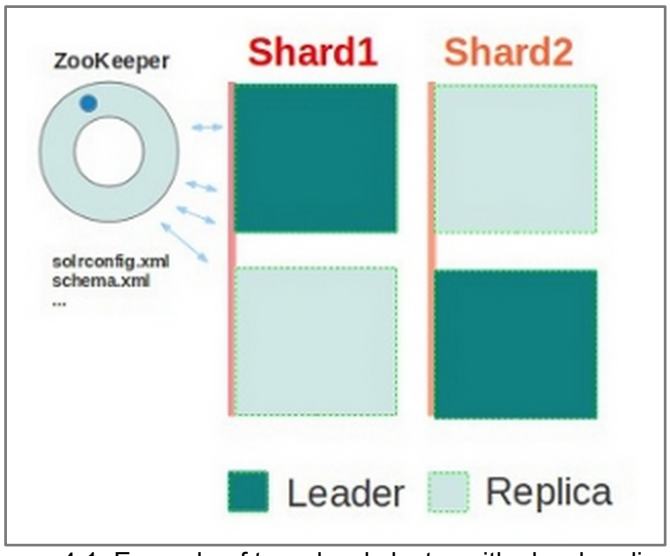

Figure 4.1: Example of two shard cluster with shard replicas.

Lucene
As we said earlier, Apache Lucene [5] is the core of Apache Solr. It is full-text index and search library that is capable of indexing every imaginable text file. When indexed, the textual information contained in the document can be extracted. It uses compressed bitsets to store an inverted index and supports binary operations such as AND, OR and XOR, which can be performed at lightning-fast speeds, even for billions of records.

HBase
HBase is a non-relational, column oriented, multi-dimensional distributed database with high performance. It is an open source implementation of Google's BigTable storage architecture. It can manage structured and semi-structured data and has some built-in features such as scalability, versioning, compression, and garbage collection. HBase is built on top of Hadoop / HDFS and the data stored in HBase can be manipulated using Hadoop's MapReduce capabilities.

The HBase Physical Architecture consists of servers in a Master-Slave relationship as shown in Figure 4.2. Typically, the HBase cluster has one Master node, called the HMaster, and multiple Region Servers called the HRegionServers. Each Region Server contains multiple Regions called HRegions.

Data in HBase is stored in tables and these tables are stored in Regions. When a table becomes too big, the Table is partitioned to span multiple Regions. The maximum Region size is a tuning parameter. These Regions are assigned to Region Servers across the cluster. Each Region Server hosts roughly the same number of Regions.

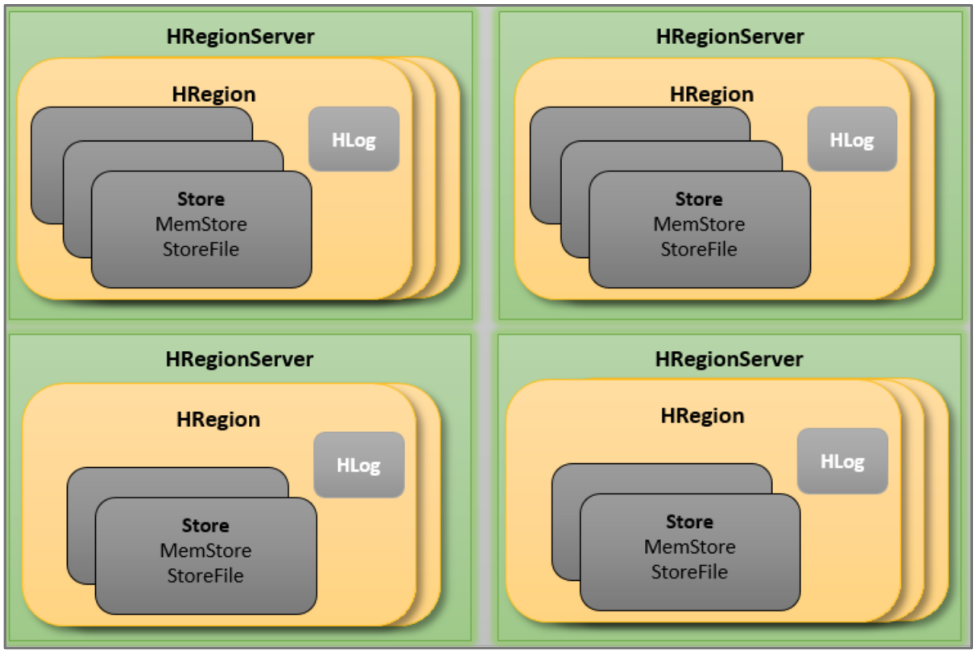

Figure 4.2: A typical HBase architecture[23]

Lily HBase Batch Indexer
The HBase Indexer provides indexing (via Solr) for content stored in HBase. Indexing is performed asynchronously, so it does not impact write throughput on HBase. SolrCloud is used for storing the actual index in order to ensure indexing scalability.

The diagram Figure 4.3 shows the connection between the main components of the Lily Indexer [20] and its connection between Solr and HBase. The Lily Repository manages a basic entity called a record. Fields in a record can be blobs. These blobs are stored either in HBase or on HDFS, depending on a size-based strategy.

The role of the Indexer is to keep the Solr-index up to date when records are created, updated or deleted. For this purpose, the Indexer listens to the HBase Side Effect Processor (SEP) events. The indexer maps Lily records onto Solr documents by deciding which records and what fields of the record need to be indexed.

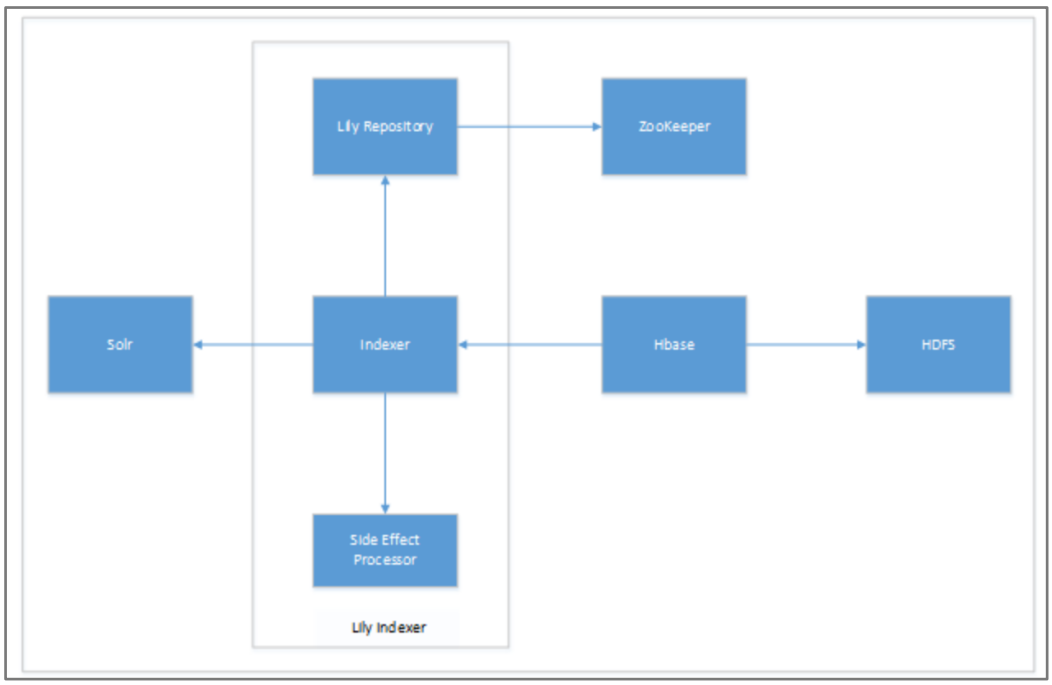

Figure 4.3: HBase-Lily-Solr Integration.

4.2 Tools

A number of tools were used in the project:

- Cloudera search
- Nutch
- Batch Lily indexer service
- Zookeeper
- HBase shell
- Linux bash utility
- Hadoop command line utility
- Eclipse IDE
- Github repo
- Cloudera Hue web UI
- Various dash boards on hadoop.dlib.vt.edu
- “vi” editor

4.3 Design/Approach

Once we settled on the HBase-> Solr architecture, we divided our team’s tasks into four phases:

Phase 1. Design two Solr schemas, one for tweets and one for webpages that will accommodate the analysis metadata of the other teams in a way that will allow for efficient retrieval.
Phase 2. Design an HBase schema that will lend itself to transferring rows to Solr.
Phase 3. Index the data from HBase into Solr. Repeat periodically.
Phase 4. Configure Solr and design custom Search Components that will leverage the metadata and provide a rich search experience.

4.4 Deliverables

Each of these phases had a distinct deliverable:

Phase 1. Two schemas, posted so that classmates can offer feedback.
Phase 2. Two HBase tables.
Phase 3. Tweet and webpage data loaded into Solr.
Phase 4. A solrconfig.xml configuration and whatever supporting code is necessary to customize Solr search.

5 Implementation

5.1 Timeline

The table below details our deliverables, job status, and timeline for the project.

#	Task	Timeline	Status	Allocated to
1	Find out what information each team needs to put up on Solr and come up with a generic schema	2/9 - 2/20	Completed. Had discussions with LDA, NER, ReducingNoise, SocialNetworks and Clustering teams.	Choudhury, Komawar and Gruss
2	Complete basic Solr configuration			
	Configure Schema.xml for basic settings	2/2 - 2/6	Completed.	Gruss, Komawar and Choudhury
	Configure solrconfig.xml accordingly	2/9 - 2/13	Completed	Gruss, Komawar and Choudhury
	Write Python script to upload documents to Solr	2/2 - 2/6	Completed. Uploaded to VTechworks	Gruss, Komawar and Choudhury
	Create a basic user guideline	2/2 - 2/13	Completed. Uploaded to VTechworks	Gruss,

	document			Komawar and Choudhury
	Set up SolrCore/SolrCloud on multiple machines.	1/26 - 1/30	Completed.	Gruss, Komawar and Choudhury
3	Configure Schema.xml for Twitter and Web Pages	2/2 - 3/29	Completed.	Gruss and Komawar
4	Configure solrconfig.xml accordingly	2/2 - 3/29	Completed.	Choudhury
5	Install Cloudera search VM and upload tweets and web pages	2/16 - 3/20	Completed.	Choudhury, Gruss and Komawar
6	Crawl web pages from small and large collection using Apache Nutch.	2/17 - 3/17	Completed. Small collection and Large Collection has been crawled and the data has been uploaded to HDFS.	Choudhury, Gruss and Komawar
7	Compute how information from different teams will be added to Solr indexes	4/13 - 4/29	Completed.	Choudhury, Gruss and Komawar
8	Develop a rationale to use appropriate ID (added to the entity)	3/17 - 3/27	Completed.	Choudhury, Gruss and Komawar
9	Analyze the data as per instructions from all machine learning teams.	3/24 - 4/24	Completed.	Choudhury, Gruss and Komawar
10	Propose a (interim) HBase schema to be used by the class.	3/9 - 3/31	Completed.	Choudhury, Gruss and Komawar
11	Configure Lily Indexer with Solr and HBase	3/23 - 4/15	Completed.	Choudhury, Gruss and Komawar
12	Setup SolrCloud on the cluster	4/6 - 4/20	Completed. We are using only one node on the Custer as per suggestion from the TA.	Choudhury, Gruss and Komawar
13	Make a custom query processor in Solr	3/30 - 4/10	Completed. A search handler has been installed in the cluster.	Choudhury, Gruss and Komawar
14	Upload all data to Solr Cloud.	4/23 - 4/27	Completed (partially). Up to 8.5 million tweets indexed. Solr throws outOfMemory error for large collection.	Choudhury, Gruss and Komawar
15	Setup Carrot2 with Solr and Cloudera search.	Started: 4/3	Attempted integration permutations with older versions of Solr and Carrot2. Worked on Solr5. Out of time to setup on	Choudhury, Gruss and Komawar

			main cluster.	
16	Create custom search to use metadata	Started 4/29, Completed 5/4	Completed	Gruss

Table 5.1: Implementation timeline of Solr team

5.2 Evaluation

Phases 1-3 (schemas, HBase, and indexing): The success of the Solr schemas, HBase design, and Lily indexing is ultimately in the successful creation of a searchable Solr collection. The schemas were revised throughout the semester as teams adjusted their requirements, and we finally settled on the schemas listed in Appendix A and attached to this report. Figure 5.1 shows the timings of two indexing jobs. The top two rows represent the two stages of the Lily Indexer's MapReduce job indexing the large tweet collection. These two stages, which completed successfully in approximately three hours, generated 36 index shards in a temporary directory in HDFS. The concluding "Go Live" stage, however, in which the 36 shards are merged together into a single Solr index, failed due to memory limitations. The bottom two rows show the indexing of the small tweet collection, which successfully completed in about 20 minutes.

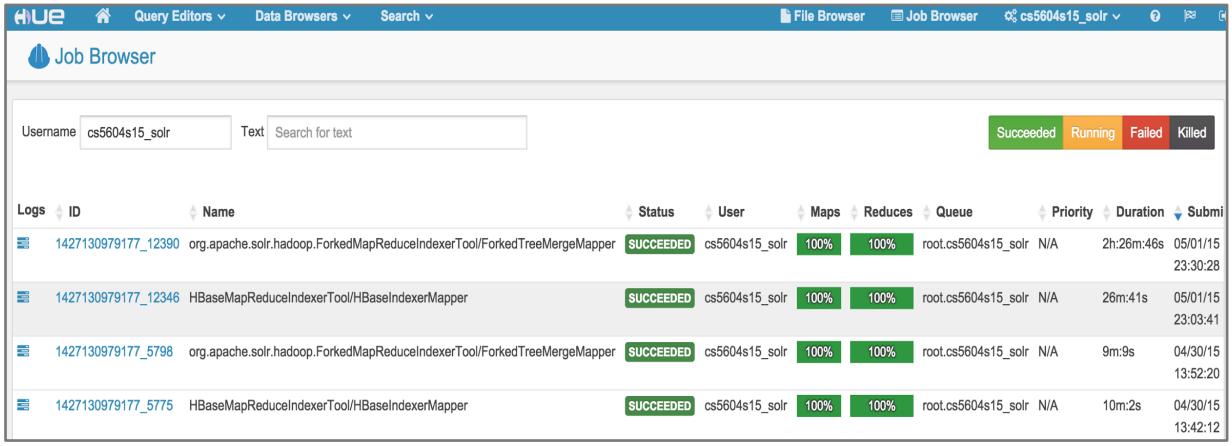

Figure 5.1: Four stages of two separate Lily indexing jobs.

Stage 4 (Solr search): The solr_test.py Python script attached to this report posts queries to our Solr server and computes the precision of the first 1000 results. Our proxy for relevance was the expected source collection of relevant documents. For example, we expected a search for "disease" to retrieve documents from the "ebola" collection, so any documents from that collection were deemed relevant. Precision, so defined, was generally extremely high (.7 and higher) as shown in Table 5.2. The number of documents retrieved ranged from 1143 to 637,498, and times were all less than one second. Solr is able to achieve these surprising speeds by keeping much of its index in memory or OS local caches.

{'query': 'election', 'num_results': 637498, 'precision': 0.998, 'time': 0.05295705795288086}
{'query': 'elect', 'num_results': 3247, 'precision': 0.978, 'time': 0.04558682441711426}
{'query': 'revolution', 'num_results': 13048, 'precision': 0.95, 'time': 0.04502081871032715}
{'query': 'uprising', 'num_results': 1769, 'precision': 0.851, 'time': 0.04298877716064453}
{'query': 'storm', 'num_results': 429329, 'precision': 0.999, 'time': 0.04975700378417969}
{'query': 'winter', 'num_results': 409987, 'precision': 0.999, 'time': 0.04920697212219238}
{'query': 'ebola', 'num_results': 306827, 'precision': 1.0, 'time': 0.04514813423156738}
{'query': 'disease', 'num_results': 6802, 'precision': 0.993, 'time': 0.041940927505493164}
{'query': 'bomb', 'num_results': 33924, 'precision': 0.857, 'time': 0.040463924407958984}
{'query': 'explosion', 'num_results': 1224, 'precision': 0.284, 'time': 0.04803609848022461}
{'query': 'crash', 'num_results': 274014, 'precision': 0.995, 'time': 0.04688715934753418}
{'query': 'plane crash', 'num_results': 193046, 'precision': 1.0, 'time': 0.056591033935546875}
{'query': 'shooting', 'num_results': 5366, 'precision': 0.744, 'time': 0.04262495040893555}
{'query': 'paris shooting', 'num_results': 446, 'precision': 0.446, 'time': 0.20793604850769043}
{'query': 'terrorist attack', 'num_results': 1143, 'precision': 0.768, 'time': 0.042675018310546875}

Table 5.2: Listing. Query results on the small collections in the cluster

6 User Manual

The IDEAL Solr application provides a customized search experience that leverages the analysis data contributed by the teams in the Information Retrieval class: NER, LDA, Social Networks, Clustering, and Classification. Document counts numbered in the millions, so it was essential that we provide a powerful and scalable solution for finding relevant information.

The IDEAL documents were split into two separate collections: tweets (3.4 million) and webpages (12,326). Table 5.3, below, summarizes the number of documents by collection.

Collection	**Tweets**	**Webpages**
Jan.25_S	495963	
charlie_hebdo_S	173847	159
ebola_S	381049	323
election_S	830282	120
plane_crash_S	266531	
suicide_bomb_attack_S	37823	
winter_storm_S	486573	783
Malaysia_Airlines_B	775565	
egypt_B	647	2146
shooting_B		8795

Table 5.3. Document counts by collection.

Although a cross-collection browse function is provided by Solr when velocity is enabled (see Figure 6.1), it is not currently available on the cluster for our collections, so you'll have to query the administrative console through port forwarding.

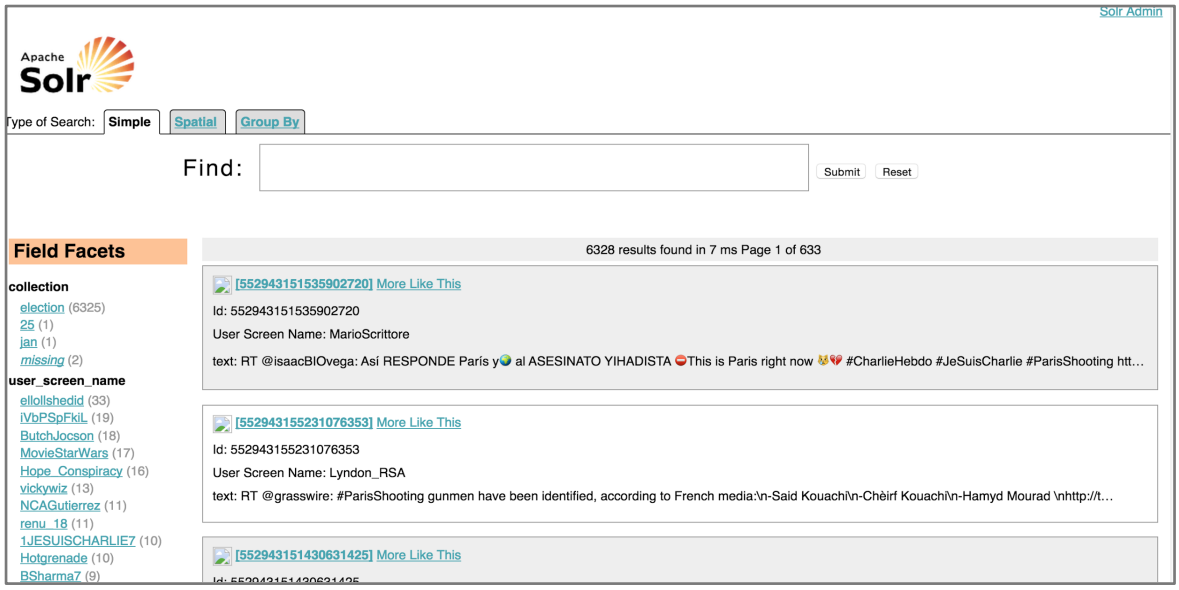

Figure 6.1: Cross-collection browse function with facets, included in Solr but not yet available on the IDEAL cluster.

To set up port forwarding, from the command line type "ssh -L 9983:localhost:8983 <username>@hadoop.dlib.vt.edu" and log in. The Solr Admin console will be available at http://localhost:9983/solr/#/.

Text queries to the IDEAL Solr collections are parsed by the Solr Extended Disjunction Maximum (edismax) query parser, which assigns boosts to the search fields as follows:

- text
- collection^3
- hashtags^3
- cluster_label^2.5
- lda_topics^2.0
- ner_people^2.0
- ner_locations^2.0
- ner_organizations^2.0

Documents will be sorted by a combination of relevance and the "Social Importance" value assigned by the Social Networks Team. If the result list is short, it will be supplemented with documents that cover similar topics to those in the original result list by consulting the collection-level topic models assigned by the LDA team. All non-null fields will be displayed, as in the example tweet result in Figure 6.2.

```
{
  "text": "Winter Storm Coming    storm     ",
  "user_screen_name": "stripesnsparkle",
  "urls": [
    "http://t.co/zuzA8x1bhe"
  ],
  "tweet_id": 3834306245955107000,
  "cluster_id": "winter_storm_S--c5",
  "collection": "winter_storm_S",
  "lang": "English",
  "id": "winter_storm_S--299751",
  "source": "twitter-search",
  "created_at": "2015-02-01T06:07:32Z",
  "cluster_label": "storm",
  "user_id": "2914255473",
  "hashtags": [
    "Chicago",
    "snow",
    "winter",
    "winterstorm",
    "cold",
    "outside",
    "white",
    "babyitscoldoutside"
  ],
  "coordinates": [
    "0.0,0.0"
  ],
  "_version_": 1500228833549222000
},
```


Figure 6.2: Tweet search result in the Solr Admin Console.

7 Developer Manual

This section is divided into two parts: first, a technical overview of the overall project architecture, and second, a step-by-step guide to all the different configurations needed to set up Solr. All source code and configuration files are included with this report in VTechWorks. Kindly refer to Section 8.

7.1 Technical Specification – Project Architecture

Development was mainly done on our local installations of the Cloudera Virtual Machine. The installation tutorial can be found in Scholar under Resources/Tutorials. This VM, popularly known as Cloudera's Quickstart VM, runs on CentOS 6.4 which is a 64 bit Linux based OS, is configured by default with a single node Hadoop cluster version CDH 5.4.x and also has Cloudera Manager installed to manage the cluster. It requires a minimum of 4GB RAM to run seamlessly.

The production cluster has Hadoop version 5.3.1 installed. The cluster has the following specification. The specification was shared in the class and we copied it from the Hadoop team's project report.

- CPU
  - Intel i5 Haswell Quad core 3.3 Ghz Xeon
- RAM
  - 660 GB in total
  - 32 GB in each of the 19 Hadoop nodes
  - 4 GB in the manager node
  - 16 GB in the tweet DB nodes
  - 16 GB in the HDFS backup node
- Storage
  - 60 TB across Hadoop, manager, and tweet DB nodes
  - 11.3 TB for backup
- Number of nodes
  - 19 Hadoop nodes
  - 1 Manager node
  - 2 Tweet DB nodes
  - 1 HDFS backup node

The eight teams in the course worked on multiple areas with the common goal of building the best Information Retrieval System. All the teams worked in close coordination and collaboration to make this project a success. Figure 7.1 is a flow chart (courtesy Hadoop team) on how different teams' efforts fit together within the system.

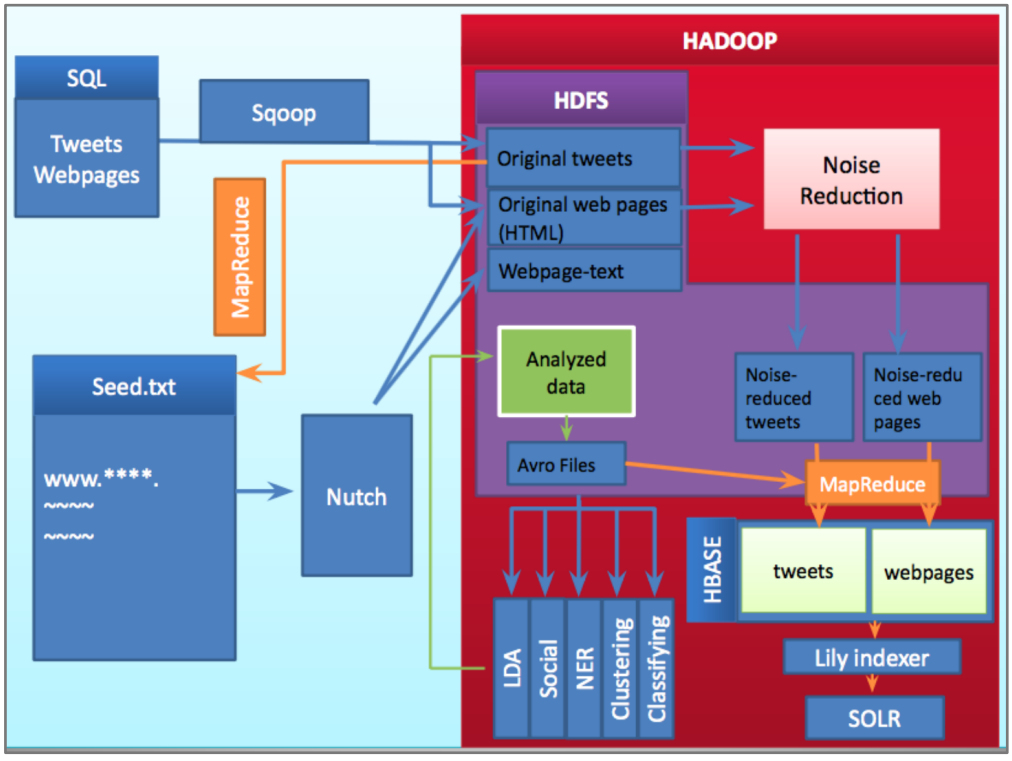

Figure 7.1: Project Architecture

7.2 IDEAL Custom Search

The configuration directories for the existing Solr collections can be found on node1.dlrl.vt.edu under /home/cs5604s15_solr/solr_collections. To make changes to the collections, for example to modify solrconfig.xml or schema.xml, edit the relevant files and issue "solrctl instancedir –update <collection> <collection>" from /home/cs5604s15_solr/solr_collections. Then issue: "solrctl –reload <collection>".

To modify our search customizations, edit the solrconfig.xml file, as shown in Figure 7.2. Our application uses the standard SearchHandler mapped under the "/select" URL pattern, but makes heavy use of default edismax boost parameters and custom SearchComponents.

```
<requestHandler name="/select" class="solr.SearchHandler">
    <lst name="defaults">
        <str name="defType">edismax</str>
        <str name="qf">
            text
            collection^3
            hashtags^3
            cluster_label^2.5
            lda_topics^2.0
            ner_people^2.0
            ner_locations^2.0
            ner_organizations^2.0
        </str>
        <str name="echoParams">explicit</str>
        <int name="rows">10</int>
        <str name="df">text</str>
        <str name="fl">*,score</str>
    </lst>
    <arr name="last-components">
        <str>idealSocialBoostComponent</str>
        <str>idealTopicSupplementComponent</str>
    </arr>
</requestHandler>
```

Figure 7.2: Solrconfig.xml configurations for our custom search.

Solr search is designed as a plugin module pattern, where the SearchHander performs the orchestration and delegates processing to a set of SearchComponents configured in solrconfig.xml. The SearchHandler first parses the request string into a Query object using either the default Lucene QueryParser or the query parser specified in the URL with the defType parameter. The SearchHandler then iterates through each SearchComponent calling prepare() and then process(), each time passing a reference to a ResponseBuilder object that contains, among other objects, a list of documents retrieved so far.

If no components are configured, Solr uses the default set, which, in order, has entries shown in Figure 7.3.

Component Name	Class Name	More Information
query	solr.QueryComponent	Described in the section Query Syntax and Parsing.
facet	solr.FacetComponent	Described in the section Faceting.
mlt	solr.MoreLikeThisComponent	Described in the section MoreLikeThis.
highlight	solr.HighlightComponent	Described in the section Highlighting.
stats	solr.StatsComponent	Described in the section The Stats Component.
debug	solr.DebugComponent	Described in the section on Common Query Parameters.
expand	solr.ExpandComponent	Described in the section Collapse and Expand Results.

Figure 7.3: Default Search Components in Solr.

Our two custom components are configured in solrconfig.xml as shown in Figure 7.4.

```
<searchComponent name="idealSocialBoostComponent" class="edu.vt.dlib.ideal.solr.IDEALSocialBoostComponent"/>
<searchComponent name="idealTopicSupplementComponent" class="edu.vt.dlib.ideal.solr.IDEALTopicSupplementComponent"/>
```

Figure 7.4: Two custom Search Components developed for the IDEAL project.

The Search Components are declared as “last-components,” in the SearchHandler tag, so they will be the final processing steps before a response is rendered to the user. The Java source code files for these classes are attached to this report and are also available under /home/cs5604s15_solr/src on node1.dlrl.vt.edu.

The IDEALSocialBoostComponent reorders the document result list according to the “social importance” field populated by the Social Networks team. The social importance score, which ranges from 0 to 1, was calculated on the basis of how often a tweet was shared and how prominent its author is in the social network, as shown in Figure 7.5. We add the social importance score to the relevance score so that social importance contributes to relevance, but doesn’t dominate it.

```
DocIterator iterator = docList.iterator();
while (iterator.hasNext()) {

    int docId = iterator.nextDoc();

    float score = iterator.score();

    Document d = rb.req.getSearcher().doc(docId);
    float socialBoost = 0;
    IndexableField socialImportanceField = d.getField("social_importance");
    if(socialImportanceField != null) {
        Number socialImportance = socialImportanceField.numericValue();
        if (socialImportance != null) {
            socialBoost = socialImportance.floatValue();

        }
        score = score + socialBoost;

    }
    scoreDocs[idx++] = new ScoreDoc(docId, score);
}
```

Figure 7.5: IDEALSocialBoostComponent.

Our second custom Search Component (IDEALTopicSupplementComponent) augments short result lists with documents that cover topics similar to those in the short list, as shown in Figure 7.6. We first see which collection dominates the small list, and then retrieve that collection's LDA topic model from the HBase table called "collection_metadata".

Figure 7.6: Sample collection LDA topic model JSON string.

Based on the words in that topic model, we create a new search string and query the Lucene index again, adding any new relevant documents to the result list, as shown in Figure 7.7 and Figure 7.8.

```
@Override
public void process(ResponseBuilder rb) throws IOException {

    DocListAndSet results = rb.getResults();

    SortSpec sortSpec = rb.getSortSpec();
    int len = sortSpec.getCount();
    int offset = sortSpec.getOffset();

    DocList docList = rb.getResults().docList;
    if (docList.size() < MIN_DOCS) {
        String collectionName = getDominantCollection(rb);
        TopicModel topicModel = getCollectionTopicModel(collectionName);
        String topTopics = topicModel.getTopTopics();

        QueryParser queryParser = new QueryParser("name", new StandardAnalyzer());
        Query query = null;
        try {
            query = queryParser.parse("q=" + topTopics);

        } catch (ParseException e) {
            e.printStackTrace();
        }

        TopDocs additionalDocs = rb.req.getSearcher().search(query, 100);
        ScoreDoc[] scoreDocs = additionalDocs.scoreDocs;
        int totalHits = additionalDocs.totalHits + docList.matches();

        int[] docs = new int[scoreDocs.length];
        float[] scores = new float[scoreDocs.length];
        for (int i = 0; i < scoreDocs.length; i++) {
            docs[i] = scoreDocs[i].doc;
            scores[i] = scoreDocs[i].score;
        }
```

Figure 7.7: Search Component that adds documents to a short result list based LDA topic similarity.

```
private TopicModel getCollectionTopicModel(String collectionName) throws IOException {

    TopicModel topicModel =new TopicModel(collectionName);
    HTable table = new HTable(conf, "collection_metadata");
    Get get = new Get(Bytes.toBytes(collectionName));
    byte[] val = table.get(get).getValue(b("analysis"), b("lda"));
    String jsonTopicModel = new String(val, StandardCharsets.UTF_8);

    Map<String, Double> map = new Gson().fromJson(jsonTopicModel, new TypeToken<HashMap<String, Double>>() {}.getType(
    topicModel.setTopicProbabiliities(map);

    return topicModel;

}
```

Figure 7.8: Retrieving collection-level metadata from HBase directly from Solr.

Extending the search capabilities of the IDEAL Solr application offers some promising areas of future research. More disciplined estimates of field weights could be deduced using test queries and relevance feedback. Also, we might consider boosting tweets that are longer, and thus carry more information, while penalizing longer webpages for being less focused. Implementation of a faceted search interface is trivial in Solr, but we might consider more elegant input interfaces, such as social graph node selectors or cluster regions.

Table 7.1 below summarizes the metadata received from each team and how that metadata was employed as part of our search.

Team	Sample metadata	How employed

Clustering	cluster_label: “storm”	cluster labels weighted heavily (2.5)
Classification	classification_labels: [“NEGATIVE”]	used in the future to remove documents from wrong collections
LDA	lda_topics: [“shooting”]	weighted 2.4 in search
NER	ner_people: [“Barrack Obama”]	people, locations, and organizations are weighted 2.0 and will all be facets in Solritas when velocity is enabled
Social Networks	importance_score: .233	Used to boost documents during ranking

Table 7.1 Team metadata and its employment in the IDEAL custom search.

7.3 HBase-Solr Indexing

Documents are inserted into Solr with the following process:

- Insert the documents into HBase
- Index the documents into Solr using the Lily Batch Indexer,
- Push modified HBase rows out to Solr using the Lily NRT indexer.

HBase Indexer Configuration

a. Insert documents:

i. Create the tables (each with two column families, ‘original’ and ‘analysis’) in the HBase shell using the commands in Figure 7:

```
create 'event_tweets', 'original', 'analysis'
create 'event_webpages', 'original', 'analysis'
```

Figure 7.9: Commands to create HBase tables.

ii. Insert documents into HBase. The following example uses a Tweet class with the following save() method in Figure 7.10:

```
HTable table = new HTable(conf, tableName);

Put put = new Put(b(this.getId()));
put.add(b("original"), b("text_original"),
b(this.getTextOriginal()));
put.add(b("original"), b("text_clean"),
b(this.getTextClean()));
put.add(b("original"), b("created_at"),
b(this.getCreatedAt()));
put.add(b("original"), b("source"), b(this.getSource()));
put.add(b("original"), b("user_screen_name"),
b(this.getUserScreenName()));
put.add(b("original"), b("user_id"), b(this.getUserId()));
put.add(b("original"), b("lang"), b(this.getLang()));
put.add(b("original"),
```

```
b("retweet_count"),b(String.valueOf(this.getRetweetCount())));
put.add(b("original"), b("favoriteCount"),
b(String.valueOf(this.getFavoriteCount())));
put.add(b("original"), b("contributorsId"),
b(this.getContributorsId()));
 put.add(b("original"), b("coordinates"),
 b(this.getCoordinates()));
 put.add(b("original"), b("urls"), b(this.getUrls()));
 put.add(b("original"), b("hashTags"), b(this.getHashTags()));
put.add(b("original"), b("user_mentions_id"),
b(this.getUserMentionsId()));
put.add(b("original"), b("in_reply_to_user_id"),
b(this.getInReplyToUserId()));
put.add(b("original"), b("in_reply_to_status_id"),
b(this.getInReplyToStatusId()));
put.add(b("analysis"), b("ner_people"),
b(this.getNerPeople()));
put.add(b("analysis"), b("ner_locations"),
b(this.getNerLocations()));
put.add(b("analysis"), b("ner_dates"), b(this.getNerDates()));
put.add(b("analysis"), b("ner_organizations"),
b(this.getNerOrganizations()));
put.add(b("analysis"), b("cluster_id"),
b(this.getClusterId()));
put.add(b("analysis"), b("cluster_label"),
b(this.getClusterLabel()));
put.add(b("analysis"), b("classification_vector"),
b(this.getClassificationVector()));
put.add(b("analysis"), b("social_vector"),
b(this.getSocialVector()));
put.add(b("analysis"), b("lda_vector"),
b(this.getLdaVector()));

table.put(put);
```

Figure 7.10: Tweet.save(Configuration conf) method.

Verify through Hue that the documents were inserted correctly. See Figure 7.11 below.

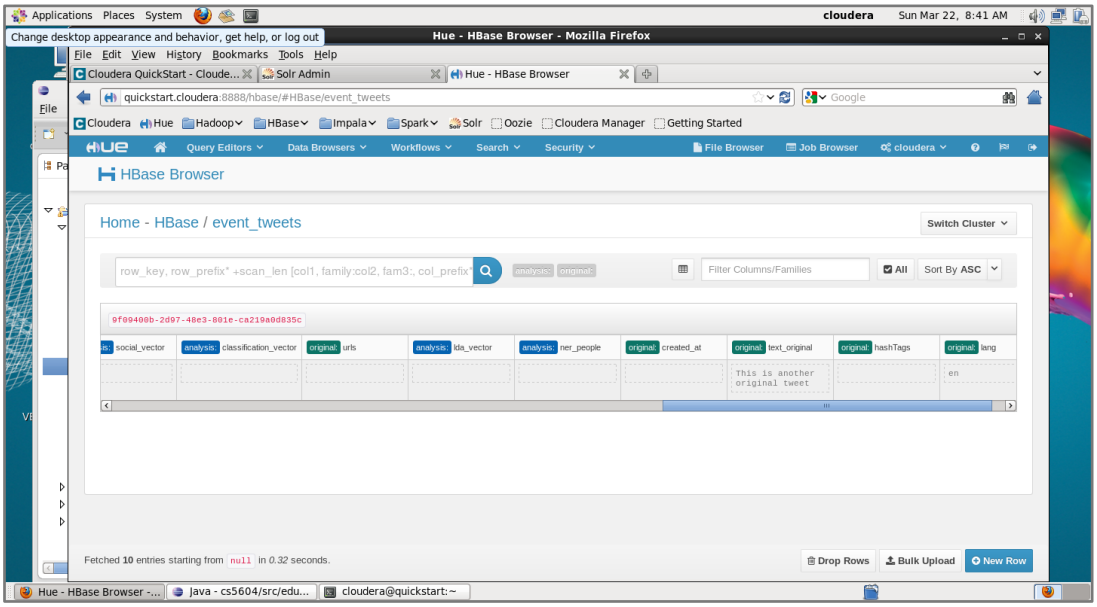

Figure 7.11: event_tweets table with column families and qualifiers.

b. Index the HBase documents into Solr.

1. Install zookeeper

```
$ sudo yum install zookeeper
```

2. Create a Solr cloud collection in $HOME=/home/cloudera. Edit the schema based on your HBase schema.

```
$ solrctl instancedir --generate conf
$ edit $HOME/conf/conf/schema.xml
$ solrctl instancedir --create hbase-collection1 conf
$ solrctl collection --create hbase-collection1 -s 1
```

3. Create a $HOME/morphline-hbase-mapper.xml file.

```
<?xml version="1.0"?>
<indexer table="record"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">
   <param name="morphlineFile" value="/etc/hbase-solr/conf/morphlines.conf"/>
</indexer>
```

4. Create a morphline configuration file in `/etc/hbase-solr/conf/morphlines.conf` as shown in Figure 7.12 below.

5. Run the Indexer tool using this command

```
hadoop --config /etc/hadoop/conf jar \
/usr/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar --conf \
/etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx500m'\
--hbase-indexer-file $HOME/morphline-hbase-mapper.xml --zk-host \
127.0.0.1/solr --collection hbase-collection1 --go-live --log4j \
src/test/resources/log4j.properties
```

The central challenge is in dealing with multivalued fields such as “ner_people.” Assuming that the cells holding multivalued fields in HBase will be single strings with “|” separators, we were able to index into Solr using the “split” morphline command in Figure 7.12.

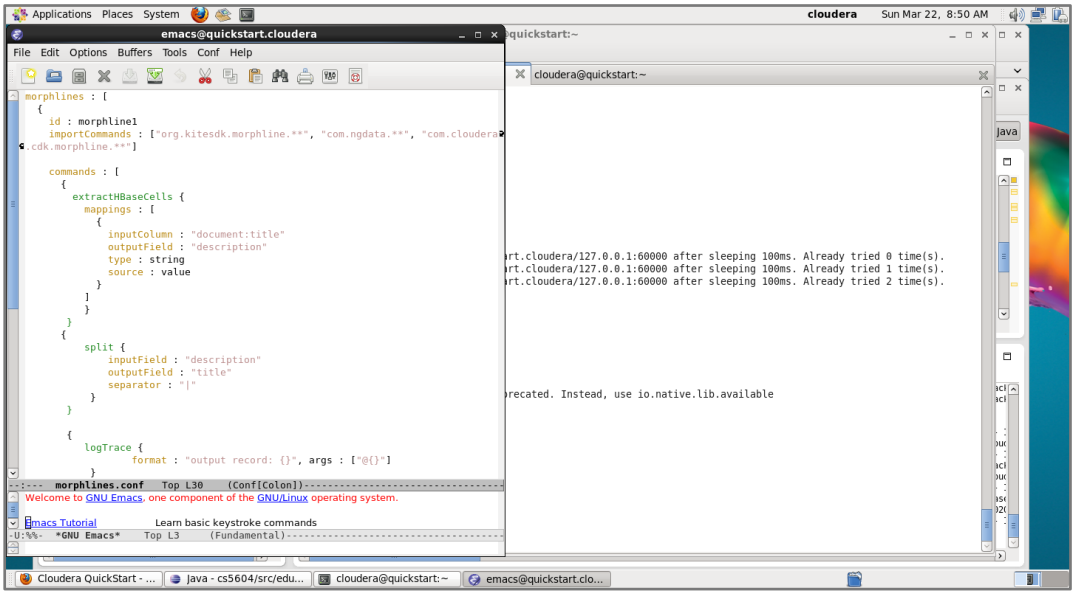

Figure 7.12: Morphline to index HBase data to Solr.

In this example, we’re splitting the title field into multiple values, and the result in Solr looked like this:

```
"title": [
          "title 1",
          "title 2",
          "title 3"
        ],
```

Each team that populated a multivalued field in Solr inserted the values in a single HBase Column with a bar separator, such as in the following row in Figure 7.13:

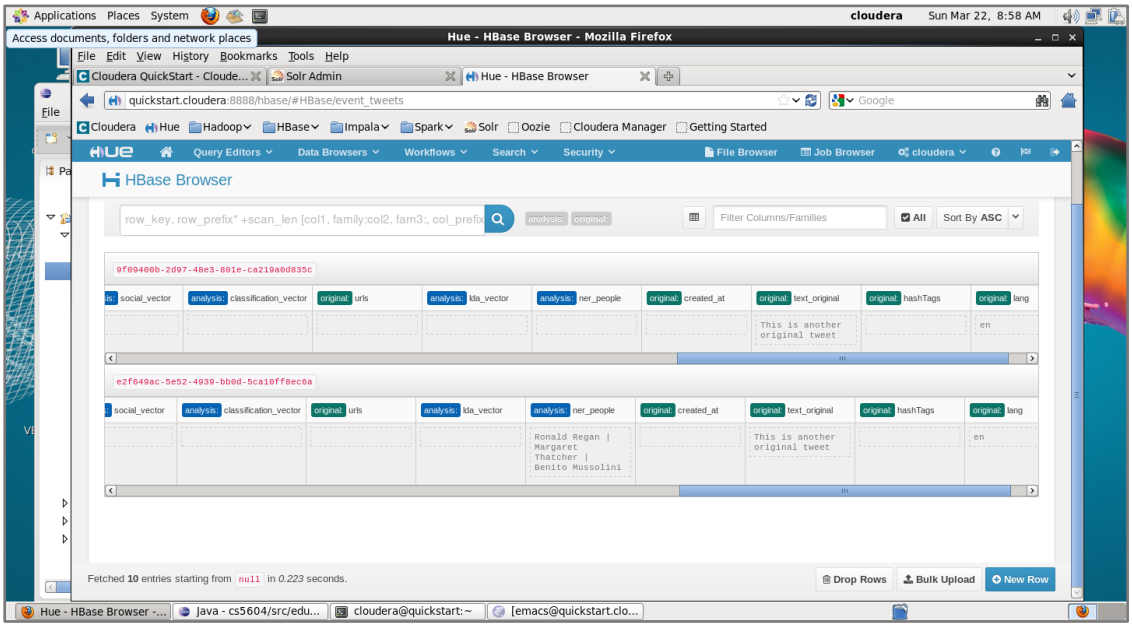

Figure 7.13: Updated HBase rows with multivalued fields.

The sample of the final version of the morphline is listed in Figure 7.14. Please check Appendix C at the end of this report for the complete file.

```
SOLR_LOCATOR : {
 # Name of solr collection
 collection : tweets

 # ZooKeeper ensemble
 zkHost : "node1.dlrl:2181/solr"
}
morphlines: [
       {
            id: morphline1
            importCommands: ["org.kitesdk.morphline.**",
"com.ngdata.**", "com.cloudera.cdk.morphline.**",
"org.apache.solr.**"]
            commands: [
                {
                    extractHBaseCells {
                        mappings: [
                            {
                                inputColumn: "original:text_clean"
                                outputField: "text"
                                type: string
                                source: value
                            }
                            {
                                inputColumn: "original:created_at"
                                outputField: "created_at"
                                type: string
```

```
                                    source: value
                                }
                                 {
                                    inputColumn: "analysis:lda_topics"
                                    outputField: "lda_topics_multiple"
                                    type: string
                                    source: value
                                }
                            ]
                        }
                    }
                    {
                        split {
                            inputField: "lda_topics_multiple"
                            outputField: "lda_topics"
                            separator: "|"
                        }
                    }
{
sanitizeUnknownSolrFields {
# Location from which to fetch Solr schema
solrLocator : ${SOLR_LOCATOR}
}
}
{
convertTimestamp {
field : created_at
inputFormats : ["unixTimeInMillis"]
inputTimezone : UTC
outputFormat : "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
outputTimezone : UTC
}
}
{
logTrace {
format : "output record: {}", args : ["@{}"]
}
}
]
}
]
```

Figure 7.14: A sample of the original Tweets_morphlines.xml

The morphlines specify the ETL procedures that are completed during the map phase of the MapReduce program (HBaseMapReduceIndexerTool/HBaseIndexerMapper). Documents are extracted from the HBase cells and packaged into SolrInputDocument data structures, which are passed to reducers that index them into separate temporary microshards in embedded Solr instances. A second MapReduce program (ForkedMapReduceIndexerTool) performs the “Go Live” phase, in which all of the microshards are merged into a production SolrCloud. Our tweet collection was split into 10 mappers and 36 reducers. To run the batch indexer on node1.dlrl.vt.edu execute

either /home/cs5604s15_solr/solr_collections/indexer/index_tweets.sh or /home/cs5604s15_solr/solr_collections/indexer/index_webpages.sh. This shell script runs the command in Figure 7.15.

```
 hadoop --config /etc/hadoop/conf jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar --conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx1000m' --hbase-indexer-file $HOME/solr_collections/indexer/tweet_morphline-hbase-mapper.xml --collection tweets --zk-host node1.dlrl:2181/solr --go-live --log4j $HOME/solr_collections/log4j.properties
```

Figure 7.15: Hadoop command to batch index the tweet collection.

Note that the command references the “tweet_morphline-hbase-mapper.xml” file, which designates the name of the HBase table to index and the applicable Morphline file, as shown in Figure 7.16.

```
<indexer table="webpages"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">
   <param name="morphlineFile"
value="/home/cs5604s15_solr/solr_collections/indexer/webpages_morphlines.conf"
/>
</indexer>
```

Figure 7.16: Configuration file for tweet indexing, tweet_morphline-hbase-mapper.xml

c. Use the Lily NRT to re-index the document in Solr. This indexer runs as a service on the cluster, and its properties can be viewed with the ‘hbase-indexer” list-indexers” command as shown below in Figure 7.17.

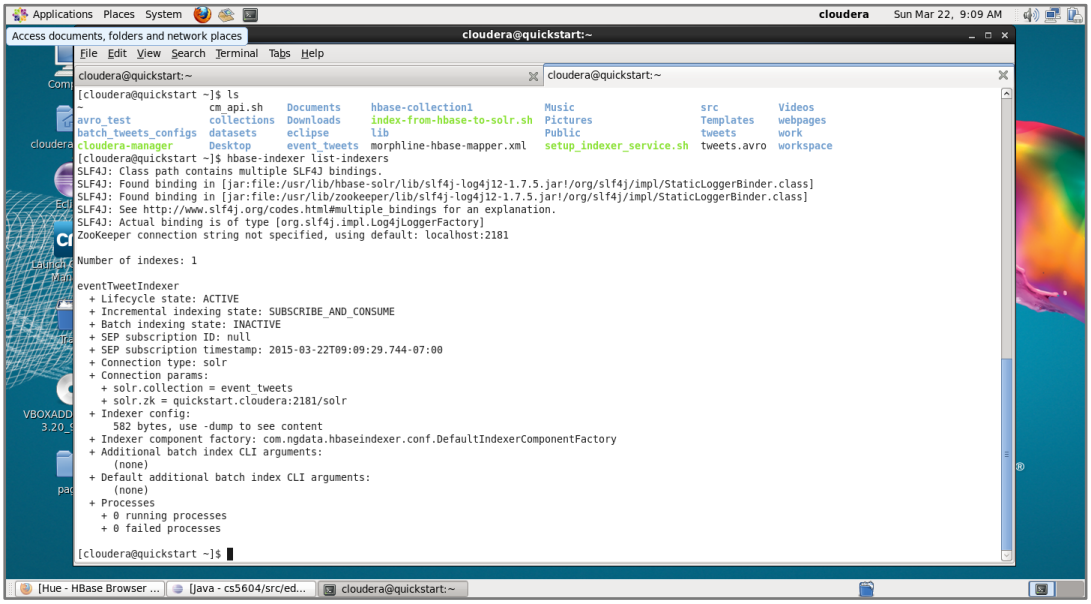

Figure 7.17: Output of “hbase-indexer list-indexers” command.

We followed instructions [18] given below to configure the Near-Real-Time Indexer, which automatically re-indexes rows that are updated in the source HBase table. The following procedures assume that the HBase Indexer is already installed.

1. You can have multiple Lily HBase Indexer services running on different nodes as is required by HBase to ingest data. Consult replication documentation for details. Run the command below to install the replication documentation.

```
$ sudo yum install hbase-solr-indexer hbase-solr-doc
```

2. Copy the following to /etc/hbase-solr/conf/hbase-indexer-site.xml

```
 <property>
   <name>hbase.zookeeper.quorum</name>
   <value>localhost</value>
</property>
<property>
   <name>hbaseindexer.zookeeper.connectstring</name>
   <value>localhost:2181</value>
</property>
```

3. Copy the following to /etc/hbase/conf/hbase-site.xml

```
  <!-- SEP is basically replication, so enable it -->
  <property>
    <name>hbase.thrift.info.bindAddress</name>
    <value>0.0.0.0</value>
   </property>
  <property>
    <name>hbase.replication</name>
    <value>true</value>
   </property>
   <!-- Source ratio of 100% makes sure that each SEP consumer is actually used
(otherwise, some can sit idle, especially with small clusters) -->
   <property>
    <name>replication.source.ratio</name>
    <value>1.0</value>
   </property>
   <!-- Maximum number of hlog entries to replicate in one go. If this is large, and
a consumer takes a while to process the events, the HBase rpc call will time out. -->
   <property>
    <name>replication.source.nb.capacity</name>
    <value>1000</value>
   </property>
   <!-- End configuring HBase cluster replication -->

   <!-- zookeeper configurations -->
   <property>
    <name>hbase.zookeeper.quorum</name>
    <value>localhost</value>
   </property>
   <property>
    <name>hbaseindexer.zookeeper.connectstring</name>
    <value>localhost:2181</value>
```

```
    </property>
    <!-- end zookeeper configurations -->
```

4. Once the contents of the HBase Indexer configuration XML file is ready, register this with Lily HBase Indexer service. This is done by uploading the configuration xml to zookeeper

```
hbase-indexer add-indexer \
--name testIndexer \
--indexer-conf $HOME/morphline-hbase-mapper.xml \
--connection-param solr.zk=localhost:2181/solr \
--connection-param solr.collection=hbase-collection1 \
--zookeeper localhost:1
```

5. Start Lily based NRT Indexer service

```
$ sudo service hbase-solr-indexer restart
```

6. Verify that the indexer is running successfully by using the following command.

```
$ hbase-indexer list-indexers
Number of indexes: 1

myIndexer
  + Lifecycle state: ACTIVE
  + Incremental indexing state: SUBSCRIBE_AND_CONSUME
  + Batch indexing state: INACTIVE
  + SEP subscription ID: Indexer_myIndexer
  + SEP subscription timestamp: 2013-06-12T11:23:35.635-07:00
  + Connection type: solr
  + Connection params:
    + solr.collection = hbase-collection1
    + solr.zk = localhost/solr
  + Indexer config:
      110 bytes, use -dump to see content
  + Batch index config:
      (none)
  + Default batch index config:
      (none)
  + Processes
    + 1 running processes
    + 0 failed processes
```

Troubleshoot:

If you encounter errors like “org.apache.zookeeper.ClientCnxn - Session 0x0 for server null, unexpected error, closing socket connection and attempting reconnect”, then restart HBase using the commands below.

```
$ sudo /etc/init.d/hbase-master stop
$ sudo /etc/init.d/zookeeper-server stop
$ sudo /etc/init.d/hbase-regionserver stop

$ sudo /etc/init.d/hbase-regionserver start
$ sudo /etc/init.d/zookeeper-server start
$ sudo /etc/init.d/hbase-master start
```

8 Inventory

File	Description
Solr Team Final Report	This document
tweet_schema.xml	Solr schema for tweets
webpages_schema.xml	Solr schema for webpages.
solrconfig.xml	Solr configuration of our custom search
tweet_morphlines.conf	Tweet Indexing Morphline file
webpages_morphlines.conf	Webpage Indexing Morphline file
IDEALSocialBoostComponent.java	Reordering Search Component.
IDEALTopicSupplementComponent.java	Result list augmentation Search Component.
test_solr.py	Python script for testing Solr searches

9 Acknowledgement

We are especially thankful to NSF grant IIS - 1319578, III: Small: Integrated Digital Event Archiving and Library (IDEAL) for the funding that supported the infrastructure and the data used in the project. We are thankful to Dr. Edward A. Fox for designing an excellent project based learning course, being the guide on the side for us, and for coordinating efforts of all the teams to help us make it a successful class project. We are also thankful to the GTA, GRA and other students of the class who supported us with ideas as well as efforts during the semester. Thanks to the authors and the contributors of open source projects, blogs and wiki pages from where we borrowed some ideas and solutions.

10 References

[1] Manning, Christopher D, et al. An Introduction to Information Retrieval. Cambridge University Press, Cambridge, 2009.
[2] Grainger, Trey, Timothy Potter, and Yonik Seeley. *Solr in action*. Manning Publications Co., Shelter Island, NY, 2014.
[3] Apache Solr Reference Guide Covering Apache Solr 4.10. Date accessed: 2015-05-06. URL: https://archive.apache.org/dist/lucene/solr/ref-guide/apache-solr-ref-guide-4.10.pdf
[4] 2012. IntegratingSolr - Solr Wiki - Apache Wiki. Date accessed: 2015-05-06. URL: https://wiki.apache.org/solr/IntegratingSolr
[5] McCandless, Michael, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action: Covers Apache Lucene 3.0*. Manning Publications Co., Shelter Island, NY, 2010.
[6] Baeza-Yates, Ricardo and Ribeiro-Neto, Berthier. Modern Information Retrieval: The Concepts and Technology behind Search, Second Edition. ACM Press Books. 2011
[7] Bird, Steven, et al. Natural Language Processing with Python. O'Reilly, 2009.
[8] Kuc, Rafal. Apache Solr 4 Cookbook. Packt Publishing, 2013.
[9] Busch, Michael, et al. "Earlybird: Real-time search at twitter." Data Engineering (ICDE), 2012 IEEE 28th International Conference on. IEEE, 2012.
[10] Białecki, Andrzej, Robert Muir, and Grant Ingersoll. "Apache lucene 4." SIGIR 2012 workshop on open source information retrieval. 2012.
[11] Stevenson, Mark, and Yorick Wilks. "Word sense disambiguation." The Oxford Handbook of Comp. Linguistics (2003): 249-265.
[12] Anil, Robin, Ted Dunning, and Ellen Friedman. *Mahout in action*. Manning, 2011.
[13] Schema.xml, Solr Team VTechworks repository. Date accessed: 2015-05-06. URL: https://vtechworks.lib.vt.edu/handle/10919/19081/submit
[14] SolrConfig.xml, Solr Team VTechworks repository. Date accessed: 2015-05-06. URL: https://vtechworks.lib.vt.edu/handle/10919/19081/submit
[15] Python code to index document, Solr Team VTechworks repository. Date accessed: 2015-05-06. URL: https://vtechworks.lib.vt.edu/handle/10919/19081/submit
[16] 2015 Cloudera, Inc. Cloudera QuickStart. Date accessed: 2015-05-06. URL: http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/quickstart.html
[17] A.Choudhury, R.Gruss, J.Cadena, N.Komawar et.al. "CS5604 Spring 2015: Proposed HBase Schema," Google Docs. Virginia Tech. March 22, 2015. URL: https://docs.google.com/document/d/18GrMmbcLgY7F6xKBIUW4LMJpnwpLmmZhfKmlgARb7qc/edit
[18] "Cloudera Search Installation Guide", Cloudera Inc. Date accessed: 2015-05-06. URL: http://www.cloudera.com/content/cloudera/en/documentation/cloudera-search/v1-latest/Cloudera-Search-User-Guide/Cloudera-Search-User-Guide.html
[19] Dimiduk, Nick, et al. HBase in action. Shelter Island: Manning, 2013.
[20] "Lily Indexer Documentation", Date accessed: 2015-05-06. URL: http://docs.ngdata.com/lily-docs-current/index.html
[21] "Solr 4.9.0 API." Solr 4.9.0 API. The Apache Software Foundation, n.d. Web. 05 Apr. 2015.
[22] "Carrot².” Carrot2: Open Source framework for building search clustering engines. Monday April 6 2015. URL: http://project.carrot2.org/index.html

[23] What's Big Data? It's not so complicated. (n.d.). Retrieved May 13, 2015, from https://whatsbigdata.be/tag/hregionserver/

[23] “Results Clustering.” Apache Wiki. Monday April 6 2015. URL: https://cwiki.apache.org/confluence/display/solr/Result+Clustering
[24] “Carrot2 Algorithms.” Carrot2: Open Source framework for building search clustering engines. Monday April 6 2015. URL: http://project.carrot2.org/algorithms.html
[25] “Apache SOLR and Carrot2 integration strategies” Carrot2 Github documentation. Date accessed: 2015-05-06. URL: http://carrot2.github.io/solr-integration-strategies/carrot2-3.8.0/
[26] “CTRnet Events Archive”, Date accessed: 2015-05-06. URL: http://www.eventsarchive.org
[27] “Using the Lily HBase Batch Indexer for Indexing” Cloudera, Date accessed: 2015-05-06. URL: http://www.cloudera.com/content/cloudera/en/documentation/cloudera-search/v1-latest/Cloudera-Search-User-Guide/csug_hbase_batch_indexer.html
[28] “How can I approximately calculate the Solr index size “. Date accessed: 2015-05-06. URL: http://stackoverflow.com/questions/17640177/how-can-i-approximately-calculate-the-solr-index-size
[29] “Generate UUIDs (or GUIDs) in Java” Johann Burkard, Date accessed: 2015-05-06. URL: http://johannburkard.de/software/uuid/
[30] “Choosing a fast unique identifier (UUID) for Lucene”, Date accessed: 2015-05-06. URL: http://www.javacodegeeks.com/2014/05/choosing-a-fast-unique-identifier-uuid-for-lucene.html
[31] “Static Fields Performance vs Dynamic Fields Performance”, Date accessed: 2015-05-06. URL: http://lucene.472066.n3.nabble.com/Static-Fields-Performance-vs-Dynamic-Fields-Performance-td4160316.html

11 Appendix

Here we have added all the configuration files required to set up the search.

A. Solr Schema.xml - for Tweets

```
<?xml version="1.0" encoding="UTF-8" ?>

<schema name="IDEAL_tweets" version="0.1">

    <fields>

        <field name="id" type="string" indexed="true" stored="true" required="true"
multiValued="false"/>

        <field name="tweet_id" type="long" indexed="true" stored="true"
required="true" multiValued="false"/>

        <field name="collection" type="text_general" indexed="true" stored="true"
required="true" multiValued="false"/>

        <field name="text" type="text_general" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="created_at" type="tdate" indexed="true" stored="true"
multiValued="false"/>

        <field name="source" type="string" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="user_screen_name" type="string" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="user_id" type="string" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="lang" type="string" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="retweet_count" type="tint" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="favorite_count" type="tint" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="_version_" type="tlong" indexed="true" stored="true"
multiValued="false"/>

        <field name="contributors_id" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="coordinates" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="urls" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
```

```
        <field name="hashtags" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="user_mentions_id" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="in_reply_to_user_id" type="tlong" indexed="true" stored="true"
required="false"
               multiValued="false"/>

        <field name="in_reply_to_status_id" type="tlong" indexed="true" stored="true"
required="false"
               multiValued="false"/>

        <!-- class analysis fields -->
        <field name="ner_people" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="ner_locations" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="ner_dates" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="ner_organizations" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="cluster_id" type="string" indexed="false" stored="true"
required="false" multiValued="false"/>
        <field name="cluster_label" type="string" indexed="false" stored="true"
required="false" multiValued="false"/>

        <field name="classification_vector_json" type="string" indexed="false"
stored="true" required="false"
               multiValued="false"/>
        <field name="classification_labels" type="string" indexed="true"
stored="true" required="false"
               multiValued="true"/>

        <field name="social_vector_json" type="string" indexed="false" stored="true"
required="false"
               multiValued="false"/>

        <field name="social_importance" type="double" indexed="false" stored="true"
required="false" multiValued="false"/>

        <field name="lda_dict_json" type="string" indexed="false" stored="true"
required="false" multiValued="false"/>

        <field name="lda_topics" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="urls_multiple" type="string" indexed="false" stored="false"
required="false" multiValued="false"/>

        <field name="hashtags_multiple" type="string" indexed="false" stored="false"
required="false"
               multiValued="false"/>

        <field name="ner_people_multiple" type="string" indexed="false"
stored="false" required="false"
               multiValued="false"/>
```

```
        <field name="ner_locations_multiple" type="string" indexed="false"
stored="false" required="false"
               multiValued="false"/>

        <field name="ner_dates_multiple" type="string" indexed="false" stored="false"
required="false"
               multiValued="false"/>

        <field name="ner_organizations_multiple" type="string" indexed="false"
stored="false" required="false"
               multiValued="false"/>

        <field name="lda_topics_multiple" type="string" indexed="false"
stored="false" required="false"
               multiValued="false"/>

        <field name="classification_labels_multiple" type="string" indexed="false"
stored="false" required="false"
               multiValued="false"/>

        <dynamicField name="*_i" type="int" indexed="true" stored="true"/>
        <dynamicField name="*_is" type="int" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_s" type="string" indexed="true" stored="true"/>
        <dynamicField name="*_ss" type="string" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_l" type="long" indexed="true" stored="true"/>
        <dynamicField name="*_ls" type="long" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_t" type="text_general" indexed="true" stored="true"/>
        <dynamicField name="*_txt" type="text_general" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_en" type="text_en" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_b" type="boolean" indexed="true" stored="true"/>
        <dynamicField name="*_bs" type="boolean" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_f" type="float" indexed="true" stored="true"/>
        <dynamicField name="*_fs" type="float" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_d" type="double" indexed="true" stored="true"/>
        <dynamicField name="*_ds" type="double" indexed="true" stored="true"
multiValued="true"/>

        <!-- Type used to index the lat and lon components for the "location"
FieldType -->
        <dynamicField name="*_coordinate" type="tdouble" indexed="true"
stored="false"/>

        <dynamicField name="*_dt" type="date" indexed="true" stored="true"/>
        <dynamicField name="*_dts" type="date" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_p" type="location" indexed="true" stored="true"/>

        <!-- some trie-coded dynamic fields for faster range queries -->
        <dynamicField name="*_ti" type="tint" indexed="true" stored="true"/>
        <dynamicField name="*_tl" type="tlong" indexed="true" stored="true"/>
        <dynamicField name="*_tf" type="tfloat" indexed="true" stored="true"/>
        <dynamicField name="*_td" type="tdouble" indexed="true" stored="true"/>
        <dynamicField name="*_tdt" type="tdate" indexed="true" stored="true"/>
```

```
        <dynamicField name="ignored_*" type="ignored" multiValued="true"/>
        <dynamicField name="attr_*" type="text_general" indexed="true" stored="true"
multiValued="true"/>

        <dynamicField name="random_*" type="random"/>

    </fields>

    <uniqueKey>id</uniqueKey>

    <types>

        <fieldType name="string" class="solr.StrField" sortMissingLast="true"/>
        <fieldType name="boolean" class="solr.BoolField" sortMissingLast="true"/>
        <fieldType name="int" class="solr.TrieIntField" precisionStep="0"
positionIncrementGap="0"/>
        <fieldType name="float" class="solr.TrieFloatField" precisionStep="0"
positionIncrementGap="0"/>
        <fieldType name="long" class="solr.TrieLongField" precisionStep="0"
positionIncrementGap="0"/>
        <fieldType name="double" class="solr.TrieDoubleField" precisionStep="0"
positionIncrementGap="0"/>


        <fieldType name="tint" class="solr.TrieIntField" precisionStep="8"
positionIncrementGap="0"/>
        <fieldType name="tfloat" class="solr.TrieFloatField" precisionStep="8"
positionIncrementGap="0"/>
        <fieldType name="tlong" class="solr.TrieLongField" precisionStep="8"
positionIncrementGap="0"/>
        <fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep="8"
positionIncrementGap="0"/>


        <fieldType name="date" class="solr.TrieDateField" precisionStep="0"
positionIncrementGap="0"/>

        <fieldType name="tdate" class="solr.TrieDateField" precisionStep="6"
positionIncrementGap="0"/>


        <!--Binary data type. The data should be sent/retrieved in as Base64 encoded
Strings -->
        <fieldtype name="binary" class="solr.BinaryField"/>


        <fieldType name="random" class="solr.RandomSortField" indexed="true"/>

        <fieldType name="text_ws" class="solr.TextField" positionIncrementGap="100">
            <analyzer>
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
            </analyzer>
        </fieldType>

        <fieldType name="text_general" class="solr.TextField"
positionIncrementGap="100">
            <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
```

```
words="stopwords.txt"/>
                <!-- in this example, we will only use synonyms at query time
                <filter class="solr.SynonymFilterFactory"
synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
                -->
                <filter class="solr.LowerCaseFilterFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.LowerCaseFilterFactory"/>
            </analyzer>
        </fieldType>

        <fieldType name="text_en" class="solr.TextField" positionIncrementGap="100">
            <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>

                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
                        />
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.EnglishPossessiveFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>

                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
                        />
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.EnglishPossessiveFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <!-- Optionally you may want to use this less aggressive stemmer
instead of PorterStemFilterFactory:
                    <filter class="solr.EnglishMinimalStemFilterFactory"/>
                -->
                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- A text field with defaults appropriate for English, plus
         aggressive word-splitting and autophrase features enabled.
         This field is just like text_en, except it adds
         WordDelimiterFilter to enable splitting and matching of
         words on case-change, alpha numeric boundaries, and
         non-alphanumeric chars.  This means certain compound word
         cases will work, for example query "wi fi" will match
         document "WiFi" or "wi-fi".
```

```
            -->
        <fieldType name="text_en_splitting" class="solr.TextField"
positionIncrementGap="100"
                   autoGeneratePhraseQueries="true">
            <analyzer type="index">
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <!-- in this example, we will only use synonyms at query time
                <filter class="solr.SynonymFilterFactory"
synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
                -->
                <!-- Case insensitive stop word removal.
                -->
                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
                        />
                <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1"
                        catenateWords="1" catenateNumbers="1" catenateAll="0"
splitOnCaseChange="1"/>
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
                        />
                <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1"
                        catenateWords="0" catenateNumbers="0" catenateAll="0"
splitOnCaseChange="1"/>
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- Less flexible matching, but less false matches.  Probably not ideal for
product names,
             but may be good for SKUs.  Can insert dashes in the wrong place and
still match. -->
        <fieldType name="text_en_splitting_tight" class="solr.TextField"
positionIncrementGap="100"
                   autoGeneratePhraseQueries="true">
            <analyzer>
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="false"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="lang/stopwords_en.txt"/>
                <filter class="solr.WordDelimiterFilterFactory" generateWordParts="0"
generateNumberParts="0"
                        catenateWords="1" catenateNumbers="1" catenateAll="0"/>
```

```
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <filter class="solr.EnglishMinimalStemFilterFactory"/>
                <!-- this filter can remove any duplicate tokens that appear at the
same position - sometimes
                     possible with WordDelimiterFilter in conjuncton with stemming. -
->
                <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- Just like text_general except it reverses the characters of
         each token, to enable more efficient leading wildcard queries. -->
        <fieldType name="text_general_rev" class="solr.TextField"
positionIncrementGap="100">
            <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt"/>
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.ReversedWildcardFilterFactory"
withOriginal="true"
                        maxPosAsterisk="3" maxPosQuestion="2"
maxFractionAsterisk="0.33"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt"/>
                <filter class="solr.LowerCaseFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- charFilter + WhitespaceTokenizer  -->
        <!--
        <fieldType name="text_char_norm" class="solr.TextField"
positionIncrementGap="100" >
          <analyzer>
            <charFilter class="solr.MappingCharFilterFactory" mapping="mapping-
ISOLatin1Accent.txt"/>
            <tokenizer class="solr.WhitespaceTokenizerFactory"/>
          </analyzer>
        </fieldType>
        -->

        <!-- This is an example of using the KeywordTokenizer along
             With various TokenFilterFactories to produce a sortable field
             that does not include some properties of the source text
          -->
        <fieldType name="alphaOnlySort" class="solr.TextField" sortMissingLast="true"
omitNorms="true">
            <analyzer>
                <!-- KeywordTokenizer does no actual tokenizing, so the entire
                     input string is preserved as a single token
                  -->
                <tokenizer class="solr.KeywordTokenizerFactory"/>
                <!-- The LowerCase TokenFilter does what you expect, which can be
```

```
                    when you want your sorting to be case insensitive
                 -->
               <filter class="solr.LowerCaseFilterFactory"/>
               <!-- The TrimFilter removes any leading or trailing whitespace -->
               <filter class="solr.TrimFilterFactory"/>
               <!-- The PatternReplaceFilter gives you the flexibility to use
                    Java Regular expression to replace any sequence of characters
                    matching a pattern with an arbitrary replacement string,
                    which may include back references to portions of the original
                    string matched by the pattern.

                    See the Java Regular Expression documentation for more
                    information on pattern and replacement string syntax.


http://docs.oracle.com/javase/7/docs/api/java/util/regex/package-summary.html
                 -->
               <filter class="solr.PatternReplaceFilterFactory"
                       pattern="([^a-z])" replacement="" replace="all"
                       />
            </analyzer>
        </fieldType>

        <fieldtype name="phonetic" stored="false" indexed="true"
class="solr.TextField">
            <analyzer>
               <tokenizer class="solr.StandardTokenizerFactory"/>
               <filter class="solr.DoubleMetaphoneFilterFactory" inject="false"/>
            </analyzer>
        </fieldtype>

        <fieldtype name="payloads" stored="false" indexed="true"
class="solr.TextField">
            <analyzer>
               <tokenizer class="solr.WhitespaceTokenizerFactory"/>
               <!--
               The DelimitedPayloadTokenFilter can put payloads on tokens... for
example,
               a token of "foo|1.4"  would be indexed as "foo" with a payload of
1.4f
               Attributes of the DelimitedPayloadTokenFilterFactory :
                "delimiter" - a one character delimiter. Default is | (pipe)
             "encoder" - how to encode the following value into a playload
               float -> org.apache.lucene.analysis.payloads.FloatEncoder,
               integer -> o.a.l.a.p.IntegerEncoder
               identity -> o.a.l.a.p.IdentityEncoder
                   Fully Qualified class name implementing PayloadEncoder, Encoder
must have a no arg constructor.
                -->
               <filter class="solr.DelimitedPayloadTokenFilterFactory"
encoder="float"/>
            </analyzer>
        </fieldtype>

        <!-- lowercases the entire field value, keeping it as a single token.  -->
        <fieldType name="lowercase" class="solr.TextField"
positionIncrementGap="100">
            <analyzer>
               <tokenizer class="solr.KeywordTokenizerFactory"/>
               <filter class="solr.LowerCaseFilterFactory"/>
```

```
            </analyzer>
        </fieldType>

        <!--
          Example of using PathHierarchyTokenizerFactory at index time, so
          queries for paths match documents at that path, or in descendent paths
        -->
        <fieldType name="descendent_path" class="solr.TextField">
            <analyzer type="index">
                <tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.KeywordTokenizerFactory"/>
            </analyzer>
        </fieldType>
        <!--
          Example of using PathHierarchyTokenizerFactory at query time, so
          queries for paths match documents at that path, or in ancestor paths
        -->
        <fieldType name="ancestor_path" class="solr.TextField">
            <analyzer type="index">
                <tokenizer class="solr.KeywordTokenizerFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/"/>
            </analyzer>
        </fieldType>

        <!-- since fields of this type are by default not stored or indexed,
             any data added to them will be ignored outright.  -->
        <fieldtype name="ignored" stored="false" indexed="false" multiValued="true"
class="solr.StrField"/>

        <!-- This point type indexes the coordinates as separate fields (subFields)
          If subFieldType is defined, it references a type, and a dynamic field
          definition is created matching *___<typename>.  Alternately, if
          subFieldSuffix is defined, that is used to create the subFields.
          Example: if subFieldType="double", then the coordinates would be
            indexed in fields myloc_0___double,myloc_1___double.
          Example: if subFieldSuffix="_d" then the coordinates would be indexed
            in fields myloc_0_d,myloc_1_d
          The subFields are an implementation detail of the fieldType, and end
          users normally should not need to know about them.
         -->
        <fieldType name="point" class="solr.PointType" dimension="2"
subFieldSuffix="_d"/>

        <!-- A specialized field for geospatial search. If indexed, this fieldType
must not be multivalued. -->
        <fieldType name="location" class="solr.LatLonType"
subFieldSuffix="_coordinate"/>

        <!-- An alternative geospatial field type new to Solr 4.  It supports
multiValued and polygon shapes.
          For more information about this and other Spatial fields new to Solr 4,
see:
          http://wiki.apache.org/solr/SolrAdaptersForLuceneSpatial4
        -->
        <fieldType name="location_rpt"
class="solr.SpatialRecursivePrefixTreeFieldType"
```

```
                    geo="true" distErrPct="0.025" maxDistErr="0.000009"
 units="degrees"/>

    </types>

 </schema>
```

B. Solr Schema.xml - for Webpages

```
<?xml version="1.0" encoding="UTF-8" ?>

<schema name="IDEAL_webpages" version="0.1">

    <fields>

        <field name="id" type="string" indexed="true" stored="true" required="true"
multiValued="false"/>

        <field name="collection" type="string" indexed="true" stored="true"
required="true" multiValued="false"/>

        <field name="title" type="text_general" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="domain" type="string" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="url" type="string" indexed="true" stored="true" required="true"
multiValued="false"/>

        <field name="text" type="text_general" indexed="true" stored="true"
required="false" multiValued="false"/>

        <!-- not used yet -->
        <field name="author" type="text_general" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="subtitle" type="text_general" indexed="true" stored="true"
required="false" multiValued="true"/>

        <field name="created_at" type="tdate" indexed="true" stored="true"
required="false" multiValued="false"/>

        <field name="section" type="text_general" indexed="true" stored="true"
required="false" multiValued="true"/>


        <!-- class analysis fields -->
        <field name="ner_people" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="ner_locations" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="ner_dates" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="ner_organizations" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
```

```
        <field name="cluster_id" type="string" indexed="false" stored="true"
required="false" multiValued="false"/>
        <field name="cluster_label" type="string" indexed="false" stored="true"
required="false" multiValued="false"/>

        <field name="classification_vector_json" type="string" indexed="false"
stored="true" required="false" multiValued="false"/>
        <field name="classification_labels" type="string" indexed="false"
stored="true" required="false" multiValued="true"/>

        <field name="social_vector_json" type="string" indexed="false" stored="true"
required="false" multiValued="false"/>

        <field name="social_importance" type="double" indexed="false" stored="true"
required="false" multiValued="false"/>

        <field name="lda_dict_json" type="string" indexed="false" stored="true"
required="false" multiValued="false"/>

        <field name="lda_topics" type="string" indexed="false" stored="true"
required="false" multiValued="true"/>

         <field name="ner_people_multiple" type="string" indexed="false"
stored="false" required="false"
                multiValued="false"/>

         <field name="ner_locations_multiple" type="string" indexed="false"
stored="false" required="false"
                multiValued="false"/>

        <field name="ner_dates_multiple" type="string" indexed="false" stored="false"
required="false"
                multiValued="false"/>

        <field name="ner_organizations_multiple" type="string" indexed="false"
stored="false" required="false"
                multiValued="false"/>

        <field name="lda_topics_multiple" type="string" indexed="false" stored="false"
required="false"
                multiValued="false"/>

        <field name="classification_labels_multiple" type="string" indexed="false"
stored="false" required="false"
                multiValued="false"/>

        <!-- The following fields are optional and to be removed if not applicable. --
>
        <field name="twitter_link" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="facebook_link" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="google_plus_link" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>
        <field name="pinterest" type="string" indexed="true" stored="true"
required="false" multiValued="true"/>

        <!-- The following fields are not to be proposed in the final draft and are
here for discussion sake only. -->
        <field name="coordinates" type="string" indexed="true" stored="true"
```

```
required="false" multiValued="true"/>

        <field name="_version_" type="tlong" indexed="true" stored="true"
multiValued="false"/>

        <dynamicField name="*_i" type="int" indexed="true" stored="true"/>
        <dynamicField name="*_is" type="int" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_s" type="string" indexed="true" stored="true"/>
        <dynamicField name="*_ss" type="string" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_l" type="long" indexed="true" stored="true"/>
        <dynamicField name="*_ls" type="long" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_t" type="text_general" indexed="true" stored="true"/>
        <dynamicField name="*_txt" type="text_general" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_en" type="text_en" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_b" type="boolean" indexed="true" stored="true"/>
        <dynamicField name="*_bs" type="boolean" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_f" type="float" indexed="true" stored="true"/>
        <dynamicField name="*_fs" type="float" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_d" type="double" indexed="true" stored="true"/>
        <dynamicField name="*_ds" type="double" indexed="true" stored="true"
multiValued="true"/>

        <!-- Type used to index the lat and lon components for the "location"
FieldType -->
        <dynamicField name="*_coordinate" type="tdouble" indexed="true"
stored="false"/>

        <dynamicField name="*_dt" type="date" indexed="true" stored="true"/>
        <dynamicField name="*_dts" type="date" indexed="true" stored="true"
multiValued="true"/>
        <dynamicField name="*_p" type="location" indexed="true" stored="true"/>

        <!-- some trie-coded dynamic fields for faster range queries -->
        <dynamicField name="*_ti" type="tint" indexed="true" stored="true"/>
        <dynamicField name="*_tl" type="tlong" indexed="true" stored="true"/>
        <dynamicField name="*_tf" type="tfloat" indexed="true" stored="true"/>
        <dynamicField name="*_td" type="tdouble" indexed="true" stored="true"/>
        <dynamicField name="*_tdt" type="tdate" indexed="true" stored="true"/>

        <dynamicField name="ignored_*" type="ignored" multiValued="true"/>
        <dynamicField name="attr_*" type="text_general" indexed="true" stored="true"
multiValued="true"/>

        <dynamicField name="random_*" type="random"/>

    </fields>

    <uniqueKey>id</uniqueKey>

    <types>

        <fieldType name="string" class="solr.StrField" sortMissingLast="true"/>
        <fieldType name="boolean" class="solr.BoolField" sortMissingLast="true"/>
```

```
        <fieldType name="int" class="solr.TrieIntField" precisionStep="0"
positionIncrementGap="0"/>
        <fieldType name="float" class="solr.TrieFloatField" precisionStep="0"
positionIncrementGap="0"/>
        <fieldType name="long" class="solr.TrieLongField" precisionStep="0"
positionIncrementGap="0"/>
        <fieldType name="double" class="solr.TrieDoubleField" precisionStep="0"
positionIncrementGap="0"/>


        <fieldType name="tint" class="solr.TrieIntField" precisionStep="8"
positionIncrementGap="0"/>
        <fieldType name="tfloat" class="solr.TrieFloatField" precisionStep="8"
positionIncrementGap="0"/>
        <fieldType name="tlong" class="solr.TrieLongField" precisionStep="8"
positionIncrementGap="0"/>
        <fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep="8"
positionIncrementGap="0"/>


        <fieldType name="date" class="solr.TrieDateField" precisionStep="0"
positionIncrementGap="0"/>

        <fieldType name="tdate" class="solr.TrieDateField" precisionStep="6"
positionIncrementGap="0"/>


        <!--Binary data type. The data should be sent/retrieved in as Base64 encoded
Strings -->
        <fieldtype name="binary" class="solr.BinaryField"/>


        <fieldType name="random" class="solr.RandomSortField" indexed="true"/>

        <fieldType name="text_ws" class="solr.TextField" positionIncrementGap="100">
            <analyzer>
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
            </analyzer>
        </fieldType>

        <fieldType name="text_general" class="solr.TextField"
positionIncrementGap="100">
            <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt"/>
                <!-- in this example, we will only use synonyms at query time
                <filter class="solr.SynonymFilterFactory"
synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
                -->
                <filter class="solr.LowerCaseFilterFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.LowerCaseFilterFactory"/>
            </analyzer>
```

```
        </fieldType>

        <fieldType name="text_en" class="solr.TextField" positionIncrementGap="100">
            <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>

                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
                        />
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.EnglishPossessiveFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>

                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
                        />
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.EnglishPossessiveFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <!-- Optionally you may want to use this less aggressive stemmer
instead of PorterStemFilterFactory:
                    <filter class="solr.EnglishMinimalStemFilterFactory"/>
                -->
                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- A text field with defaults appropriate for English, plus
         aggressive word-splitting and autophrase features enabled.
         This field is just like text_en, except it adds
         WordDelimiterFilter to enable splitting and matching of
         words on case-change, alpha numeric boundaries, and
         non-alphanumeric chars.  This means certain compound word
         cases will work, for example query "wi fi" will match
         document "WiFi" or "wi-fi".
            -->
        <fieldType name="text_en_splitting" class="solr.TextField"
positionIncrementGap="100"
                   autoGeneratePhraseQueries="true">
            <analyzer type="index">
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <!-- in this example, we will only use synonyms at query time
                <filter class="solr.SynonymFilterFactory"
synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
                -->
                <!-- Case insensitive stop word removal.
                -->
                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
```

```
                        />
                <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1"
                        catenateWords="1" catenateNumbers="1" catenateAll="0"
splitOnCaseChange="1"/>
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory"
                        ignoreCase="true"
                        words="lang/stopwords_en.txt"
                        />
                <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1"
                        catenateWords="0" catenateNumbers="0" catenateAll="0"
splitOnCaseChange="1"/>
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <filter class="solr.PorterStemFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- Less flexible matching, but less false matches.  Probably not ideal for
product names,
             but may be good for SKUs.  Can insert dashes in the wrong place and still
match. -->
        <fieldType name="text_en_splitting_tight" class="solr.TextField"
positionIncrementGap="100"
                   autoGeneratePhraseQueries="true">
            <analyzer>
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="false"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="lang/stopwords_en.txt"/>
                <filter class="solr.WordDelimiterFilterFactory" generateWordParts="0"
generateNumberParts="0"
                        catenateWords="1" catenateNumbers="1" catenateAll="0"/>
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
                <filter class="solr.EnglishMinimalStemFilterFactory"/>
                <!-- this filter can remove any duplicate tokens that appear at the
same position - sometimes
                     possible with WordDelimiterFilter in conjuncton with stemming. --
>
                <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- Just like text_general except it reverses the characters of
         each token, to enable more efficient leading wildcard queries. -->
        <fieldType name="text_general_rev" class="solr.TextField"
```

```
positionIncrementGap="100">
            <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt"/>
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.ReversedWildcardFilterFactory" withOriginal="true"
                        maxPosAsterisk="3" maxPosQuestion="2"
maxFractionAsterisk="0.33"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt"/>
                <filter class="solr.LowerCaseFilterFactory"/>
            </analyzer>
        </fieldType>

        <!-- charFilter + WhitespaceTokenizer  -->
        <!--
        <fieldType name="text_char_norm" class="solr.TextField"
positionIncrementGap="100" >
          <analyzer>
            <charFilter class="solr.MappingCharFilterFactory" mapping="mapping-
ISOLatin1Accent.txt"/>
            <tokenizer class="solr.WhitespaceTokenizerFactory"/>
          </analyzer>
        </fieldType>
        -->

        <!-- This is an example of using the KeywordTokenizer along
             With various TokenFilterFactories to produce a sortable field
             that does not include some properties of the source text
          -->
        <fieldType name="alphaOnlySort" class="solr.TextField" sortMissingLast="true"
omitNorms="true">
            <analyzer>
                <!-- KeywordTokenizer does no actual tokenizing, so the entire
                     input string is preserved as a single token
                  -->
                <tokenizer class="solr.KeywordTokenizerFactory"/>
                <!-- The LowerCase TokenFilter does what you expect, which can be
                     when you want your sorting to be case insensitive
                  -->
                <filter class="solr.LowerCaseFilterFactory"/>
                <!-- The TrimFilter removes any leading or trailing whitespace -->
                <filter class="solr.TrimFilterFactory"/>
                <!-- The PatternReplaceFilter gives you the flexibility to use
                     Java Regular expression to replace any sequence of characters
                     matching a pattern with an arbitrary replacement string,
                     which may include back references to portions of the original
                     string matched by the pattern.

                     See the Java Regular Expression documentation for more
                     information on pattern and replacement string syntax.

                     http://docs.oracle.com/javase/7/docs/api/java/util/regex/package-
summary.html
```

```
                -->
            <filter class="solr.PatternReplaceFilterFactory"
                    pattern="([^a-z])" replacement="" replace="all"
                    />
        </analyzer>
    </fieldType>

    <fieldtype name="phonetic" stored="false" indexed="true"
class="solr.TextField">
        <analyzer>
            <tokenizer class="solr.StandardTokenizerFactory"/>
            <filter class="solr.DoubleMetaphoneFilterFactory" inject="false"/>
        </analyzer>
    </fieldtype>

    <fieldtype name="payloads" stored="false" indexed="true"
class="solr.TextField">
        <analyzer>
            <tokenizer class="solr.WhitespaceTokenizerFactory"/>
            <!--
            The DelimitedPayloadTokenFilter can put payloads on tokens... for
example,
            a token of "foo|1.4"  would be indexed as "foo" with a payload of 1.4f
            Attributes of the DelimitedPayloadTokenFilterFactory :
             "delimiter" - a one character delimiter. Default is | (pipe)
         "encoder" - how to encode the following value into a playload
            float -> org.apache.lucene.analysis.payloads.FloatEncoder,
            integer -> o.a.l.a.p.IntegerEncoder
            identity -> o.a.l.a.p.IdentityEncoder
                Fully Qualified class name implementing PayloadEncoder, Encoder
must have a no arg constructor.
             -->
            <filter class="solr.DelimitedPayloadTokenFilterFactory"
encoder="float"/>
        </analyzer>
    </fieldtype>

    <!-- lowercases the entire field value, keeping it as a single token.  -->
    <fieldType name="lowercase" class="solr.TextField" positionIncrementGap="100">
        <analyzer>
            <tokenizer class="solr.KeywordTokenizerFactory"/>
            <filter class="solr.LowerCaseFilterFactory"/>
        </analyzer>
    </fieldType>

    <!--
      Example of using PathHierarchyTokenizerFactory at index time, so
      queries for paths match documents at that path, or in descendent paths
    -->
    <fieldType name="descendent_path" class="solr.TextField">
        <analyzer type="index">
            <tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/"/>
        </analyzer>
        <analyzer type="query">
            <tokenizer class="solr.KeywordTokenizerFactory"/>
        </analyzer>
    </fieldType>
    <!--
      Example of using PathHierarchyTokenizerFactory at query time, so
      queries for paths match documents at that path, or in ancestor paths
```

```
        -->
        <fieldType name="ancestor_path" class="solr.TextField">
            <analyzer type="index">
                <tokenizer class="solr.KeywordTokenizerFactory"/>
            </analyzer>
            <analyzer type="query">
                <tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/"/>
            </analyzer>
        </fieldType>

        <!-- since fields of this type are by default not stored or indexed,
             any data added to them will be ignored outright.  -->
        <fieldtype name="ignored" stored="false" indexed="false" multiValued="true"
class="solr.StrField"/>

        <!-- This point type indexes the coordinates as separate fields (subFields)
          If subFieldType is defined, it references a type, and a dynamic field
          definition is created matching *___<typename>.  Alternately, if
          subFieldSuffix is defined, that is used to create the subFields.
          Example: if subFieldType="double", then the coordinates would be
            indexed in fields myloc_0___double,myloc_1___double.
          Example: if subFieldSuffix="_d" then the coordinates would be indexed
            in fields myloc_0_d,myloc_1_d
          The subFields are an implementation detail of the fieldType, and end
          users normally should not need to know about them.
         -->
        <fieldType name="point" class="solr.PointType" dimension="2"
subFieldSuffix="_d"/>

        <!-- A specialized field for geospatial search. If indexed, this fieldType
must not be multivalued. -->
        <fieldType name="location" class="solr.LatLonType"
subFieldSuffix="_coordinate"/>

        <!-- An alternative geospatial field type new to Solr 4.  It supports
multiValued and polygon shapes.
          For more information about this and other Spatial fields new to Solr 4, see:
          http://wiki.apache.org/solr/SolrAdaptersForLuceneSpatial4
        -->
        <fieldType name="location_rpt"
class="solr.SpatialRecursivePrefixTreeFieldType"
                   geo="true" distErrPct="0.025" maxDistErr="0.000009"
units="degrees"/>


    </types>

</schema>
```

C. SolrConfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
 Licensed to the Apache Software Foundation (ASF) under one or more
 contributor license agreements.  See the NOTICE file distributed with
 this work for additional information regarding copyright ownership.
 The ASF licenses this file to You under the Apache License, Version 2.0
 (the "License"); you may not use this file except in compliance with
```

```
 the License.  You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
-->

<!--
     For more details about configurations options that may appear in
     this file, see http://wiki.apache.org/solr/SolrConfigXml.
-->
<config>
    <!-- In all configuration below, a prefix of "solr." for class names
         is an alias that causes solr to search appropriate packages,
         including org.apache.solr.(search|update|request|core|analysis)

         You may also specify a fully qualified Java classname if you
         have your own custom plugins.
      -->

    <!-- Controls what version of Lucene various components of Solr
         adhere to.  Generally, you want to use the latest version to
         get all bug fixes and improvements. It is highly recommended
         that you fully re-index after changing this setting as it can
         affect both how text is indexed and queried.
    -->
    <luceneMatchVersion>4.10.3</luceneMatchVersion>

    <!-- <lib/> directives can be used to instruct Solr to load any Jars
         identified and use them to resolve any "plugins" specified in
         your solrconfig.xml or schema.xml (ie: Analyzers, Request
         Handlers, etc...).

         All directories and paths are resolved relative to the
         instanceDir.

         Please note that <lib/> directives are processed in the order
         that they appear in your solrconfig.xml file, and are "stacked"
         on top of each other when building a ClassLoader - so if you have
         plugin jars with dependencies on other jars, the "lower level"
         dependency jars should be loaded first.

         If a "./lib" directory exists in your instanceDir, all files
         found in it are included as if you had used the following
         syntax...

                <lib dir="./lib" />
      -->

    <!-- A 'dir' option by itself adds any files found in the directory
         to the classpath, this is useful for including all jars in a
         directory.

         When a 'regex' is specified in addition to a 'dir', only the
         files in that directory which completely match the regex
         (anchored on both ends) will be included.
```

```
        If a 'dir' option (with or without a regex) is used and nothing
        is found that matches, a warning will be logged.

        The examples below can be used to load some solr-contribs along
        with their external dependencies.
      -->
    <lib dir="${solr.install.dir:../../..}/contrib/extraction/lib" regex=".*\.jar"/>
    <lib dir="${solr.install.dir:../../..}/dist/" regex="solr-cell-\d.*\.jar"/>

    <lib dir="${solr.install.dir:../../..}/contrib/clustering/lib/" regex=".*\.jar"/>
    <lib dir="${solr.install.dir:../../..}/dist/" regex="solr-clustering-\d.*\.jar"/>

    <lib dir="${solr.install.dir:../../..}/contrib/langid/lib/" regex=".*\.jar"/>
    <lib dir="${solr.install.dir:../../..}/dist/" regex="solr-langid-\d.*\.jar"/>

    <lib dir="${solr.install.dir:../../..}/contrib/velocity/lib" regex=".*\.jar"/>
    <lib dir="${solr.install.dir:../../..}/dist/" regex="solr-velocity-\d.*\.jar"/>

    <!-- an exact 'path' can be used instead of a 'dir' to specify a
         specific jar file.  This will cause a serious error to be logged
         if it can't be loaded.
      -->
    <!--
       <lib path="../a-jar-that-does-not-exist.jar" />
    -->

    <!-- Data Directory

         Used to specify an alternate directory to hold all index data
         other than the default ./data under the Solr home.  If
         replication is in use, this should match the replication
         configuration.
      -->
    <dataDir>${solr.data.dir:}</dataDir>


    <!-- The DirectoryFactory to use for indexes.

         solr.StandardDirectoryFactory is filesystem
         based and tries to pick the best implementation for the current
         JVM and platform.  solr.NRTCachingDirectoryFactory, the default,
         wraps solr.StandardDirectoryFactory and caches small files in memory
         for better NRT performance.

         One can force a particular implementation via solr.MMapDirectoryFactory,
         solr.NIOFSDirectoryFactory, or solr.SimpleFSDirectoryFactory.

         solr.RAMDirectoryFactory is memory based, not
         persistent, and doesn't work with replication.
      -->
    <directoryFactory name="DirectoryFactory"

class="${solr.directoryFactory:solr.NRTCachingDirectoryFactory}">


        <!-- These will be used if you are using the solr.HdfsDirectoryFactory,
             otherwise they will be ignored. If you don't plan on using hdfs,
             you can safely remove this section. -->
        <!-- The root directory that collection data should be written to. -->
```

```
        <str name="solr.hdfs.home">${solr.hdfs.home:}</str>
        <!-- The hadoop configuration files to use for the hdfs client. -->
        <str name="solr.hdfs.confdir">${solr.hdfs.confdir:}</str>
        <!-- Enable/Disable the hdfs cache. -->
        <str
name="solr.hdfs.blockcache.enabled">${solr.hdfs.blockcache.enabled:true}</str>
        <!-- Enable/Disable using one global cache for all SolrCores.
             The settings used will be from the first HdfsDirectoryFactory created. --
>
        <str
name="solr.hdfs.blockcache.global">${solr.hdfs.blockcache.global:true}</str>

    </directoryFactory>

    <!-- The CodecFactory for defining the format of the inverted index.
         The default implementation is SchemaCodecFactory, which is the official
Lucene
         index format, but hooks into the schema to provide per-field customization of
         the postings lists and per-document values in the fieldType element
         (postingsFormat/docValuesFormat). Note that most of the alternative
implementations
         are experimental, so if you choose to customize the index format, its a good
         idea to convert back to the official format e.g. via
IndexWriter.addIndexes(IndexReader)
         before upgrading to a newer version to avoid unnecessary reindexing.
    -->
    <codecFactory class="solr.SchemaCodecFactory"/>

    <!-- To enable dynamic schema REST APIs, use the following for <schemaFactory>:

         <schemaFactory class="ManagedIndexSchemaFactory">
           <bool name="mutable">true</bool>
           <str name="managedSchemaResourceName">managed-schema</str>
         </schemaFactory>

         When ManagedIndexSchemaFactory is specified, Solr will load the schema from
         he resource named in 'managedSchemaResourceName', rather than from
schema.xml.
         Note that the managed schema resource CANNOT be named schema.xml.  If the
managed
         schema does not exist, Solr will create it after reading schema.xml, then
rename
         'schema.xml' to 'schema.xml.bak'.

         Do NOT hand edit the managed schema - external modifications will be ignored
and
         overwritten as a result of schema modification REST API calls.

         When ManagedIndexSchemaFactory is specified with mutable = true, schema
         modification REST API calls will be allowed; otherwise, error responses will
be
         sent back for these requests.
    -->
    <schemaFactory class="ClassicIndexSchemaFactory"/>

    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
         Index Config - These settings control low-level behavior of indexing
         Most example settings here show the default value, but are commented
         out, to more easily see where customizations have been made.
```

```
        Note: This replaces <indexDefaults> and <mainIndex> from older versions
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <indexConfig>
        <!-- maxFieldLength was removed in 4.0. To get similar behavior, include a
             LimitTokenCountFilterFactory in your fieldType definition. E.g.
         <filter class="solr.LimitTokenCountFilterFactory" maxTokenCount="10000"/>
        -->
        <!-- Maximum time to wait for a write lock (ms) for an IndexWriter. Default:
1000 -->
        <!-- <writeLockTimeout>1000</writeLockTimeout>  -->

        <!-- The maximum number of simultaneous threads that may be
             indexing documents at once in IndexWriter; if more than this
             many threads arrive they will wait for others to finish.
             Default in Solr/Lucene is 8. -->
        <!-- <maxIndexingThreads>8</maxIndexingThreads>  -->

        <!-- Expert: Enabling compound file will use less files for the index,
             using fewer file descriptors on the expense of performance decrease.
             Default in Lucene is "true". Default in Solr is "false" (since 3.6) -->
        <!-- <useCompoundFile>false</useCompoundFile> -->

        <!-- ramBufferSizeMB sets the amount of RAM that may be used by Lucene
             indexing for buffering added documents and deletions before they are
             flushed to the Directory.
             maxBufferedDocs sets a limit on the number of documents buffered
             before flushing.
             If both ramBufferSizeMB and maxBufferedDocs is set, then
             Lucene will flush based on whichever limit is hit first.
             The default is 100 MB.  -->
        <!-- <ramBufferSizeMB>100</ramBufferSizeMB> -->
        <!-- <maxBufferedDocs>1000</maxBufferedDocs> -->

        <!-- Expert: Merge Policy
             The Merge Policy in Lucene controls how merging of segments is done.
             The default since Solr/Lucene 3.3 is TieredMergePolicy.
             The default since Lucene 2.3 was the LogByteSizeMergePolicy,
             Even older versions of Lucene used LogDocMergePolicy.
          -->
        <!--
            <mergePolicy class="org.apache.lucene.index.TieredMergePolicy">
              <int name="maxMergeAtOnce">10</int>
              <int name="segmentsPerTier">10</int>
            </mergePolicy>
          -->

        <!-- Merge Factor
             The merge factor controls how many segments will get merged at a time.
             For TieredMergePolicy, mergeFactor is a convenience parameter which
             will set both MaxMergeAtOnce and SegmentsPerTier at once.
             For LogByteSizeMergePolicy, mergeFactor decides how many new segments
             will be allowed before they are merged into one.
             Default is 10 for both merge policies.
          -->
        <!--
        <mergeFactor>10</mergeFactor>
          -->

        <!-- Expert: Merge Scheduler
             The Merge Scheduler in Lucene controls how merges are
```

```
     performed.  The ConcurrentMergeScheduler (Lucene 2.3 default)
     can perform merges in the background using separate threads.
     The SerialMergeScheduler (Lucene 2.2 default) does not.
 -->
<!--
   <mergeScheduler class="org.apache.lucene.index.ConcurrentMergeScheduler"/>
   -->

<!-- LockFactory

     This option specifies which Lucene LockFactory implementation
     to use.

     single = SingleInstanceLockFactory - suggested for a
              read-only index or when there is no possibility of
              another process trying to modify the index.
     native = NativeFSLockFactory - uses OS native file locking.
              Do not use when multiple solr webapps in the same
              JVM are attempting to share a single index.
     simple = SimpleFSLockFactory  - uses a plain file for locking

     Defaults: 'native' is default for Solr3.6 and later, otherwise
               'simple' is the default

     More details on the nuances of each LockFactory...
     http://wiki.apache.org/lucene-java/AvailableLockFactories
-->
<lockType>${solr.lock.type:native}</lockType>

<!-- Unlock On Startup

     If true, unlock any held write or commit locks on startup.
     This defeats the locking mechanism that allows multiple
     processes to safely access a lucene index, and should be used
     with care. Default is "false".

     This is not needed if lock type is 'single'
 -->
<!--
<unlockOnStartup>false</unlockOnStartup>
  -->

<!-- Expert: Controls how often Lucene loads terms into memory
     Default is 128 and is likely good for most everyone.
  -->
<!-- <termIndexInterval>128</termIndexInterval> -->

<!-- If true, IndexReaders will be opened/reopened from the IndexWriter
     instead of from the Directory. Hosts in a master/slave setup
     should have this set to false while those in a SolrCloud
     cluster need to be set to true. Default: true
  -->
<!--
<nrtMode>true</nrtMode>
  -->

<!-- Commit Deletion Policy
     Custom deletion policies can be specified here. The class must
     implement org.apache.lucene.index.IndexDeletionPolicy.
```

```
          The default Solr IndexDeletionPolicy implementation supports
          deleting index commit points on number of commits, age of
          commit point and optimized status.

          The latest commit point should always be preserved regardless
          of the criteria.
    -->
    <!--
    <deletionPolicy class="solr.SolrDeletionPolicy">
    -->
    <!-- The number of commit points to be kept -->
    <!-- <str name="maxCommitsToKeep">1</str> -->
    <!-- The number of optimized commit points to be kept -->
    <!-- <str name="maxOptimizedCommitsToKeep">0</str> -->
    <!--
        Delete all commit points once they have reached the given age.
        Supports DateMathParser syntax e.g.
      -->
    <!--
       <str name="maxCommitAge">30MINUTES</str>
       <str name="maxCommitAge">1DAY</str>
    -->
    <!--
    </deletionPolicy>
    -->

    <!-- Lucene Infostream

          To aid in advanced debugging, Lucene provides an "InfoStream"
          of detailed information when indexing.

          Setting the value to true will instruct the underlying Lucene
          IndexWriter to write its info stream to solr's log. By default,
          this is enabled here, and controlled through log4j.properties.
      -->
    <infoStream>true</infoStream>

    <!--
        Use true to enable this safety check, which can help
        reduce the risk of propagating index corruption from older segments
        into new ones, at the expense of slower merging.
    -->
    <checkIntegrityAtMerge>false</checkIntegrityAtMerge>
</indexConfig>


<!-- JMX

     This example enables JMX if and only if an existing MBeanServer
     is found, use this if you want to configure JMX through JVM
     parameters. Remove this to disable exposing Solr configuration
     and statistics to JMX.

     For more details see http://wiki.apache.org/solr/SolrJmx
  -->
<jmx/>
<!-- If you want to connect to a particular server, specify the
     agentId
  -->
<!-- <jmx agentId="myAgent" /> -->
```

```
<!-- If you want to start a new MBeanServer, specify the serviceUrl -->
<!-- <jmx serviceUrl="service:jmx:rmi:///jndi/rmi://localhost:9999/solr"/>
  -->

<!-- The default high-performance update handler -->
<updateHandler class="solr.DirectUpdateHandler2">

    <!-- Enables a transaction log, used for real-time get, durability, and
         and solr cloud replica recovery.  The log can grow as big as
         uncommitted changes to the index, so use of a hard autoCommit
         is recommended (see below).
         "dir" - the target directory for transaction logs, defaults to the
                solr data directory.  -->
    <updateLog>
        <str name="dir">${solr.ulog.dir:}</str>
    </updateLog>

    <!-- AutoCommit

         Perform a hard commit automatically under certain conditions.
         Instead of enabling autoCommit, consider using "commitWithin"
         when adding documents.

         http://wiki.apache.org/solr/UpdateXmlMessages

         maxDocs - Maximum number of documents to add since the last
                   commit before automatically triggering a new commit.

         maxTime - Maximum amount of time in ms that is allowed to pass
                   since a document was added before automatically
                   triggering a new commit.
         openSearcher - if false, the commit causes recent index changes
           to be flushed to stable storage, but does not cause a new
           searcher to be opened to make those changes visible.

         If the updateLog is enabled, then it's highly recommended to
         have some sort of hard autoCommit to limit the log size.
      -->
    <autoCommit>
        <maxTime>${solr.autoCommit.maxTime:15000}</maxTime>
        <openSearcher>false</openSearcher>
    </autoCommit>

    <!-- softAutoCommit is like autoCommit except it causes a
         'soft' commit which only ensures that changes are visible
         but does not ensure that data is synced to disk.  This is
         faster and more near-realtime friendly than a hard commit.
      -->

    <autoSoftCommit>
        <maxTime>${solr.autoSoftCommit.maxTime:-1}</maxTime>
    </autoSoftCommit>

    <!-- Update Related Event Listeners

         Various IndexWriter related events can trigger Listeners to
         take actions.

         postCommit - fired after every commit or optimize command
         postOptimize - fired after every optimize command
```

```
      -->
    <!-- The RunExecutableListener executes an external command from a
         hook such as postCommit or postOptimize.

         exe - the name of the executable to run
         dir - dir to use as the current working directory. (default=".")
         wait - the calling thread waits until the executable returns.
                (default="true")
         args - the arguments to pass to the program.  (default is none)
         env - environment variables to set.  (default is none)
      -->
    <!-- This example shows how RunExecutableListener could be used
         with the script based replication...
         http://wiki.apache.org/solr/CollectionDistribution
      -->
    <!--
       <listener event="postCommit" class="solr.RunExecutableListener">
         <str name="exe">solr/bin/snapshooter</str>
         <str name="dir">.</str>
         <bool name="wait">true</bool>
         <arr name="args"> <str>arg1</str> <str>arg2</str> </arr>
         <arr name="env"> <str>MYVAR=val1</str> </arr>
       </listener>
      -->

</updateHandler>

<!-- IndexReaderFactory

     Use the following format to specify a custom IndexReaderFactory,
     which allows for alternate IndexReader implementations.

     ** Experimental Feature **

     Please note - Using a custom IndexReaderFactory may prevent
     certain other features from working. The API to
     IndexReaderFactory may change without warning or may even be
     removed from future releases if the problems cannot be
     resolved.


     ** Features that may not work with custom IndexReaderFactory **

     The ReplicationHandler assumes a disk-resident index. Using a
     custom IndexReader implementation may cause incompatibility
     with ReplicationHandler and may cause replication to not work
     correctly. See SOLR-1366 for details.

  -->
<!--
<indexReaderFactory name="IndexReaderFactory" class="package.class">
  <str name="someArg">Some Value</str>
</indexReaderFactory >
-->
<!-- By explicitly declaring the Factory, the termIndexDivisor can
     be specified.
  -->
<!--
   <indexReaderFactory name="IndexReaderFactory"
                       class="solr.StandardIndexReaderFactory">
```

```
      <int name="setTermIndexDivisor">12</int>
    </indexReaderFactory >
   -->

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     Query section - these settings control query time things like caches
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<query>
    <!-- Max Boolean Clauses

         Maximum number of clauses in each BooleanQuery,  an exception
         is thrown if exceeded.

         ** WARNING **

         This option actually modifies a global Lucene property that
         will affect all SolrCores.  If multiple solrconfig.xml files
         disagree on this property, the value at any given moment will
         be based on the last SolrCore to be initialized.

      -->
    <maxBooleanClauses>1024</maxBooleanClauses>


    <!-- Solr Internal Query Caches

         There are two implementations of cache available for Solr,
         LRUCache, based on a synchronized LinkedHashMap, and
         FastLRUCache, based on a ConcurrentHashMap.

         FastLRUCache has faster gets and slower puts in single
         threaded operation and thus is generally faster than LRUCache
         when the hit ratio of the cache is high (> 75%), and may be
         faster under other scenarios on multi-cpu systems.
    -->

    <!-- Filter Cache

         Cache used by SolrIndexSearcher for filters (DocSets),
         unordered sets of *all* documents that match a query.  When a
         new searcher is opened, its caches may be prepopulated or
         "autowarmed" using data from caches in the old searcher.
         autowarmCount is the number of items to prepopulate.  For
         LRUCache, the autowarmed items will be the most recently
         accessed items.

         Parameters:
           class - the SolrCache implementation LRUCache or
               (LRUCache or FastLRUCache)
           size - the maximum number of entries in the cache
           initialSize - the initial capacity (number of entries) of
               the cache.  (see java.util.HashMap)
           autowarmCount - the number of entries to prepopulate from
               and old cache.
      -->
    <filterCache class="solr.FastLRUCache"
                 size="512"
                 initialSize="512"
                 autowarmCount="0"/>
```

```
    <!-- Query Result Cache

         Caches results of searches - ordered lists of document ids
         (DocList) based on a query, a sort, and the range of documents requested.
      -->
    <queryResultCache class="solr.LRUCache"
                      size="512"
                      initialSize="512"
                      autowarmCount="0"/>

    <!-- Document Cache

         Caches Lucene Document objects (the stored fields for each
         document).  Since Lucene internal document ids are transient,
         this cache will not be autowarmed.
      -->
    <documentCache class="solr.LRUCache"
                   size="512"
                   initialSize="512"
                   autowarmCount="0"/>

    <!-- custom cache currently used by block join -->
    <cache name="perSegFilter"
           class="solr.search.LRUCache"
           size="10"
           initialSize="0"
           autowarmCount="10"
           regenerator="solr.NoOpRegenerator"/>

    <!-- Field Value Cache

         Cache used to hold field values that are quickly accessible
         by document id.  The fieldValueCache is created by default
         even if not configured here.
      -->
    <!--
       <fieldValueCache class="solr.FastLRUCache"
                        size="512"
                        autowarmCount="128"
                        showItems="32" />
      -->

    <!-- Custom Cache

         Example of a generic cache.  These caches may be accessed by
         name through SolrIndexSearcher.getCache(),cacheLookup(), and
         cacheInsert().  The purpose is to enable easy caching of
         user/application level data.  The regenerator argument should
         be specified as an implementation of solr.CacheRegenerator
         if autowarming is desired.
      -->
    <!--
       <cache name="myUserCache"
              class="solr.LRUCache"
              size="4096"
              initialSize="1024"
              autowarmCount="1024"
              regenerator="com.mycompany.MyRegenerator"
              />
      -->
```

```
<!-- Lazy Field Loading

     If true, stored fields that are not requested will be loaded
     lazily.  This can result in a significant speed improvement
     if the usual case is to not load all stored fields,
     especially if the skipped fields are large compressed text
     fields.
-->
<enableLazyFieldLoading>true</enableLazyFieldLoading>

<!-- Use Filter For Sorted Query

     A possible optimization that attempts to use a filter to
     satisfy a search.  If the requested sort does not include
     score, then the filterCache will be checked for a filter
     matching the query. If found, the filter will be used as the
     source of document ids, and then the sort will be applied to
     that.

     For most situations, this will not be useful unless you
     frequently get the same search repeatedly with different sort
     options, and none of them ever use "score"
  -->
<!--
   <useFilterForSortedQuery>true</useFilterForSortedQuery>
  -->

<!-- Result Window Size

     An optimization for use with the queryResultCache.  When a search
     is requested, a superset of the requested number of document ids
     are collected.  For example, if a search for a particular query
     requests matching documents 10 through 19, and queryWindowSize is 50,
     then documents 0 through 49 will be collected and cached.  Any further
     requests in that range can be satisfied via the cache.
  -->
<queryResultWindowSize>20</queryResultWindowSize>

<!-- Maximum number of documents to cache for any entry in the
     queryResultCache.
  -->
<queryResultMaxDocsCached>200</queryResultMaxDocsCached>

<!-- Query Related Event Listeners

     Various IndexSearcher related events can trigger Listeners to
     take actions.

     newSearcher - fired whenever a new searcher is being prepared
     and there is a current searcher handling requests (aka
     registered).  It can be used to prime certain caches to
     prevent long request times for certain requests.

     firstSearcher - fired whenever a new searcher is being
     prepared but there is no current registered searcher to handle
     requests or to gain autowarming data from.
```

```
         -->
       <!-- QuerySenderListener takes an array of NamedList and executes a
            local query request for each NamedList in sequence.
         -->
       <listener event="newSearcher" class="solr.QuerySenderListener">
           <arr name="queries">
               <!--
                  <lst><str name="q">solr</str><str name="sort">price asc</str></lst>
                  <lst><str name="q">rocks</str><str name="sort">weight
asc</str></lst>
                -->
           </arr>
       </listener>
       <listener event="firstSearcher" class="solr.QuerySenderListener">
           <arr name="queries">
               <lst>
                   <str name="q">static firstSearcher warming in solrconfig.xml</str>
               </lst>
           </arr>
       </listener>

       <!-- Use Cold Searcher

            If a search request comes in and there is no current
            registered searcher, then immediately register the still
            warming searcher and use it.  If "false" then all requests
            will block until the first searcher is done warming.
         -->
       <useColdSearcher>false</useColdSearcher>

       <!-- Max Warming Searchers

            Maximum number of searchers that may be warming in the
            background concurrently.  An error is returned if this limit
            is exceeded.

            Recommend values of 1-2 for read-only slaves, higher for
            masters w/o cache warming.
         -->
       <maxWarmingSearchers>2</maxWarmingSearchers>

   </query>


   <!-- Request Dispatcher

        This section contains instructions for how the SolrDispatchFilter
        should behave when processing requests for this SolrCore.

        handleSelect is a legacy option that affects the behavior of requests
        such as /select?qt=XXX

        handleSelect="true" will cause the SolrDispatchFilter to process
        the request and dispatch the query to a handler specified by the
        "qt" param, assuming "/select" isn't already registered.

        handleSelect="false" will cause the SolrDispatchFilter to
        ignore "/select" requests, resulting in a 404 unless a handler
        is explicitly registered with the name "/select"
```

```
       handleSelect="true" is not recommended for new users, but is the default
       for backwards compatibility
    -->
  <requestDispatcher handleSelect="false">
      <!-- Request Parsing

           These settings indicate how Solr Requests may be parsed, and
           what restrictions may be placed on the ContentStreams from
           those requests

           enableRemoteStreaming - enables use of the stream.file
           and stream.url parameters for specifying remote streams.

           multipartUploadLimitInKB - specifies the max size (in KiB) of
           Multipart File Uploads that Solr will allow in a Request.

           formdataUploadLimitInKB - specifies the max size (in KiB) of
           form data (application/x-www-form-urlencoded) sent via
           POST. You can use POST to pass request parameters not
           fitting into the URL.

           addHttpRequestToContext - if set to true, it will instruct
           the requestParsers to include the original HttpServletRequest
           object in the context map of the SolrQueryRequest under the
           key "httpRequest". It will not be used by any of the existing
           Solr components, but may be useful when developing custom
           plugins.

           *** WARNING ***
           The settings below authorize Solr to fetch remote files, You
           should make sure your system has some authentication before
           using enableRemoteStreaming="true"

        -->
      <requestParsers enableRemoteStreaming="true"
                      multipartUploadLimitInKB="2048000"
                      formdataUploadLimitInKB="2048"
                      addHttpRequestToContext="false"/>

      <!-- HTTP Caching

           Set HTTP caching related parameters (for proxy caches and clients).

           The options below instruct Solr not to output any HTTP Caching
           related headers
        -->
      <httpCaching never304="true"/>
      <!-- If you include a <cacheControl> directive, it will be used to
           generate a Cache-Control header (as well as an Expires header
           if the value contains "max-age=")

           By default, no Cache-Control header is generated.

           You can use the <cacheControl> option even if you have set
           never304="true"
        -->
      <!--
         <httpCaching never304="true" >
           <cacheControl>max-age=30, public</cacheControl>
         </httpCaching>
```

```
       -->
    <!-- To enable Solr to respond with automatically generated HTTP
         Caching headers, and to response to Cache Validation requests
         correctly, set the value of never304="false"

         This will cause Solr to generate Last-Modified and ETag
         headers based on the properties of the Index.

         The following options can also be specified to affect the
         values of these headers...

         lastModFrom - the default value is "openTime" which means the
         Last-Modified value (and validation against If-Modified-Since
         requests) will all be relative to when the current Searcher
         was opened.  You can change it to lastModFrom="dirLastMod" if
         you want the value to exactly correspond to when the physical
         index was last modified.

         etagSeed="..." is an option you can change to force the ETag
         header (and validation against If-None-Match requests) to be
         different even if the index has not changed (ie: when making
         significant changes to your config file)

         (lastModifiedFrom and etagSeed are both ignored if you use
         the never304="true" option)
       -->
    <!--
       <httpCaching lastModifiedFrom="openTime"
                    etagSeed="Solr">
         <cacheControl>max-age=30, public</cacheControl>
       </httpCaching>
      -->
  </requestDispatcher>

  <!-- Request Handlers

       http://wiki.apache.org/solr/SolrRequestHandler

       Incoming queries will be dispatched to a specific handler by name
       based on the path specified in the request.

       Legacy behavior: If the request path uses "/select" but no Request
       Handler has that name, and if handleSelect="true" has been specified in
       the requestDispatcher, then the Request Handler is dispatched based on
       the qt parameter.  Handlers without a leading '/' are accessed this way
       like so: http://host/app/[core/]select?qt=name  If no qt is
       given, then the requestHandler that declares default="true" will be
       used or the one named "standard".

       If a Request Handler is declared with startup="lazy", then it will
       not be initialized until the first request that uses it.

    -->
  <!-- SearchHandler

       http://wiki.apache.org/solr/SearchHandler

       For processing Search Queries, the primary Request Handler
       provided with Solr is "SearchHandler" It delegates to a sequent
       of SearchComponents (see below) and supports distributed
```

```
        queries across multiple shards
     -->
    <requestHandler name="/select" class="solr.SearchHandler">
        <lst name="defaults">
            <str name="defType">edismax</str>
            <str name="qf">
                text
                collection^3
                hashtags^3
                cluster_label^2.5
                lda_topics^2.0
                ner_people^2.0
                ner_locations^2.0
                ner_organizations^2.0
            </str>
            <str name="echoParams">explicit</str>
            <int name="rows">10</int>
            <str name="df">text</str>
            <str name="fl">*,score</str>
        </lst>
        <arr name="last-components">
            <str>idealSocialBoostComponent</str>
            <str>idealTopicSupplementComponent</str>
        </arr>
    </requestHandler>

    <!-- A request handler that returns indented JSON by default -->
    <requestHandler name="/idealquery"
class="edu.vt.dlib.ideal.solr.IDEALSearchRequestHandler">
        <lst name="defaults">
            <str name="echoParams">explicit</str>
            <str name="wt">json</str>
            <str name="indent">true</str>
            <str name="df">text</str>
        </lst>
    </requestHandler>


    <!-- A request handler that returns indented JSON by default -->
    <requestHandler name="/query" class="solr.SearchHandler">
        <lst name="defaults">
            <str name="echoParams">explicit</str>
            <str name="wt">json</str>
            <str name="indent">true</str>
            <str name="df">text</str>
        </lst>
    </requestHandler>


    <!-- realtime get handler, guaranteed to return the latest stored fields of
         any document, without the need to commit or open a new searcher.  The
         current implementation relies on the updateLog feature being enabled.

         ** WARNING **
         Do NOT disable the realtime get handler at /get if you are using
         SolrCloud otherwise any leader election will cause a full sync in ALL
         replicas for the shard in question. Similarly, a replica recovery will
         also always fetch the complete index from the leader because a partial
         sync will not be possible in the absence of this handler.
    -->
```

```
<requestHandler name="/get" class="solr.RealTimeGetHandler">
    <lst name="defaults">
        <str name="omitHeader">true</str>
        <str name="wt">json</str>
        <str name="indent">true</str>
    </lst>
</requestHandler>

<!--
  The export request handler is used to export full sorted result sets.
  Do not change these defaults.
-->

<requestHandler name="/export" class="solr.SearchHandler">
    <lst name="invariants">
        <str name="rq">{!xport}</str>
        <str name="wt">xsort</str>
        <str name="distrib">false</str>
    </lst>

    <arr name="components">
        <str>query</str>
    </arr>
</requestHandler>


<!-- A Robust Example

     This example SearchHandler declaration shows off usage of the
     SearchHandler with many defaults declared

     Note that multiple instances of the same Request Handler
     (SearchHandler) can be registered multiple times with different
     names (and different init parameters)
  -->
<requestHandler name="/browse" class="solr.SearchHandler">
    <lst name="defaults">
        <str name="echoParams">explicit</str>

        <!-- VelocityResponseWriter settings -->
        <str name="wt">velocity</str>
        <str name="v.template">browse</str>
        <str name="v.layout">layout</str>
        <str name="title">IDEAL</str>

        <!-- Query settings -->
        <str name="defType">edismax</str>
        <str name="df">text</str>
        <str name="mm">100%</str>
        <str name="q.alt">*:*</str>
        <str name="rows">10</str>
        <str name="fl">id, collection, user_screen_name, text,score</str>


        <int name="mlt.count">3</int>

        <!-- Faceting defaults -->
        <str name="facet">on</str>
        <str name="facet.missing">true</str>
        <str name="facet.field">collection</str>
```

```
        <str name="facet.field">user_screen_name</str>

        <str name="facet.mincount">1</str>

        <!-- Highlighting defaults -->
        <str name="hl.preserveMulti">true</str>
        <str name="hl.encoder">html</str>
        <str name="hl.simple.pre">&lt;b&gt;</str>
        <str name="hl.simple.post">&lt;/b&gt;</str>
        <str name="f.title.hl.fragsize">0</str>
        <str name="f.title.hl.alternateField">title</str>
        <str name="f.name.hl.fragsize">0</str>
        <str name="f.name.hl.alternateField">name</str>
        <str name="f.content.hl.snippets">3</str>
        <str name="f.content.hl.fragsize">200</str>
        <str name="f.content.hl.alternateField">content</str>
        <str name="f.content.hl.maxAlternateFieldLength">750</str>

        <!-- Spell checking defaults -->
        <str name="spellcheck">on</str>
        <str name="spellcheck.extendedResults">false</str>
        <str name="spellcheck.count">5</str>
        <str name="spellcheck.alternativeTermCount">2</str>
        <str name="spellcheck.maxResultsForSuggest">5</str>
        <str name="spellcheck.collate">true</str>
        <str name="spellcheck.collateExtendedResults">true</str>
        <str name="spellcheck.maxCollationTries">5</str>
        <str name="spellcheck.maxCollations">3</str>
    </lst>

    <!-- append spellchecking to our list of components -->
    <arr name="last-components">
        <str>spellcheck</str>
    </arr>
</requestHandler>


<!-- Update Request Handler.

     http://wiki.apache.org/solr/UpdateXmlMessages

     The canonical Request Handler for Modifying the Index through
     commands specified using XML, JSON, CSV, or JAVABIN

     Note: Since solr1.1 requestHandlers requires a valid content
     type header if posted in the body. For example, curl now
     requires: -H 'Content-type:text/xml; charset=utf-8'

     To override the request content type and force a specific
     Content-type, use the request parameter:
       ?update.contentType=text/csv

     This handler will pick a response format to match the input
     if the 'wt' parameter is not explicit
  -->
<requestHandler name="/update" class="solr.UpdateRequestHandler">
    <!-- See below for information on defining
         updateRequestProcessorChains that can be used by name
         on each Update Request
      -->
```

```
    <!--
       <lst name="defaults">
         <str name="update.chain">dedupe</str>
       </lst>
       -->
  </requestHandler>

  <!-- The following are implicitly added
  <requestHandler name="/update/json" class="solr.UpdateRequestHandler">
        <lst name="defaults">
         <str name="stream.contentType">application/json</str>
       </lst>
  </requestHandler>
  <requestHandler name="/update/csv" class="solr.UpdateRequestHandler">
        <lst name="defaults">
         <str name="stream.contentType">application/csv</str>
       </lst>
  </requestHandler>
  -->

  <!-- Solr Cell Update Request Handler

       http://wiki.apache.org/solr/ExtractingRequestHandler

    -->
  <requestHandler name="/update/extract"
                  startup="lazy"
                  class="solr.extraction.ExtractingRequestHandler">
      <lst name="defaults">
          <str name="lowernames">true</str>
          <str name="uprefix">ignored_</str>

          <!-- capture link hrefs but ignore div attributes -->
          <str name="captureAttr">true</str>
          <str name="fmap.a">links</str>
          <str name="fmap.div">ignored_</str>
      </lst>
  </requestHandler>


  <!-- Field Analysis Request Handler

       RequestHandler that provides much the same functionality as
       analysis.jsp. Provides the ability to specify multiple field
       types and field names in the same request and outputs
       index-time and query-time analysis for each of them.

       Request parameters are:
       analysis.fieldname - field name whose analyzers are to be used

       analysis.fieldtype - field type whose analyzers are to be used
       analysis.fieldvalue - text for index-time analysis
       q (or analysis.q) - text for query time analysis
       analysis.showmatch (true|false) - When set to true and when
           query analysis is performed, the produced tokens of the
           field value analysis will be marked as "matched" for every
           token that is produces by the query analysis
   -->
  <requestHandler name="/analysis/field"
                  startup="lazy"
```

```
                  class="solr.FieldAnalysisRequestHandler"/>


   <!-- Document Analysis Handler

        http://wiki.apache.org/solr/AnalysisRequestHandler

        An analysis handler that provides a breakdown of the analysis
        process of provided documents. This handler expects a (single)
        content stream with the following format:

        <docs>
          <doc>
            <field name="id">1</field>
            <field name="name">The Name</field>
            <field name="text">The Text Value</field>
          </doc>
          <doc>...</doc>
          <doc>...</doc>
          ...
        </docs>

      Note: Each document must contain a field which serves as the
      unique key. This key is used in the returned response to associate
      an analysis breakdown to the analyzed document.

      Like the FieldAnalysisRequestHandler, this handler also supports
      query analysis by sending either an "analysis.query" or "q"
      request parameter that holds the query text to be analyzed. It
      also supports the "analysis.showmatch" parameter which when set to
      true, all field tokens that match the query tokens will be marked
      as a "match".
   -->
   <requestHandler name="/analysis/document"
                   class="solr.DocumentAnalysisRequestHandler"
                   startup="lazy"/>

   <!-- Admin Handlers

        Admin Handlers - This will register all the standard admin
        RequestHandlers.
     -->
   <requestHandler name="/admin/"
                   class="solr.admin.AdminHandlers"/>
   <!-- This single handler is equivalent to the following... -->
   <!--
      <requestHandler name="/admin/luke"       class="solr.admin.LukeRequestHandler"
/>
      <requestHandler name="/admin/system"     class="solr.admin.SystemInfoHandler"
/>
      <requestHandler name="/admin/plugins"    class="solr.admin.PluginInfoHandler"
/>
      <requestHandler name="/admin/threads"    class="solr.admin.ThreadDumpHandler"
/>
      <requestHandler name="/admin/properties"
class="solr.admin.PropertiesRequestHandler" />
      <requestHandler name="/admin/file"
class="solr.admin.ShowFileRequestHandler" >
     -->
   <!-- If you wish to hide files under ${solr.home}/conf, explicitly
```

```
        register the ShowFileRequestHandler using the definition below.
        NOTE: The glob pattern ('*') is the only pattern supported at present, *.xml
will
              not exclude all files ending in '.xml'. Use it to exclude _all_ updates
     -->
    <!--
       <requestHandler name="/admin/file"
                       class="solr.admin.ShowFileRequestHandler" >
         <lst name="invariants">
           <str name="hidden">synonyms.txt</str>
           <str name="hidden">anotherfile.txt</str>
           <str name="hidden">*</str>
         </lst>
       </requestHandler>
      -->

    <!-- ping/healthcheck -->
    <requestHandler name="/admin/ping" class="solr.PingRequestHandler">
        <lst name="invariants">
            <str name="q">solrpingquery</str>
        </lst>
        <lst name="defaults">
            <str name="echoParams">all</str>
        </lst>
        <!-- An optional feature of the PingRequestHandler is to configure the
             handler with a "healthcheckFile" which can be used to enable/disable
             the PingRequestHandler.
             relative paths are resolved against the data dir
          -->
        <!-- <str name="healthcheckFile">server-enabled.txt</str> -->
    </requestHandler>

    <!-- Echo the request contents back to the client -->
    <requestHandler name="/debug/dump" class="solr.DumpRequestHandler">
        <lst name="defaults">
            <str name="echoParams">explicit</str>
            <str name="echoHandler">true</str>
        </lst>
    </requestHandler>

    <!-- Solr Replication

         The SolrReplicationHandler supports replicating indexes from a
         "master" used for indexing and "slaves" used for queries.

         http://wiki.apache.org/solr/SolrReplication

         It is also necessary for SolrCloud to function (in Cloud mode, the
         replication handler is used to bulk transfer segments when nodes
         are added or need to recover).

         https://wiki.apache.org/solr/SolrCloud/
      -->
    <requestHandler name="/replication" class="solr.ReplicationHandler">
        <!--
           To enable simple master/slave replication, uncomment one of the
           sections below, depending on whether this solr instance should be
           the "master" or a "slave".  If this instance is a "slave" you will
           also need to fill in the masterUrl to point to a real machine.
        -->
```

```
    <!--
       <lst name="master">
         <str name="replicateAfter">commit</str>
         <str name="replicateAfter">startup</str>
         <str name="confFiles">schema.xml,stopwords.txt</str>
       </lst>
    -->
    <!--
       <lst name="slave">
         <str name="masterUrl">http://your-master-hostname:8983/solr</str>
         <str name="pollInterval">00:00:60</str>
       </lst>
    -->
  </requestHandler>

  <!-- Search Components

       Search components are registered to SolrCore and used by
       instances of SearchHandler (which can access them by name)

       By default, the following components are available:

       <searchComponent name="query"     class="solr.QueryComponent" />
       <searchComponent name="facet"     class="solr.FacetComponent" />
       <searchComponent name="mlt"       class="solr.MoreLikeThisComponent" />
       <searchComponent name="highlight" class="solr.HighlightComponent" />
       <searchComponent name="stats"     class="solr.StatsComponent" />
       <searchComponent name="debug"     class="solr.DebugComponent" />

       Default configuration in a requestHandler would look like:

       <arr name="components">
         <str>query</str>
         <str>facet</str>
         <str>mlt</str>
         <str>highlight</str>
         <str>stats</str>
         <str>debug</str>
       </arr>

       If you register a searchComponent to one of the standard names,
       that will be used instead of the default.

       To insert components before or after the 'standard' components, use:

       <arr name="first-components">
         <str>myFirstComponentName</str>
       </arr>

       <arr name="last-components">
         <str>myLastComponentName</str>
       </arr>

       NOTE: The component registered with the name "debug" will
       always be executed after the "last-components"

     -->

  <!-- Spell Check
```

```
         The spell check component can return a list of alternative spelling
         suggestions.

         http://wiki.apache.org/solr/SpellCheckComponent
      -->

    <searchComponent name="idealSocialBoostComponent"
class="edu.vt.dlib.ideal.solr.IDEALSocialBoostComponent"/>
    <searchComponent name="idealTopicSupplementComponent"
class="edu.vt.dlib.ideal.solr.IDEALTopicSupplementComponent"/>

    <searchComponent name="spellcheck" class="solr.SpellCheckComponent">

        <str name="queryAnalyzerFieldType">text_general</str>

        <!-- Multiple "Spell Checkers" can be declared and used by this
             component
          -->

        <!-- a spellchecker built from a field of the main index -->
        <lst name="spellchecker">
            <str name="name">default</str>
            <str name="field">text</str>
            <str name="classname">solr.DirectSolrSpellChecker</str>
            <!-- the spellcheck distance measure used, the default is the internal
levenshtein -->
            <str name="distanceMeasure">internal</str>
            <!-- minimum accuracy needed to be considered a valid spellcheck
suggestion -->
            <float name="accuracy">0.5</float>
            <!-- the maximum #edits we consider when enumerating terms: can be 1 or 2
-->
            <int name="maxEdits">2</int>
            <!-- the minimum shared prefix when enumerating terms -->
            <int name="minPrefix">1</int>
            <!-- maximum number of inspections per result. -->
            <int name="maxInspections">5</int>
            <!-- minimum length of a query term to be considered for correction -->
            <int name="minQueryLength">4</int>
            <!-- maximum threshold of documents a query term can appear to be
considered for correction -->
            <float name="maxQueryFrequency">0.01</float>
            <!-- uncomment this to require suggestions to occur in 1% of the documents
                <float name="thresholdTokenFrequency">.01</float>
            -->
        </lst>

        <!-- a spellchecker that can break or combine words.  See "/spell" handler
below for usage -->
        <lst name="spellchecker">
            <str name="name">wordbreak</str>
            <str name="classname">solr.WordBreakSolrSpellChecker</str>
            <str name="field">name</str>
            <str name="combineWords">true</str>
            <str name="breakWords">true</str>
            <int name="maxChanges">10</int>
        </lst>

        <!-- a spellchecker that uses a different distance measure -->
        <!--
```

```
        <lst name="spellchecker">
          <str name="name">jarowinkler</str>
          <str name="field">spell</str>
          <str name="classname">solr.DirectSolrSpellChecker</str>
          <str name="distanceMeasure">
            org.apache.lucene.search.spell.JaroWinklerDistance
          </str>
        </lst>
       -->

    <!-- a spellchecker that use an alternate comparator

         comparatorClass be one of:
          1. score (default)
          2. freq (Frequency first, then score)
          3. A fully qualified class name
      -->
    <!--
       <lst name="spellchecker">
         <str name="name">freq</str>
         <str name="field">lowerfilt</str>
         <str name="classname">solr.DirectSolrSpellChecker</str>
         <str name="comparatorClass">freq</str>
      -->

    <!-- A spellchecker that reads the list of words from a file -->
    <!--
       <lst name="spellchecker">
         <str name="classname">solr.FileBasedSpellChecker</str>
         <str name="name">file</str>
         <str name="sourceLocation">spellings.txt</str>
         <str name="characterEncoding">UTF-8</str>
         <str name="spellcheckIndexDir">spellcheckerFile</str>
       </lst>
       -->
  </searchComponent>

  <!-- A request handler for demonstrating the spellcheck component.

       NOTE: This is purely as an example.  The whole purpose of the
       SpellCheckComponent is to hook it into the request handler that
       handles your normal user queries so that a separate request is
       not needed to get suggestions.

       IN OTHER WORDS, THERE IS REALLY GOOD CHANCE THE SETUP BELOW IS
       NOT WHAT YOU WANT FOR YOUR PRODUCTION SYSTEM!

       See http://wiki.apache.org/solr/SpellCheckComponent for details
       on the request parameters.
    -->
  <requestHandler name="/spell" class="solr.SearchHandler" startup="lazy">
      <lst name="defaults">
          <str name="df">text</str>
          <!-- Solr will use suggestions from both the 'default' spellchecker
               and from the 'wordbreak' spellchecker and combine them.
               collations (re-written queries) can include a combination of
               corrections from both spellcheckers -->
          <str name="spellcheck.dictionary">default</str>
          <str name="spellcheck.dictionary">wordbreak</str>
          <str name="spellcheck">on</str>
```

```
            <str name="spellcheck.extendedResults">true</str>
            <str name="spellcheck.count">10</str>
            <str name="spellcheck.alternativeTermCount">5</str>
            <str name="spellcheck.maxResultsForSuggest">5</str>
            <str name="spellcheck.collate">true</str>
            <str name="spellcheck.collateExtendedResults">true</str>
            <str name="spellcheck.maxCollationTries">10</str>
            <str name="spellcheck.maxCollations">5</str>
        </lst>
        <arr name="last-components">
            <str>spellcheck</str>
        </arr>
    </requestHandler>

    <!-- This causes long startup times on big indexes, even when never used.  See
SOLR-6679
    <searchComponent name="suggest" class="solr.SuggestComponent">
      <lst name="suggester">
        <str name="name">mySuggester</str>
        <str name="lookupImpl">FuzzyLookupFactory</str>
        <str name="dictionaryImpl">DocumentDictionaryFactory</str>
        <str name="field">cat</str>
        <str name="weightField">price</str>
        <str name="suggestAnalyzerFieldType">string</str>
      </lst>
    </searchComponent>

    <requestHandler name="/suggest" class="solr.SearchHandler" startup="lazy">
      <lst name="defaults">
        <str name="suggest">true</str>
        <str name="suggest.count">10</str>
      </lst>
      <arr name="components">
        <str>suggest</str>
      </arr>
    </requestHandler>
    -->

    <!-- Term Vector Component

         http://wiki.apache.org/solr/TermVectorComponent
      -->
    <searchComponent name="tvComponent" class="solr.TermVectorComponent"/>

    <!-- A request handler for demonstrating the term vector component

         This is purely as an example.

         In reality you will likely want to add the component to your
         already specified request handlers.
      -->
    <requestHandler name="/tvrh" class="solr.SearchHandler" startup="lazy">
        <lst name="defaults">
            <str name="df">text</str>
            <bool name="tv">true</bool>
        </lst>
        <arr name="last-components">
            <str>tvComponent</str>
        </arr>
    </requestHandler>
```

```
    <!-- Clustering Component

         You'll need to set the solr.clustering.enabled system property
         when running solr to run with clustering enabled:

              java -Dsolr.clustering.enabled=true -jar start.jar

         http://wiki.apache.org/solr/ClusteringComponent
         http://carrot2.github.io/solr-integration-strategies/
      -->
    <searchComponent name="clustering"
                     enable="${solr.clustering.enabled:false}"
                     class="solr.clustering.ClusteringComponent">
        <lst name="engine">
            <str name="name">lingo</str>

            <!-- Class name of a clustering algorithm compatible with the Carrot2
framework.

                 Currently available open source algorithms are:
                 * org.carrot2.clustering.lingo.LingoClusteringAlgorithm
                 * org.carrot2.clustering.stc.STCClusteringAlgorithm
                 * org.carrot2.clustering.kmeans.BisectingKMeansClusteringAlgorithm

                 See http://project.carrot2.org/algorithms.html for more information.

                 A commercial algorithm Lingo3G (needs to be installed separately) is
defined as:
                 * com.carrotsearch.lingo3g.Lingo3GClusteringAlgorithm
              -->
            <str
name="carrot.algorithm">org.carrot2.clustering.lingo.LingoClusteringAlgorithm</str>

            <!-- Override location of the clustering algorithm's resources
                 (attribute definitions and lexical resources).

                 A directory from which to load algorithm-specific stop words,
                 stop labels and attribute definition XMLs.

                 For an overview of Carrot2 lexical resources, see:
                 http://download.carrot2.org/head/manual/#chapter.lexical-resources

                 For an overview of Lingo3G lexical resources, see:
                 http://download.carrotsearch.com/lingo3g/manual/#chapter.lexical-
resources
              -->
            <str name="carrot.resourcesDir">clustering/carrot2</str>
        </lst>

        <!-- An example definition for the STC clustering algorithm. -->
        <lst name="engine">
            <str name="name">stc</str>
            <str
name="carrot.algorithm">org.carrot2.clustering.stc.STCClusteringAlgorithm</str>
        </lst>

        <!-- An example definition for the bisecting kmeans clustering algorithm. -->
        <lst name="engine">
            <str name="name">kmeans</str>
```

```
            <str
name="carrot.algorithm">org.carrot2.clustering.kmeans.BisectingKMeansClusteringAlgorit
hm</str>
        </lst>
    </searchComponent>

    <!-- A request handler for demonstrating the clustering component

         This is purely as an example.

         In reality you will likely want to add the component to your
         already specified request handlers.
      -->
    <requestHandler name="/clustering"
                    startup="lazy"
                    enable="${solr.clustering.enabled:false}"
                    class="solr.SearchHandler">
        <lst name="defaults">
            <bool name="clustering">true</bool>
            <bool name="clustering.results">true</bool>
            <!-- Field name with the logical "title" of a each document (optional) -->
            <str name="carrot.title">name</str>
            <!-- Field name with the logical "URL" of a each document (optional) -->
            <str name="carrot.url">id</str>
            <!-- Field name with the logical "content" of a each document (optional) -
->
            <str name="carrot.snippet">features</str>
            <!-- Apply highlighter to the title/ content and use this for clustering.
-->
            <bool name="carrot.produceSummary">true</bool>
            <!-- the maximum number of labels per cluster -->
            <!--<int name="carrot.numDescriptions">5</int>-->
            <!-- produce sub clusters -->
            <bool name="carrot.outputSubClusters">false</bool>

            <!-- Configure the remaining request handler parameters. -->
            <str name="defType">edismax</str>
            <str name="qf">
                text^0.5 features^1.0 name^1.2 sku^1.5 id^10.0 manu^1.1 cat^1.4
            </str>
            <str name="q.alt">*:*</str>
            <str name="rows">10</str>
            <str name="fl">*,score</str>
        </lst>
        <arr name="last-components">
            <str>clustering</str>
        </arr>
    </requestHandler>

    <!-- Terms Component

         http://wiki.apache.org/solr/TermsComponent

         A component to return terms and document frequency of those
         terms
      -->
    <searchComponent name="terms" class="solr.TermsComponent"/>

    <!-- A request handler for demonstrating the terms component -->
    <requestHandler name="/terms" class="solr.SearchHandler" startup="lazy">
```

```
    <lst name="defaults">
        <bool name="terms">true</bool>
        <bool name="distrib">false</bool>
    </lst>
    <arr name="components">
        <str>terms</str>
    </arr>
</requestHandler>


<!-- Query Elevation Component

     http://wiki.apache.org/solr/QueryElevationComponent

     a search component that enables you to configure the top
     results for a given query regardless of the normal lucene
     scoring.
  -->
<searchComponent name="elevator" class="solr.QueryElevationComponent">
    <!-- pick a fieldType to analyze queries -->
    <str name="queryFieldType">string</str>
    <str name="config-file">elevate.xml</str>
</searchComponent>

<!-- A request handler for demonstrating the elevator component -->
<requestHandler name="/elevate" class="solr.SearchHandler" startup="lazy">
    <lst name="defaults">
        <str name="echoParams">explicit</str>
        <str name="df">text</str>
    </lst>
    <arr name="last-components">
        <str>elevator</str>
    </arr>
</requestHandler>

<!-- Highlighting Component

     http://wiki.apache.org/solr/HighlightingParameters
  -->
<searchComponent class="solr.HighlightComponent" name="highlight">
    <highlighting>
        <!-- Configure the standard fragmenter -->
        <!-- This could most likely be commented out in the "default" case -->
        <fragmenter name="gap"
                    default="true"
                    class="solr.highlight.GapFragmenter">
            <lst name="defaults">
                <int name="hl.fragsize">100</int>
            </lst>
        </fragmenter>

        <!-- A regular-expression-based fragmenter
             (for sentence extraction)
          -->
        <fragmenter name="regex"
                    class="solr.highlight.RegexFragmenter">
            <lst name="defaults">
                <!-- slightly smaller fragsizes work better because of slop -->
                <int name="hl.fragsize">70</int>
                <!-- allow 50% slop on fragment sizes -->
```

```
        <float name="hl.regex.slop">0.5</float>
        <!-- a basic sentence pattern -->
        <str name="hl.regex.pattern">[-\w ,/\n\&quot;&apos;]{20,200}</str>
    </lst>
</fragmenter>

<!-- Configure the standard formatter -->
<formatter name="html"
           default="true"
           class="solr.highlight.HtmlFormatter">
    <lst name="defaults">
        <str name="hl.simple.pre"><![CDATA[<em>]]></str>
        <str name="hl.simple.post"><![CDATA[</em>]]></str>
    </lst>
</formatter>

<!-- Configure the standard encoder -->
<encoder name="html"
         class="solr.highlight.HtmlEncoder"/>

<!-- Configure the standard fragListBuilder -->
<fragListBuilder name="simple"
                 class="solr.highlight.SimpleFragListBuilder"/>

<!-- Configure the single fragListBuilder -->
<fragListBuilder name="single"
                 class="solr.highlight.SingleFragListBuilder"/>

<!-- Configure the weighted fragListBuilder -->
<fragListBuilder name="weighted"
                 default="true"
                 class="solr.highlight.WeightedFragListBuilder"/>

<!-- default tag FragmentsBuilder -->
<fragmentsBuilder name="default"
                  default="true"
                  class="solr.highlight.ScoreOrderFragmentsBuilder">
    <!--
    <lst name="defaults">
      <str name="hl.multiValuedSeparatorChar">/</str>
    </lst>
    -->
</fragmentsBuilder>

<!-- multi-colored tag FragmentsBuilder -->
<fragmentsBuilder name="colored"
                  class="solr.highlight.ScoreOrderFragmentsBuilder">
    <lst name="defaults">
        <str name="hl.tag.pre"><![CDATA[
   <b style="background:yellow">,<b style="background:lawgreen">,
   <b style="background:aquamarine">,<b style="background:magenta">,
   <b style="background:palegreen">,<b style="background:coral">,
   <b style="background:wheat">,<b style="background:khaki">,
   <b style="background:lime">,<b style="background:deepskyblue">]]></str>
        <str name="hl.tag.post"><![CDATA[</b>]]></str>
    </lst>
</fragmentsBuilder>

<boundaryScanner name="default"
                 default="true"
```

```
                            class="solr.highlight.SimpleBoundaryScanner">
                <lst name="defaults">
                    <str name="hl.bs.maxScan">10</str>
                    <str name="hl.bs.chars">.,!? &#9;&#10;&#13;</str>
                </lst>
            </boundaryScanner>

            <boundaryScanner name="breakIterator"
                            class="solr.highlight.BreakIteratorBoundaryScanner">
                <lst name="defaults">
                    <!-- type should be one of CHARACTER, WORD(default), LINE and
SENTENCE -->
                    <str name="hl.bs.type">WORD</str>
                    <!-- language and country are used when constructing Locale
object.  -->
                    <!-- And the Locale object will be used when getting instance of
BreakIterator -->
                    <str name="hl.bs.language">en</str>
                    <str name="hl.bs.country">US</str>
                </lst>
            </boundaryScanner>
        </highlighting>
    </searchComponent>

    <!-- Update Processors

         Chains of Update Processor Factories for dealing with Update
         Requests can be declared, and then used by name in Update
         Request Processors

         http://wiki.apache.org/solr/UpdateRequestProcessor

      -->
    <!-- Deduplication

         An example dedup update processor that creates the "id" field
         on the fly based on the hash code of some other fields.  This
         example has overwriteDupes set to false since we are using the
         id field as the signatureField and Solr will maintain
         uniqueness based on that anyway.

      -->
    <!--
       <updateRequestProcessorChain name="dedupe">
         <processor class="solr.processor.SignatureUpdateProcessorFactory">
           <bool name="enabled">true</bool>
           <str name="signatureField">id</str>
           <bool name="overwriteDupes">false</bool>
           <str name="fields">name,features,cat</str>
           <str name="signatureClass">solr.processor.Lookup3Signature</str>
         </processor>
         <processor class="solr.LogUpdateProcessorFactory" />
         <processor class="solr.RunUpdateProcessorFactory" />
       </updateRequestProcessorChain>
      -->

    <!-- Language identification

         This example update chain identifies the language of the incoming
         documents using the langid contrib. The detected language is
```

```
        written to field language_s. No field name mapping is done.
        The fields used for detection are text, title, subject and description,
        making this example suitable for detecting languages form full-text
        rich documents injected via ExtractingRequestHandler.
        See more about langId at http://wiki.apache.org/solr/LanguageDetection
     -->
    <!--
     <updateRequestProcessorChain name="langid">
       <processor
class="org.apache.solr.update.processor.TikaLanguageIdentifierUpdateProcessorFactory">
        <str name="langid.fl">text,title,subject,description</str>
        <str name="langid.langField">language_s</str>
        <str name="langid.fallback">en</str>
       </processor>
       <processor class="solr.LogUpdateProcessorFactory" />
       <processor class="solr.RunUpdateProcessorFactory" />
     </updateRequestProcessorChain>
    -->

    <!-- Script update processor

      This example hooks in an update processor implemented using JavaScript.

      See more about the script update processor at
http://wiki.apache.org/solr/ScriptUpdateProcessor
    -->
    <!--
      <updateRequestProcessorChain name="script">
        <processor class="solr.StatelessScriptUpdateProcessorFactory">
          <str name="script">update-script.js</str>
          <lst name="params">
            <str name="config_param">example config parameter</str>
          </lst>
        </processor>
        <processor class="solr.RunUpdateProcessorFactory" />
      </updateRequestProcessorChain>
    -->

    <!-- Response Writers

         http://wiki.apache.org/solr/QueryResponseWriter

         Request responses will be written using the writer specified by
         the 'wt' request parameter matching the name of a registered
         writer.

         The "default" writer is the default and will be used if 'wt' is
         not specified in the request.
      -->
    <!-- The following response writers are implicitly configured unless
         overridden...
      -->
    <!--
       <queryResponseWriter name="xml"
                            default="true"
                            class="solr.XMLResponseWriter" />
       <queryResponseWriter name="json" class="solr.JSONResponseWriter"/>
       <queryResponseWriter name="python" class="solr.PythonResponseWriter"/>
       <queryResponseWriter name="ruby" class="solr.RubyResponseWriter"/>
       <queryResponseWriter name="php" class="solr.PHPResponseWriter"/>
```

```
      <queryResponseWriter name="phps" class="solr.PHPSerializedResponseWriter"/>
      <queryResponseWriter name="csv" class="solr.CSVResponseWriter"/>
      <queryResponseWriter name="schema.xml" class="solr.SchemaXmlResponseWriter"/>
     -->

    <queryResponseWriter name="json" class="solr.JSONResponseWriter">
        <!-- For the purposes of the tutorial, JSON responses are written as
         plain text so that they are easy to read in *any* browser.
         If you expect a MIME type of "application/json" just remove this override.
        -->
        <str name="content-type">text/plain; charset=UTF-8</str>
    </queryResponseWriter>

    <!--
       Custom response writers can be declared as needed...
     -->
    <queryResponseWriter name="velocity" class="solr.VelocityResponseWriter"
startup="lazy"/>


    <!-- XSLT response writer transforms the XML output by any xslt file found
         in Solr's conf/xslt directory.  Changes to xslt files are checked for
         every xsltCacheLifetimeSeconds.
     -->
    <queryResponseWriter name="xslt" class="solr.XSLTResponseWriter">
        <int name="xsltCacheLifetimeSeconds">5</int>
    </queryResponseWriter>

    <!-- Query Parsers

         http://wiki.apache.org/solr/SolrQuerySyntax

         Multiple QParserPlugins can be registered by name, and then
         used in either the "defType" param for the QueryComponent (used
         by SearchHandler) or in LocalParams
     -->
    <!-- example of registering a query parser -->
    <!--
       <queryParser name="myparser" class="com.mycompany.MyQParserPlugin"/>
     -->

    <!-- Function Parsers

         http://wiki.apache.org/solr/FunctionQuery

         Multiple ValueSourceParsers can be registered by name, and then
         used as function names when using the "func" QParser.
     -->
    <!-- example of registering a custom function parser  -->
    <!--
       <valueSourceParser name="myfunc"
                          class="com.mycompany.MyValueSourceParser" />
     -->


    <!-- Document Transformers
         http://wiki.apache.org/solr/DocTransformers
     -->
    <!--
       Could be something like:
```

```
      <transformer name="db" class="com.mycompany.LoadFromDatabaseTransformer" >
        <int name="connection">jdbc://....</int>
      </transformer>

      To add a constant value to all docs, use:
      <transformer name="mytrans2"
class="org.apache.solr.response.transform.ValueAugmenterFactory" >
        <int name="value">5</int>
      </transformer>

      If you want the user to still be able to change it with _value:something_ use
this:
      <transformer name="mytrans3"
class="org.apache.solr.response.transform.ValueAugmenterFactory" >
        <double name="defaultValue">5</double>
      </transformer>

       If you are using the QueryElevationComponent, you may wish to mark documents
that get boosted.  The
       EditorialMarkerFactory will do exactly that:
      <transformer name="qecBooster"
class="org.apache.solr.response.transform.EditorialMarkerFactory" />
     -->


   <!-- Legacy config for the admin interface -->
   <admin>
       <defaultQuery>*:*</defaultQuery>
   </admin>

</config>
```

D. Morphline.conf

This is a morphline.conf file for **tweets**. We have a created a similar file for webpages, which you can extract from VTechWorks.

```
SOLR_LOCATOR : {
  # Name of solr collection
  collection : tweets

  # ZooKeeper ensemble
  zkHost : "node1.dlrl:2181/solr"
}

morphlines: [
        {
            id: morphline1
            importCommands: ["org.kitesdk.morphline.**", "com.ngdata.**",
"com.cloudera.cdk.morphline.**", "org.apache.solr.**"]

            commands: [
                {
                    extractHBaseCells {
```

```
                    mappings: [
                        {
                            inputColumn: "original:text_clean"
                            outputField: "text"
                            type: string
                            source: value
                        }
                      {
                            inputColumn: "original:created_at"
                            outputField: "created_at"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "original:collection"
                            outputField: "collection"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "original:source"
                            outputField: "source"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "original:user_screen_name"
                            outputField: "user_screen_name"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "original:user_id"
                            outputField: "user_id"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "original:lang"
                            outputField: "lang"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "original:retweet_count"
                            outputField: "retweet_count"
                            type: int
                            source: value
                        }
                        {
                            inputColumn: "original:favorite_count"
                            outputField: "favorite_count"
                            type: int
                            source: value
                        }
                        {
                            inputColumn: "original:contributors_id"
                            outputField: "contributors_id"
                            type: string
                            source: value
```

```
}
{
    inputColumn: "original:coordinates"
    outputField: "coordinates"
    type: string
    source: value
}
{
    inputColumn: "original:urls"
    outputField: "urls_multiple"
    type: string
    source: value
}
{
    inputColumn: "original:hashtags"
    outputField: "hashtags_multiple"
    type: string
    source: value
}
{
    inputColumn: "original:user_mentions_id"
    outputField: "user_mentions_id_multiple"
    type: string
    source: value
}
{
    inputColumn: "original:in_reply_to_user_id"
    outputField: "in_reply_to_user_id"
    type: string
    source: value
}
{
    inputColumn: "original:in_reply_to_status_id"
    outputField: "in_reply_to_status_id"
    type: string
    source: value
}
{
    inputColumn: "analysis:cluster_label"
    outputField: "cluster_label"
    type: string
    source: value
}
{
    inputColumn: "analysis:cluster_id"
    outputField: "cluster_id"
    type: string
    source: value
}
{
    inputColumn: "analysis:ner_people"
    outputField: "ner_people_multiple"
    type: string
    source: value
}
{
    inputColumn: "analysis:ner_locations"
    outputField: "ner_locations_multiple"
    type: string
    source: value
```

```
                        }
                        {
                            inputColumn: "analysis:ner_dates"
                            outputField: "ner_dates_multiple"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "analysis:ner_organizations"
                            outputField: "ner_organizations_multiple"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "analysis:importance"
                            outputField: "social_vector_json"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "analysis:class"
                            outputField: "classification_labels_multiple"
                            type: string
                            source: value
                        }
                        {
                            inputColumn: "analysis:lda_topics"
                            outputField: "lda_topics_multiple"
                            type: string
                            source: value
                        }
                    ]
  }
}
        {
            split {
                inputField: "urls_multiple"
                outputField: "urls"
                separator: "|"
            }
        }
       {
            split {
                inputField: "hashtags_multiple"
                outputField: "hashtags"
                separator: "|"
            }
        }
       {
            split {
                inputField: "user_mentions_id_multiple"
                outputField: "user_mentions_id"
                separator: "|"
            }
        }
        {
            split {
                inputField: "ner_people_multiple"
                outputField: "ner_people"
                separator: "|"
```

```
                    }
                }
                {
                    split {
                        inputField: "ner_locations_multiple"
                        outputField: "ner_locations"
                        separator: "|"
                    }
                }
                {
                    split {
                        inputField: "ner_dates_multiple"
                        outputField: "ner_dates"
                        separator: "|"
                    }
                }
                {
                    split {
                        inputField: "ner_organizations_multiple"
                        outputField: "ner_organizations"
                        separator: "|"
                    }
                }
                {
                    split {
                        inputField: "classification_labels_multiple"
                        outputField: "classification_labels"
                        separator: "|"
                    }
                }
                {
                    split {
                        inputField: "lda_topics_multiple"
                        outputField: "lda_topics"
                        separator: "|"
                    }
                }


# This command deletes record fields that are unknown to Solr
# schema.xml. Solr throws an exception on any attempt to load a
# document that contains a field that is not specified in schema.xml.
{
sanitizeUnknownSolrFields {
# Location from which to fetch Solr schema
solrLocator : ${SOLR_LOCATOR}
}
}

# convert timestamp field to native Solr timestamp format
# such as 2012-09-06T07:14:34Z to 2012-09-06T07:14:34.000Z
{
convertTimestamp {
field : created_at
inputFormats : ["unixTimeInMillis"]
inputTimezone : UTC
outputFormat : "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
outputTimezone : UTC
}
```

```
}
{
logTrace {
format : "output record: {}", args : ["@{}"]
}
}
]
}
]
```