Final Report – CS 5604 Fall 2016

Solr Team

CS 5604: Information Storage and Retrieval

Instructor: Dr. Edward A. Fox

Liuqing Li, Anusha Pillai, Ke Tian, Ye Wang
{liuqing, anusha89, ketian, yewang16} @vt.edu

Virginia Polytechnic Institute and State University
Blacksburg, VA, 24061

December 7, 2016

List of Contents

List of Tables

List of Figures

1 Abstract

Researchers make full use of tweet and webpage data in various research areas, such as topic modeling, natural language processing (NLP), text mining, social networks, etc. In 2012, more than 100 million users posted 340 million tweets every day [1]. Meanwhile, people requested about 1.6 billion searches per day [2]. It is necessary to design some strategies to store and retrieve such big data.

Based on the Hadoop Cluster in the Digital Library and Research Laboratory (DLRL) and the previous work in Spring 2016, our team improved the general search infrastructure supporting the IDEAL and GETAR projects, both supported by NSF. The main responsibility was to configure the Basic Indexing and Incremental Indexing (Near Real Time, NRT Indexing) for tweets and web page collections in DLRL's Hadoop Cluster. The goal of Basic Indexing was to index the big collection that contains more than 1.2 billion tweets. The idea of NRT Indexing was to monitor real-time changes in HBase and update the Solr results as appropriate. The main motivation behind the Custom Ranking was to design and implement a new scoring function to re-rank the retrieved results in Solr. Based on the text similarity, a basic document recommender was also created to retrieve the similar documents related to a specific one. Finally, new well written manuals could be easier for users and developers to read and get familiar with Solr and its relevant tools.

Throughout the semester we closely collaborated with the Collection Management Tweets (CMT), Collection Management Webpages (CMW), Classification (CLA), Clustering and Topic Analysis (CTA), and Front-End (FE) teams in getting requirements, input data, and suggestions for data visualization. In order to accomplish our first goal, we worked with the CMT, CMW and FE teams to coordinate the columns in HBase and fields in Solr. We created multiple field types instead of some simple type for the real scenario. To achieve the second goal, we read the report from the previous Solr team and the official tutorials for NRT indexing, and deployed the functionality into the Hadoop Cluster. For the custom ranking and document recommender, we learnt more about the Solr plugins (e.g., request handler, search component) and built our own plugins to accomplish the task. To make a readable document, we recorded our operations step by step and described each step in detail for users and developers.

Our work focuses on Solr, as well as other important components such as HBase, Lily Indexer, and Morphline. Beyond these, there are also some relevant components including Hadoop Distributed File System (HDFS) and Zookeeper. Our Solr team will present part of these components later in this report.

2 Overview

2.1 Management

Great management is the key to success, and the key to great management involves planning and communication. To achieve success, we have created a roadmap that explains our short and long term goals. We have decided to meet once per week to discuss our ongoing and future planned activities, our approach towards a problem, and discuss impediments to the project. It is during this time that we brainstorm the best possible design or implementation of a task and we also discuss the best way to address an impediment.

Apart from these, we meet in class twice a week, where we meet with and talk to other teams. Being the Solr team, it is important to be a strong link amongst other teams. It is important to understand the needs of the Front-End team, incorporate their needs by writing efficient queries, index the correct data and ensure the data model or the schema is in line with what is needed and is modelled in such a way that indexing is simple and efficient.

Apart from the group meetings in and out of class, we maintain a Google Drive directory and ensure that everything is documented and placed on the drive. We maintain a document file that contains all the important information and discussions that we have either in person or via emails and chats. We also use Google Hangout to effectively communicate within the group.

We also set up a GitHub repository to maintain our development activities and use it as our initial content management system. The link is shown below:
https://github.com/CS5604SOLR/Dataset_for_Test

Apart from the management and planning activities, we also maintain tutorials that give instructions on how to set up and use different systems and applications like Cloudera VM, upload a dataset into Hadoop and HBase, set up Solr, etc. These have been included in section 8 as a part of the Developer Manual.

2.2 Challenges

Working on a project implementation with a vast number of new technologies we were not familiar with was quite a learning curve for us. Learning Hadoop and the HDFS filesystem, HBase, Solr, Lily Indexer and Morphline was time consuming and difficult initially but became easier with time. Though many technologies were not needed to implement our part of the search engine, we still needed to understand and implement them to help us understand the system better. Since both the collection teams are still working on getting data into HBase, we had to insert test data into HBase. The four main challenges are as follows:

1. Though the Spring 2016 Solr team did some work on the small collection, it is still a great challenge to index the 1.2 billion tweets collection. For the big one, there are more fields and

field types, so we need to take more time to discuss with other teams to create an accurate schema file that is going to be used in the *Integrated Digital Event Archive and Library* (IDEAL) [22] and *Global Event and Trend Archive Research* (GETAR) [23] systems. This might be an iterative process.

2. Since the data in the HBase system will be populated by the Collections and Classification teams we must wait for them to populate the data before we can go ahead and test the modified schema file and indexing on the actual data.

3. Most learning is done using the Virtual Cloudera setup on our local machines. Due to memory and processing power constraints we faced many issues in setting the VM up and ensuring it worked fast without crashing.

4. The previous team described the custom ranking and document recommendation in their report, but there are still some typos and we cannot follow their steps or run their codes correctly. In this case, it took more time for us to go over the official documents instead of their report.

2.3 Solution Developed

Most of the work that we have done revolves around learning the different technologies and implementing a few small projects by referring to many online resources that include the web documentation provided by Apache and tutorials provided by *Java code geeks* [28]. We have also been using the book *Solr in Action* [4] to help us better understand Apache Solr architecture and how to implement scalable search using it. After gaining knowledge and confidence, we implemented a few hands-on examples that helped us build a strong foundation towards the final design for the project. Some of the tasks that we carried out are:

1. Learning the basics and implementing small projects based on technologies like Hadoop, HBase, Solr, and Lily Indexer to get hands on experience and gain confidence. Each of us set up a VM in our local machines that we used to play around with and gain experience on various technologies used. We uploaded the small collection data file into HDFS and HBase, created a Solr collection and indexed the collection using the Lily Indexer.

2. We are now building search queries to retrieve data. This is configured in *solrconfig.xml*. The request handlers that we define in this file can be used by the Front-End team. We have also started working on the schema file for the IDEAL and GETAR systems.

3. Since our work is based on the work of last semester's team, we first need to follow what they had completed and then start working on our detailed tasks. These tasks include the modification of the schema file to include additional fields, the big dataset indexing, incremental indexing, custom ranking, document recommendation and implementation of faceted search.

4. Since we are building a small portion of the entire IDEAL and GETAR systems, we are understanding and learning the infrastructure in-depth so that we can provide a good design.

2.4 Future Work

2.4.1 Search

This semester, we overwrote the default MoreLikeThis request handler for the basic document recommendation. It is a good approach to get familiar with the multiple request handlers in Solr. Later, we plan to customize more request handlers to meet the search requirements of users, such as custom weighting, results with specific fields and profanity filter.

2.4.2 Custom Ranking

Our current custom ranking function is a simple one. In the near future, our Solr team plans to customize more search components based on more features such as the topic or cluster probabilities. Now, we are able to write our own Java code to read various field values in HBase. In this case, by creating or modifying the search components, more re-ranking tasks could be done during the whole procedure.

2.4.3 Document Recommendation

The document recommendation can be divided into two types, which are the textual similarity based recommendation and the collaborative filtering. As mentioned above, currently, we accomplished the first approach by leveraging the MoreLikeThis plugin. As a result, it is possible to retrieve more similar documents with a given document ID. For the future work, conceptual similarity based recommendation that is a deeper version of the textual similarity based recommendation could be implemented with the information provided by the CTA team. Moreover, user logs can also be used to do the collaborative filtering job.

2.4.4 Solr

At the beginning of this semester, we planned to deploy a two-node Solr server to address the poor response time from Hue. Fortunately, the FE team successfully connected Blacklight with Solr so that the results can be retrieved in a short time at present. However, the FE team didn’t test their efficient tool on the big dataset. Besides this, our data is increasing over time, so we have to face the time cost problem soon. In this case, we still need to figure out SolrCloud and multiple Solr nodes in Cloudera Search, and finally deploy a multiple-node Solr server in the Hadoop Cluster.
Detailed user and developer manuals should also be well-written to describe the clear steps for them to read and use.

3 Literature Review

In this literature review, we discuss both Solr basics and the achievements of the Spring 2016 Solr team. We extend their work by adding more functionalities (e.g., similarity and recommendation). We read their report thoroughly and point out the limitations of their project (e.g., the absence of query recommendation). We also take points from their report for better finding relevant documents.

The textbook *Introduction to Information Retrieval* [2] provides a general overview on information retrieval systems. The concepts from the textbook provide us a solid theoretical background on key concepts of information retrieval systems. For example, in chapter 4, we learn how to construct indexes by using blocked sort-based indexing and single-pass in-memory indexing. These indexing techniques help us understand Solr indexing more easily. In chapter 6, we learn scoring, term weighting and the vector space model. These concepts help us to build our document vector model and similarity function. Using the vector space model, we could transform documents into vectors. Based on the relevance scores, we could compute the similarities between two vectors.

To specifically focus on our project, we need to read more relevant materials on Solr [3]. Solr, which is maintained by the Apache Software Foundation, provides powerful distributed indexing, replication and load-balanced querying. To obtain a better understanding of Solr, we read *Solr in Action* [4] and *Lucene in Action* [5]. Particularly, *Solr in Action* [4] provides the background knowledge and necessary techniques to build the Solr environment and run Solr successfully. We also read *Solr Reference Guide 6.2* [6], which is released on 13 September 2016, to get the newest update on Solr. The Solr reference guide provides detailed instructions on installing Solr and schema file configurations. We also refer to the Solr Wiki [7] for Solr configurations and features. Apache Lucene [8] is a high-performance, full-featured text search engine library written entirely in Java. Solr is built on top of Apache Lucene. The book *Lucene in Action* [5] covers the architecture description of Lucene. Apache Lucene provides us feasibilities to manipulate the Solr framework and customize new features using Solr.

To set up the Solr environment, we utilize Cloudera [9] to get started with launching the Solr environment. Cloudera is a commercial cloud-based company, which provides a well-integrated virtual machine with Hadoop [10], HBase [11,12], ZooKeeper [13,14] and Lily Indexer [15]. By carefully reading the manual from Cloudera, we are able to reproduce the indexing using Solr in the virtual machine.

The Spring 2016 Solr team has already set up the Solr environment of Solr in the DLRL Hadoop cluster with Hue UI components [16]. Their project utilized the Solr framework to index a large set of tweets and webpages. Our project aims to extend the S16 work by adding query similarity and recommendation. We plan to deploy similarity computation based on Apache Lucene similarity APIs [17]. We also plan to deploy recommendation in Solr based on Solr-recommender [18].

4 Requirements

To compare with the previous team's work, we create our requirements for this semester. Table 1 shows the detailed requirements.

Table 1 Solr requirements in Fall 2016

	Spring 2016	**Fall 2016**
schema.xml	Coarse grained	Fine grained
	No copyfields	Copyfields for all fields search
	Create stopwords.txt & profanity.txt	Update the two files
morphlines.conf	Two field types: string and text	Multiple field types
	Field "time" => string	Field "time" => datetime
	No multiple-valued fields	Multiple-valued field parser
Basic Indexing	Small collection	1.2 billion tweets dataset
Incremental Indexing	Virtual Cloudera (VC)	VC & Hadoop Cluster (HC)
Recommendation	Brief description	Implemented in VC & HC
Custom Ranking	Brief description	Implemented in VC & HC
Solr Admin UI	Brief description	Detailed description
	Limited faceted search	Detailed faceted search

Some of the requirements are explained as follows:

1. Modify the schema file (working with CMT, CMW, FE). We build on the work of last semester's Solr team, so will modify the schema file for indexing. Because we will index the columns in HBase provided by the CMT, CMW, CLA and CTA teams, we need to know how they store the data in HBase, and they need to know how we will use the data. The FE team will build an interface based on our indexing schema, so the FE team and our team also need to cooperate with each other closely.

2. Improve the incremental indexing method. Last semester's Solr team only implemented incremental indexing in the virtual machine and did not implement it in the real Hadoop cluster. We need to implement it in the real cluster. In order to make it suitable for this semester's HBase columns, we need to modify their schema file. We must also improve the incremental strategy. For example, we can update the indexes only after several changes have been made, and not re-index so frequently, or we can index the frequently queried fields first.

3. Improve the current scoring function. The results are based on the scoring function. In order to make the recall and the precision of results better, we will improve the current scoring function. We hope that the results can be very relevant to what users are thinking.

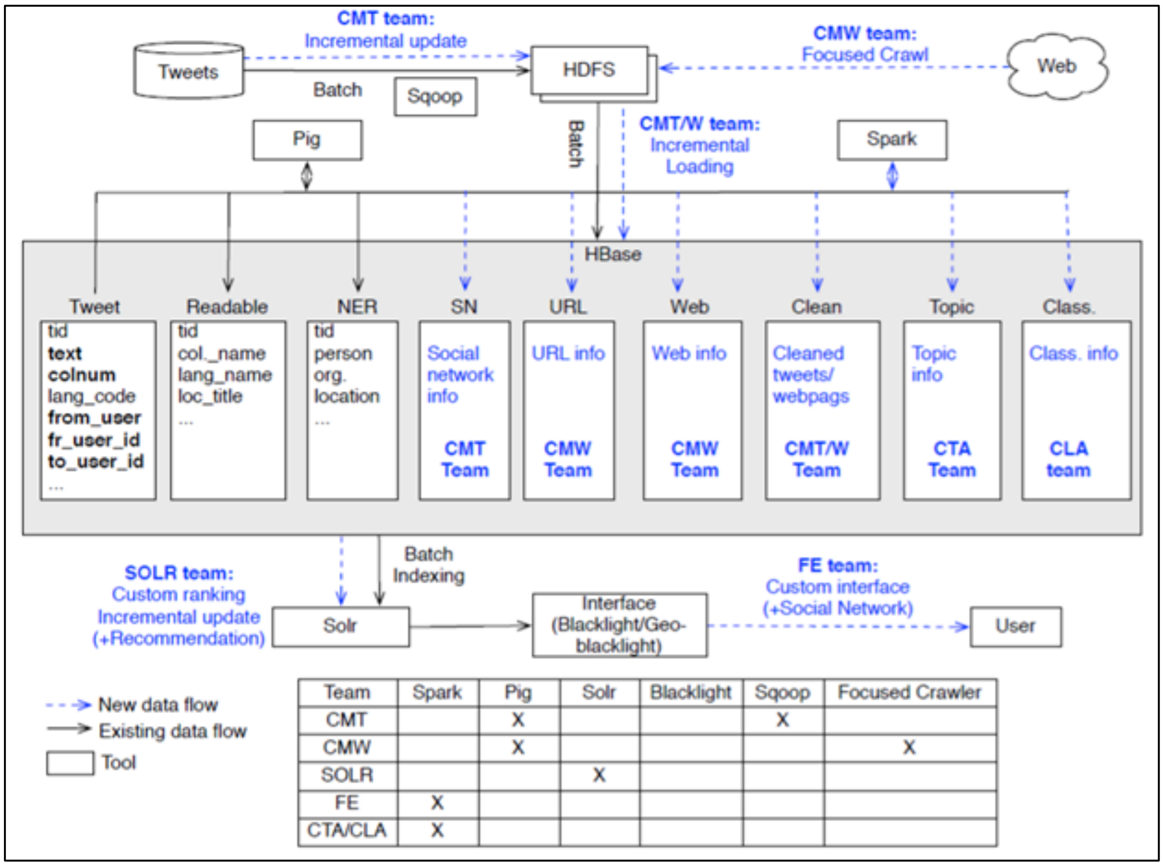
4. Design and implement a recommendation service. We hope that when a user makes a query, the user can see several related queries. We can use methods like query expansion to implement this feature.

5. Work with FE to connect Blacklight with Solr in Cloudera. The FE team focuses on Blacklight, and we will work together with them to ensure that Blacklight can make use of the APIs that Solr provides.

5 Design

An overview of the design of the IDEAL [22] and GETAR [23] systems was presented by Prof. Fox and our GRA. We have understood that as a part of the system some initial work has been completed towards spotting events, but currently the selection of events and the setting up of collection methods is manual. We will build our system over the already implemented infrastructure and design. Also from various discussions, we have understood the different parts of this system and have also understood the significance of each. We first look at the current design followed by our implementation changes.

5.1 General Design



Team	Spark	Pig	Solr	Blacklight	Sqoop	Focused Crawler
CMT		X			X	
CMW		X				X
SOLR			X			
FE	X					
CTA/CLA	X					

Figure 1 Architecture of IDEAL and GETAR [24]

Figure 1 shows the architectural diagram of the IDEAL and GETAR infrastructure [24]. The legend in the bottom left corner tells us the implemented parts and the new components that were expected to be implemented this semester. Here, the data will be managed in HBase as column families and columns, and updated with certain additional fields like the organization, posting date, location of the tweets, etc. Solr then indexes the documents as a batch process using the Lily Batch Indexer.

5.2 Detailed Design

Figure 2 shows the brief architecture of Solr, which also presents the data flow during the whole process. The CMT, CMW, CLA and CTA teams will collect, clean, and process the tweet and webpage collections. The data of these collections will be imported into HDFS and shown in HBase. Our Solr team will configure the two key files (i.e., schema.xml and solrconfig.xml) in Solr to design our own tasks.

For the schema.xml, we leverage it to do the basic indexing and incremental indexing tasks. During this period, to connect HBase with Solr, Lily HBase Indexer has to be applied to map the columns in HBase into the fields in Solr.

For the solrconfig.xml, we create new plugins like request handlers and search components that can be used for document recommendation and custom ranking. We will explain these plugins in Chapter 8.

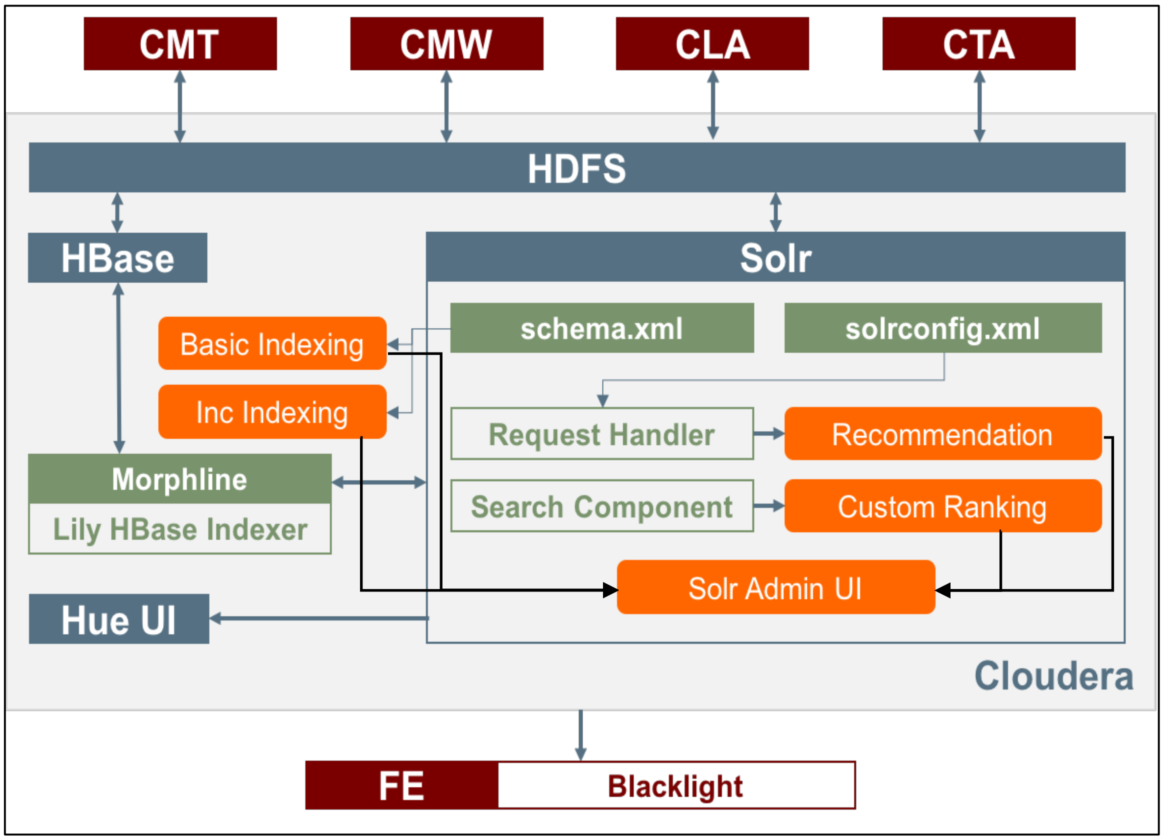


Figure 2 Architecture of Solr

5.2.1 Basic Indexing

The key point of basic indexing is to generate the proper schema.xml. The HBase structure has already been established during the class (see **Appendix A**). As mentioned above, the structure and data can be treated as input for our Solr team. Indexing cannot be performed on all the data stored in HBase. We need to decide which fields should be indexed or shown to the end-user. We have worked with all the other teams, especially the Front-End team, to finalize this. Later, we updated the schema file based on the HBase structure and the UI requirements. Table 2 shows the fields and attributes of the schema file.

Table 2 Fields and Attributes of schema.xml

Indexed Field Name	Field Type	Stored	Multiple Values
collection_name_s	string	Yes	No
archive_source_s	string	Yes	No
source_s	string	Yes	No
screen_name_s	string	Yes	No
created_time_dt	date	Yes	No
language_s	string	Yes	No
text_t	text_general	Yes	Yes
location_s	string	Yes	Yes
t_year_i	int	Yes	No
t_month_i	int	Yes	No
hashtags_s	string	Yes	Yes
mentions_s	string	Yes	Yes
classification_s	string	No	No
events_s	string	Yes	Yes
sner_people_s	string	Yes	Yes
sner_org_s	string	Yes	Yes
sner_loc_s	string	Yes	Yes
t_importance_f	float	Yes	No

title_s	string	Yes	No
author_s	string	Yes	Yes
sub_urls_s	string	Yes	Yes
organization_s	string	Yes	No
fetched_time_dt	date	Yes	No
w_importance_f	float	Yes	No
doc_type_s	string	Yes	No
label_list_s	string	Yes	Yes
probability_list_f	float	Yes	Yes
cluster_label_s	string	Yes	Yes
doc_probability_f	float	Yes	Yes

This was shared with the other teams so that they ensure there is a field in HBase that is populated with the correct value as required by our team or the FE team. We also had a discussion with our GRA about the basic indexing job for the real datasets. Since our current schema file is much more precise than the previous one, there may have been some improvements during the indexing process. Based on our knowledge, we have also made some field type modifications to fit for the data format to be used by our custom ranking function and the document recommendation.

5.2.2 Incremental Indexing

For the big dataset, it will take a great amount of time for basic indexing, which could not solve the problem of frequent inserts, deletes, and updates, so we need to find a better way to monitor the changes of the cells in HBase, and then update the results in Solr immediately.
After reading the documents from Cloudera, we notice that Cloudera provides a special indexer named Lily HBase NRT Indexer. It processes a continuous stream of HBase cell updates into the live search index. In this case, it makes the incremental indexing much easier, because our team just needs to follow the tutorials to configure the indexer service. The detailed process will be described in Chapter 8.

5.2.3 Custom Ranking

By default, when user sends a search query, the retrieved results come back in default TF-IDF order. Consequently, we need to modify the scoring strategy to meet our own needs or retrieve more reasonable results. Because the CMT, CMW, CLA, and CTA teams can pre-process the data, we may leverage their outcomes to customize the ranking function.

$$\boldsymbol{Score = Doc_{score,Solr} + Doc_{importance}}$$
$$+ W_{topic} \times Doc_{score,topic} + W_{cluster} \times Doc_{score,cluster}$$

The above formula presents our design. The new score is the sum of four different scores, which are the default Solr score, the importance score from the collection teams, and two scores from the CTA team. Unfortunately, we didn’t have enough time to do the parsing job and get those values from HBase; the final implementation is based on the top two scores. Definitely, we are able to re-rank the query results and show the differences versus the original results.

5.2.4 Recommendation

Based on the query search by the user, a recommendation system can recommend similar tweets or webpages. This can be done based on similarity of content or can be based on the user click history.
Item based recommendation means the document features are used to calculate the similarities. The similarities between a document and the other documents in the collection are computed and the top k similar documents are identified. The ranks of the documents are computed and returned. This has been implemented using the MoreLikeThis functionality of Solr.
User based recommendation means the user’s browsing history is extracted using the log file. We can then compute the similarity between the responses of the users. Based on the user-user similarity, we can rank the documents that are viewed by the top k users and display them as recommendations.

6 Implementation

6.1 Overview

We have had several discussions with Prof. Fox, our GRA and other teams during the whole procedure to get a better understanding of our tasks. Based on the current Hadoop Cluster and previous work, our team is aiming at improving the current infrastructure and adding more functionalities for a better search job.

Our approach will be as follows:
1. Review the previous work. We followed the tutorials and instructions to get familiar with the technologies.
2. Understand the data flow in the current infrastructure. It helped our Solr team know the role in the whole situation and the input and output from other teams.
3. Collaborate with other teams. We worked with other teams to improve the current schema file and clarify how the column families and columns are stored in HBase.
4. Basic Indexing. Index the test collection and the 1.2 billion tweets dataset in either live mode or batch mode.
5. Incremental Indexing. Deploy the Lily HBase NRT Indexer service on both Virtual Cloudera and the Hadoop Cluster.
6. Custom Ranking. Modify the default ranking method and establish our own custom ranking on both Virtual Cloudera and the Hadoop Cluster.
7. Document Recommendation. Deploy the MoreLikeThis plugin to accomplish the basic recommendation task on both Virtual Cloudera and the Hadoop Cluster.

6.2 Timeline

The timeline of our Solr team is shown below, including the task, duration, current status and responsible member for accomplishment. Moreover, milestones and deliverables are also listed in Table 3, that may be beneficial for other teams' work.

Table 3 Timeline of Solr Team

#	Week		Task	Status	Assigned To
1M	1	08/23 - 08/26	Set up Solr team	Done	All
2M	1	08/23 - 08/26	Set up Solr on local machine and do tutorial	Done	All
3	1	08/23 - 08/26	Set up share folder using Google Drive and Hangouts for instant messaging	Done	Liuqing

4M	2	08/29 - 09/02	Set up a Cloudera VirtualBox VM	Done	All
5	2	08/29 - 09/02	Review the existing schema file	Done	All
6	3	09/05 - 09/09	Prepare for team presentation	Done	All
7	3	09/05 - 09/09	**[Virtual Cloudera**] Import the original small collection into HBase	Done	Anusha, Ke
8	4	09/12 - 09/16	**[Virtual Cloudera**] Create a Solr instance for the original small collection	Done	Anusha, Ke
9M	4	09/12 - 09/16	**[Virtual Cloudera]** Use Lily Indexer to map columns in HBase with fields in Solr and make a basic query test (Live Mode)	Done	Liuqing
10	4	09/12 - 09/16	**[Hadoop Cluster**] Import the original small collection into HBase	Done	Liuqing
11	4	09/12 - 09/16	**[Hadoop Cluster**] Create a Solr instance for the original small collection	Done	Liuqing
12D	4, 5	09/12 - 09/23	Share the previous schema file with relevant teams for modification	Done	All
13D	5	09/19 - 09/23	**[Hadoop Cluster**] Use Lily Indexer to map columns in HBase with fields in Solr for FE team future test (Live Mode)	Done	Liuqing
14	6	09/26 - 09/30	Update the stoplist.txt and profanity.txt	Done	All
15	6	09/26 - 09/30	Divide our team into two subgroups for the NRT and recommender tasks	Done	All
16M	6	09/26 - 09/30	**[Virtual Cloudera**] Use Lily Indexer to accomplish the offline indexing job (Batch Mode)	Done	All

17M	7	10/03 - 10/07	**[Virtual Cloudera**] Use Lily Indexer to accomplish the incremental indexing job (NRT)	Done	Liuqing, Ke
18	6, 7, 8	09/26 - 10/14	Learn the knowledge of faceted search and recommender	Done	Anusha, Ye
19M	8	10/10 - 10/14	**[Virtual Cloudera]** Modify the schema file to make a basic facet search query	Done	Ke
20D	8	10/10 - 10/14	**[Hadoop Cluster**] Use Lily Indexer to accomplish the incremental indexing (NRT)	Postponed (No. 29)	Liuqing
21M	9	10/17 - 10/21	Design and Implement a basic recommender	Done	Anusha, Ye
22	10	10/24 - 10/28	Learn the knowledge of custom ranking function	Done	Liuqing
23	10	10/24 - 10/28	Further work on facet search	Done	Ke
24D	11	10/31 - 11/04	Create the schema.xml based on the HBase structure	Done	All
25	12	11/07 - 11/11	Further work on recommender	Done	Anusha, Ye
26	12	11/07 - 11/11	Build custom ranking handler	Done	Liuqing, Ke
27M	12, 13	11/07 - 11/18	Index the big dataset (raw tweets)	Done	All
28	13	11/14 - 11/18	Deploy the MoreLikeThis handler and custom ranking component	Done	All
29	14	11/21 - 11/25	**[Hadoop Cluster**] Use Lily Indexer to accomplish the incremental indexing (NRT)	Done	Liuqing
30	14	11/21 - 11/25	Learn about SolrCloud	Done	Anusha, Ye, Ke
31M	15	11/28 - End	Index the big dataset (raw tweets and processed data)	Done	Liuqing
M – Milestones **D** – Deliverables					

7 User Manual

In this section, we describe how a user without comprehensive background on Solr could utilize Solr for query and search. How to use Solr for indexing and querying is not intuitive. The purpose of this user manual is to provide more intuitive examples for users to work with Solr.

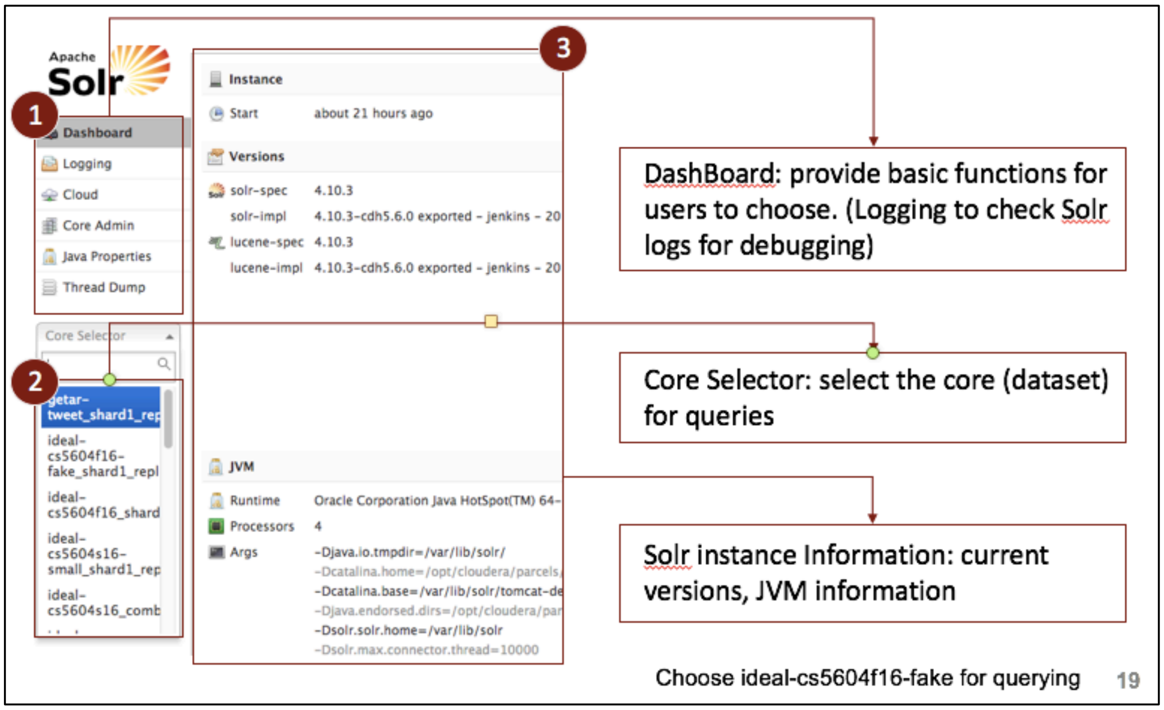
7.1 Solr Admin Interface

The following figure presents a Solr admin user interface. The Solr admin interface is the main user interface when users interact with Solr. Particularly, Solr provides a list of features in a Web interface. These features make it easy for Solr users to view Solr configuration details, run queries and analyze document fields. However, understanding the Web interface is not an easy problem, especially for non-savvy users. In this section, we provide detailed examples and descriptions on how to use the Solr Web interface.

Figure 3 shows the main Solr Web interface. Table 4 provides basic functions and detailed descriptions on monitoring the Solr interface, i.e., logging, Cloud, Core Admin, Java Properties, and Thread Dump. These basic properties provide useful functions to facilitate the management of the running Solr instance.

Table 4 Dashboard Functions with Explanations

Dashboard	Description
Logging	Showing Solr logs for debugging. The Logging page shows recent messages logged by this Solr node.
Cloud	Status information about each collection & node in the cluster
Core Admin	Provides some basic functionality for managing users' collections.
Java Properties	Easy access to one of the most essential components of a top-performing Solr system. Users can see all the properties of the JVM running Solr.
Thread Dump	Inspect the currently active threads on your server. Each thread is listed and access to the stacktraces is available where applicable.



```
Start         about 21 hours ago
solr-spec     4.10.3
solr-impl     4.10.3-cdh5.6.0 exported - jenkins - 20
lucene-spec   4.10.3
lucene-impl   4.10.3-cdh5.6.0 exported - jenkins - 20
Runtime       Oracle Corporation Java HotSpot(TM) 64-
Processors    4
Args          -Djava.io.tmpdir=/var/lib/solr/
              -Dcatalina.home=/opt/cloudera/parcels/
              -Dcatalina.base=/var/lib/solr/tomcat-de
              -Djava.endorsed.dirs=/opt/cloudera/par
              -Dsolr.solr.home=/var/lib/solr
              -Dsolr.max.connector.thread=10000
```

DashBoard: provide basic functions for users to choose. (Logging to check Solr logs for debugging)

Core Selector: select the core (dataset) for queries

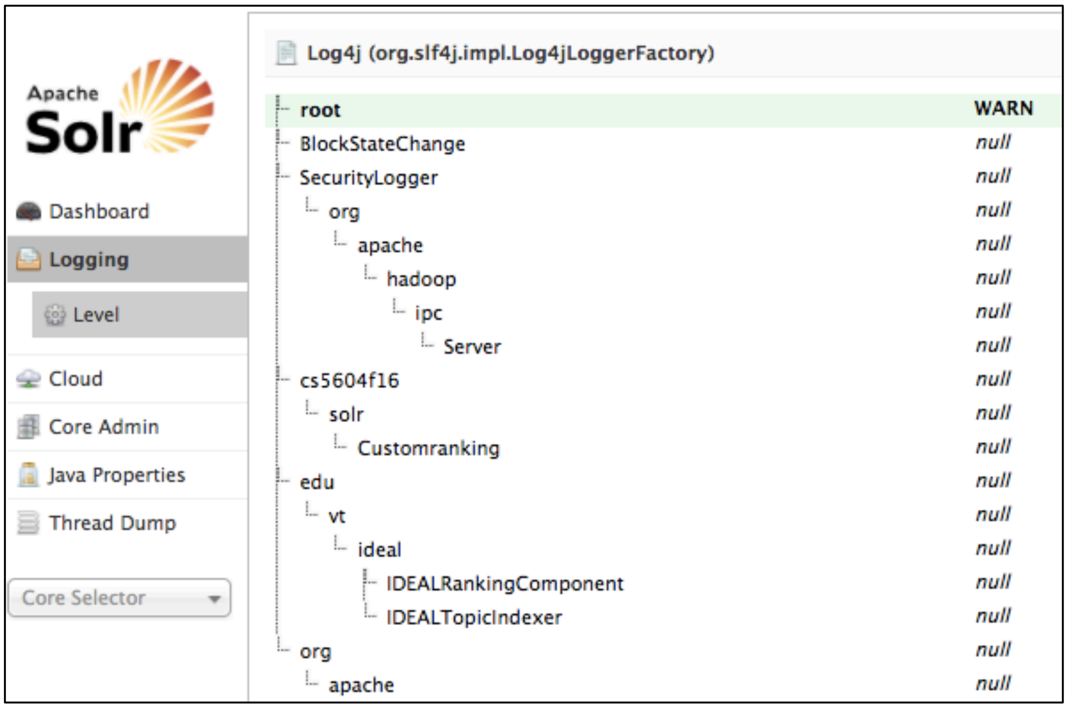
Solr instance Information: current versions, JVM information

Choose ideal-cs5604f16-fake for querying

Figure 3 Solr Admin UI

Under each basic function, users could explore more detailed information about the current state. For example, Figure 4 shows the logs with different levels. The log level starts with the root and follows the hierarchy. The logs in different levels help users quickly identify the suspicious component if the system fails. For example, in our project, we particularly focus on the logs from the custom ranking. We are able to quickly identify the custom ranking logs by following the path root->cs5604f16->solr->Customranking.

Figure 5 shows the radial graph of nodes. The "Graph (Radial)" option provides a different visual view of each node. The visualized graph provides a more direct impression on the dependence relations between different nodes. With the different colors, a user could easily identify the current status of the node. For example, the node with green color demonstrates that the node is currently active, the node with yellow color demonstrates the node is currently down. The user could restart the Solr instance to recover the node.



```
Log4j (org.slf4j.impl.Log4jLoggerFactory)

root                                  WARN
BlockStateChange                      null
SecurityLogger                        null
  org                                 null
    apache                            null
      hadoop                          null
        ipc                           null
          Server                      null
cs5604f16                             null
  solr                                null
    Customranking                     null
edu                                   null
  vt                                  null
    ideal                             null
      IDEALRankingComponent           null
      IDEALTopicIndexer               null
org                                   null
  apache                              null
```

Figure 4 Solr Logging Layout

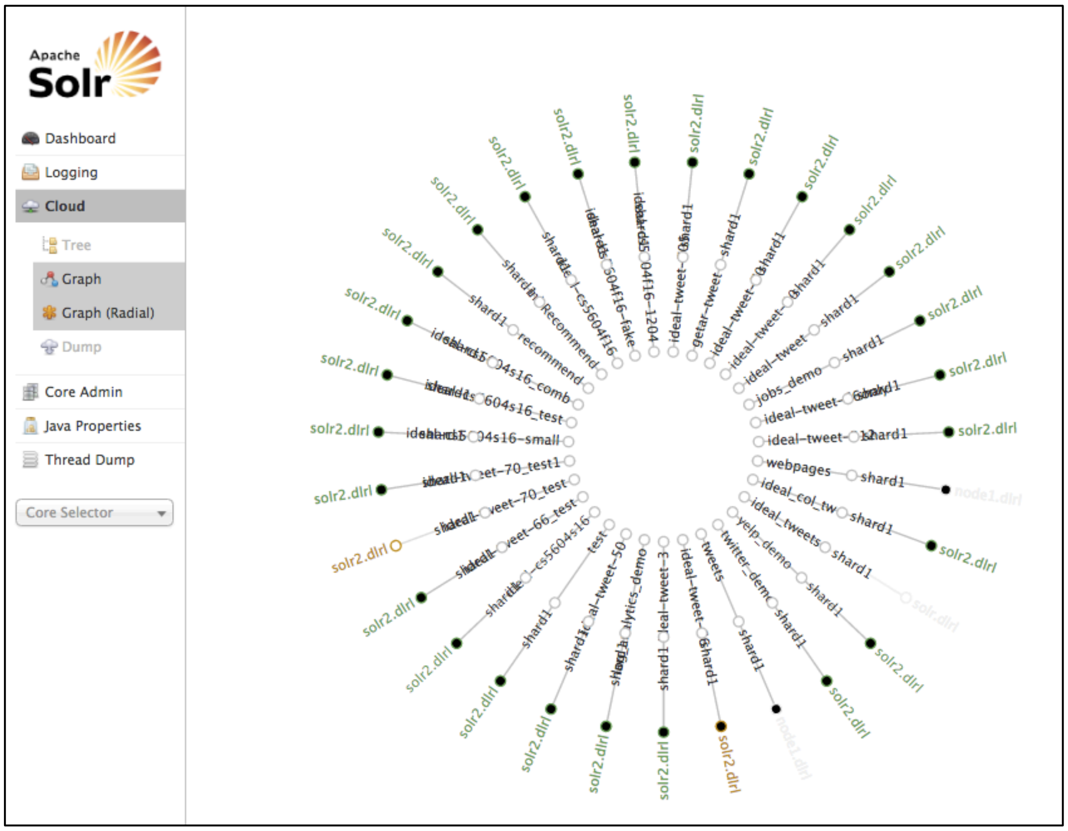


Figure 5 Solr Graph Radial in the Cloud Option

7.2 Solr Query

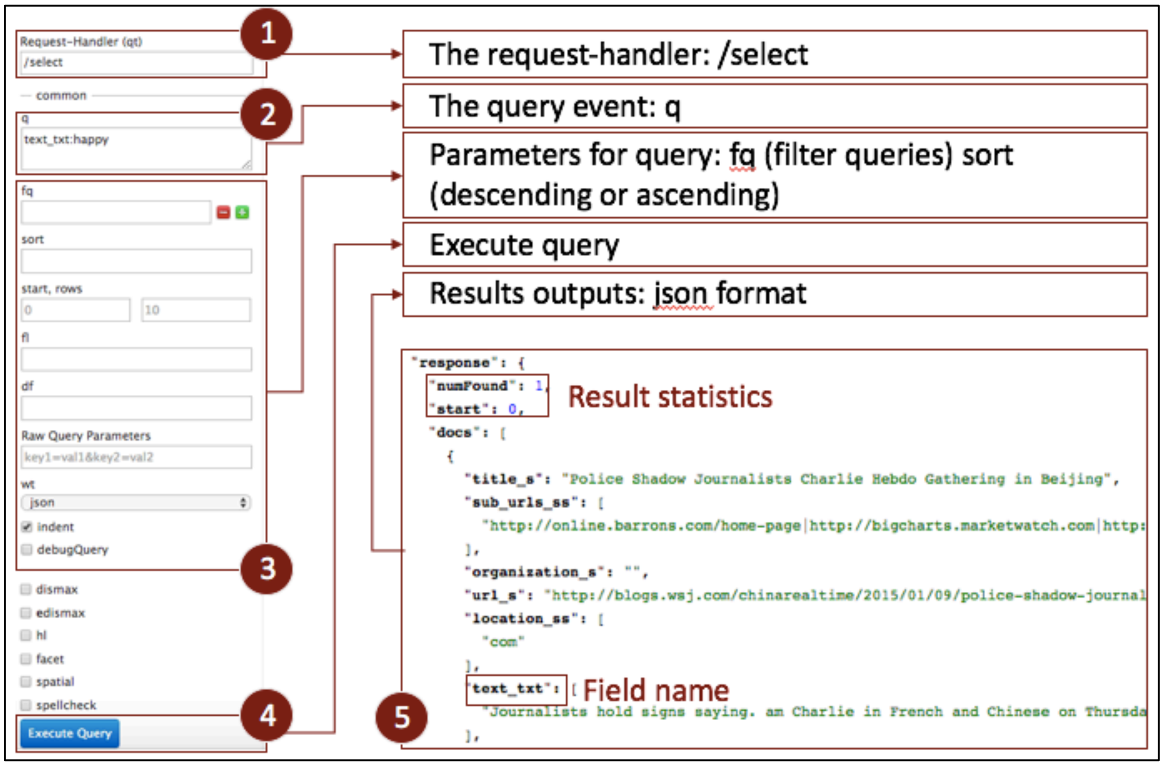
Using Solr for indexing and querying is an important functionality provided by the Solr admin UI. To efficiently utilize the Solr admin interface, we need to understand the meaning of each parameter. Table 5 presents the query parameters and the corresponding descriptions.
Figure 6 presents the admin query interface for users. The parts labeled 1, 2, 3, and 4 relate to the user specified query command.
In Figure 6, the user specifies "/select" as the request handler, which is used to select items from the indexed collection. We choose the query as text_txt:happy, which is to find the keyword "happy" in the field named as text_txt. We choose wt as json, which means the query output is in the json format. The complete request is shown in the following link:

http://localhost:8983/solr/ideal-cs5604f16
fake_shard1_replica1/select?q=text_txt%3Ahappy&wt=json&indent=true

In the result (part 5), we obtain how many documents (doc) are found in the result, giving statistics. For each document, we are also able to check the content in each field.

Table 5 Solr Query Parameters with Descriptions

Query Parameters	Description
Qt	The qt is shorthand for Request Handler. A request handler processes requests coming to Solr.
Q	Query parameters. It must be a query specified in SolrQuerySyntax. e.g., text_txt:happy.
Fq	Fq stands for Filter Query. fq is used to specify a query that can be used to restrict the return set.
Sort	"Sort" is used to rank the returned results based on some principles. Sorting can be done on the "score" of the document.
Star,rows	"Start" is used to paginate results from a query. When specified, it indicates the offset in the complete result set.
Fl	Fl stands for Field List. The fl parameter can be used to specify a set of fields to return, limiting the amount of information in the response
Df	Df stands for the default field
Wt	Wt parameter selects the Response Writer that Solr should use to format the query's response.


1
2
3
4
5
The request-handler: /select
The query event: q
Parameters for query: fq (filter queries) sort (descending or ascending)
Execute query
Results outputs: json format
Result statistics
Field name
common
dismax
edismax
hl
facet
spatial
spellcheck
Execute Query

Request-Handler (qt): /select
q: text_txt:happy
fq:
sort:
start, rows: 0, 10
fl:
df:
Raw Query Parameters: key1=val1&key2=val2
wt: json
[✓ indent]
debugQuery

```
"response": {
  "numFound": 1,
  "start": 0,
  "docs": [
    {
      "title_s": "Police Shadow Journalists Charlie Hebdo Gathering in Beijing",
      "sub_urls_ss": [
        "http://online.barrons.com/home-page|http://bigcharts.marketwatch.com|http:
      ],
      "organization_s": "",
      "url_s": "http://blogs.wsj.com/chinarealtime/2015/01/09/police-shadow-journal
      "location_ss": [
        "com"
      ],
      "text_txt": [
        "Journalists hold signs saying. am Charlie in French and Chinese on Thursda
      ],
```

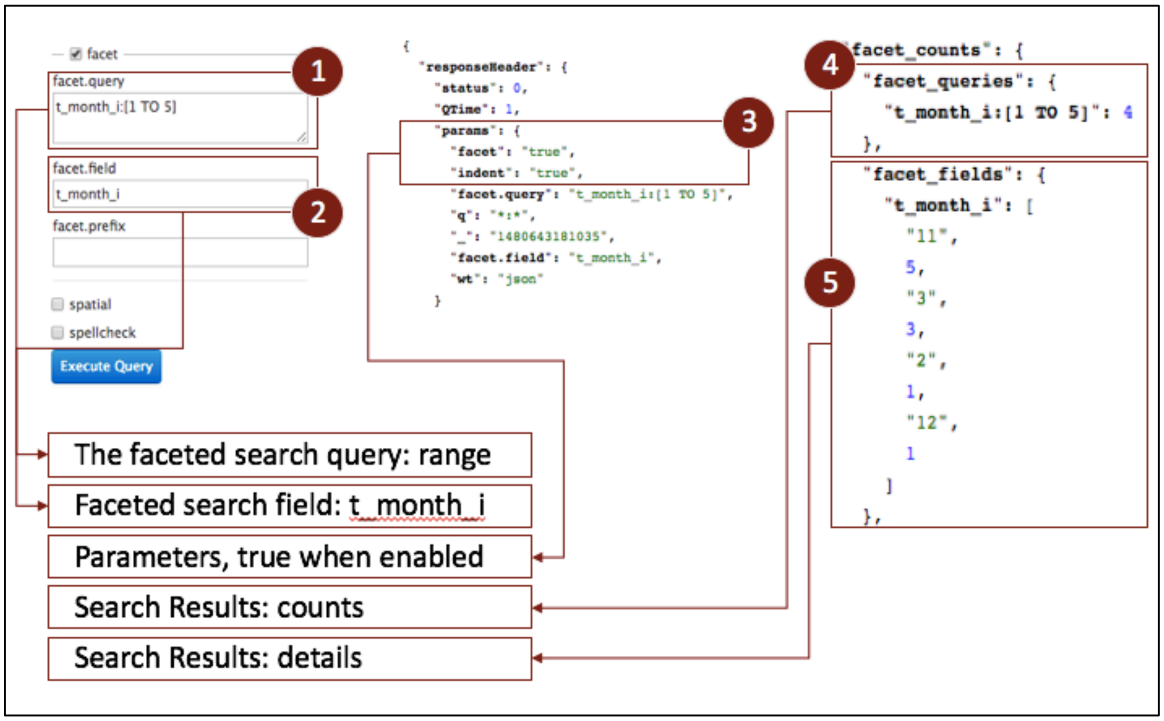
Figure 6 User Guide for Solr Basic Query

7.3 Faceted Search

Faceted Search is a technique for accessing information organized according to a faceted field defined in Solr. Faceted search allows users to explore a collection of information by applying multiple filters. The parameters for faceted search are defined in Chapter 8.

Figure 7 presents the layout for faceted search. In the example, we use faceted search to search for a range where field t_month_i is between 1 and 5. The query specification for faceted search is t_month_i:[1 TO 5]. The faceted field is set as t_month_i. The result in part 4 shows that 4 items match the faceted search requirement; the count of the faceted search result is 4. Part 5 show the detailed statistics about the faceted search result. In the t_month_i field, there are 3 items with t_month_i=3 and 1 item with t_month_i=2. The total count of the range [1,5] is 1+3=4, which is consistent with the search result. The complete faceted search request is shown in the following link:

http://localhost:8983/solr/ideal-cs5604f16-fake_shard1_replica1/select?q=*%3A*&wt=json&indent=true&facet=true&facet.query=t_month_i%3A%5B1+TO+5%5D&facet.field=t_month_i



```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "facet": "true",
      "indent": "true",
      "facet.query": "t_month_i:[1 TO 5]",
      "q": "*:*",
      "_": "1480643181035",
      "facet.field": "t_month_i",
      "wt": "json"
    }
```

```
"facet_counts": {
  "facet_queries": {
    "t_month_i:[1 TO 5]": 4
  },
  "facet_fields": {
    "t_month_i": [
      "11",
      5,
      "3",
      3,
      "2",
      1,
      "12",
      1
    ]
  },
```

Figure 7 User Guide for Solr Faceted Search

7.4 Document Recommendation

We have implemented the textual similarity based document recommendation system. This has been implemented using the MoreLikeThis (MLT) feature provided by Solr. To make use of this we need to make certain configuration changes. The configuration changes to be made have been explained in the developer manual. Here we show how one can make use of the MoreLikeThis handler in Solr for recommendation.

MLT provides us with certain parameters that can be used to configure the way in which we want Solr to perform the recommendation. There are two approaches to recommendation. One is proactive and the other is reactive. The proactive way is implemented using the search handler. In this we enable MLT during search. This provides us with a set of similar items during the search itself. Every item of the search result has a set of items that are like it.

It can be implemented using the same search handler but with more query string parameters to be passed. A search handler can be used as follows:

In general, users use the HTTP API and not the Solr Admin UI. To see the results we will use the Solr Admin UI.

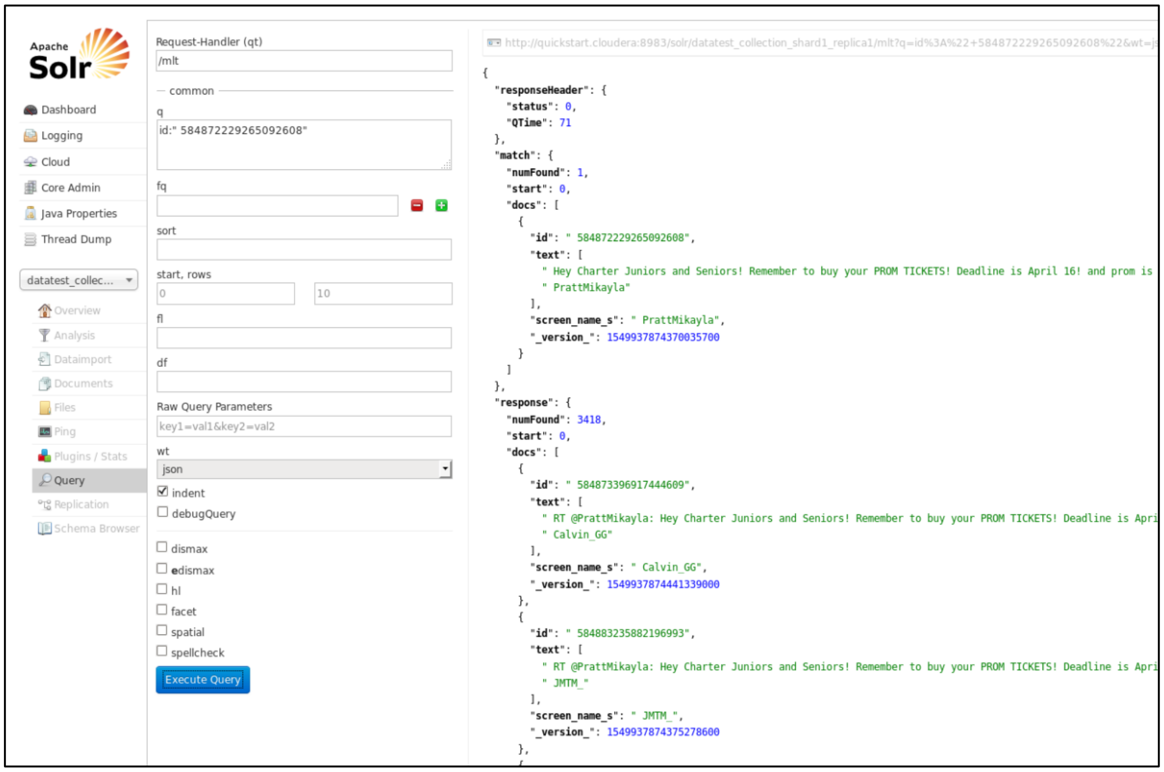
http://quickstart.cloudera:8983/solr/ideal-cs5604f16-fake_shard1_replica1/select?q=Twitter&fl=text_txt&wt=json&indent=true&mlt=true&mlt.fl=text_txt

We make use of the ideal-cs5604f16-fake_shard1_replica1 core. The above URL calls the select handler with a search query “Twitter”. By passing mlt = true we are telling the handler to return a similar set of items for each item in the search result space. This is a little bit more time consuming. Other parameters include mlt.fl = text_txt, which tells the handler to compute similarity based on the text_txt field.

The reactive way is to make use of the mlt handler. This does not provide the user with the similar items when a search query is fired but it issues another search query once the user selects the item for which he/she wants similar items. After issuing a search query then the user will use the ID of the item for which a similar item is needed in the mlt handler as shown below. After issuing a search query we need items that are like the item with the id=584872229265092608, thus we fire the query below to get similar items.

http://quickstart.cloudera:8983/solr/datatest_collection_shard1_replica1/mlt?q=id%3A%22+584872229265092608%22

In the URL, datatest_collection_shard1_replica1 is the core name. %3A and %22 are codes for colon (:) and double quotation mark ("), respectively.



Request-Handler (qt): /mlt
common
q: id:" 584872229265092608"
fq
sort
start, rows: 0 10
fl
df
Raw Query Parameters: key1=val1&key2=val2
wt: json
☑ indent
☐ debugQuery
☐ dismax
☐ edismax
☐ hl
☐ facet
☐ spatial
☐ spellcheck

http://quickstart.cloudera:8983/solr/datatest_collection_shard1_replica1/mlt?q=id%3A%22+584872229265092608%22&wt=js

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 71
  },
  "match": {
    "numFound": 1,
    "start": 0,
    "docs": [
      {
        "id": " 584872229265092608",
        "text": [
          " Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is April 16! and prom is
          " PrattMikayla"
        ],
        "screen_name_s": " PrattMikayla",
        "_version_": 1549937874370035700
      }
    ]
  },
  "response": {
    "numFound": 3418,
    "start": 0,
    "docs": [
      {
        "id": " 584873396917444609",
        "text": [
          " RT @PrattMikayla: Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is Apri
          " Calvin_GG"
        ],
        "screen_name_s": " Calvin_GG",
        "_version_": 1549937874441339000
      },
      {
        "id": " 584883235882196993",
        "text": [
          " RT @PrattMikayla: Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is Apri
          " JMTM_"
        ],
        "screen_name_s": " JMTM_",
        "_version_": 1549937874375278600
      },
```

Figure 8 User Guide for Document Recommendation

In the result, the “match” means the item set matching the given query. The “response” is the set of similar items to the item with ID = 584872229265092608. These also are a few parameters that can be passed along with the handler; see Table 6.

Table 6 MLT Parameters and Descriptions [26]

Parameter	Description	Default Value
mlt.fl	The fields to use for similarity.	N/A
ml.mintf	Minimum Term Frequency- the frequency below which terms will be ignored in the source doc.	DEFAULT_MIN_TERM_FREQ = 2
mlt.mindf	Minimum Document Frequency- the frequency at which words will be ignored which do not occur in at least this many docs.	DEFAULT_MIN_DOC_FREQ = 5
mlt.minwl	minimum word length below which words will be ignored.	DEFAULT_MIN_WORD_LENGTH = 0
mlt.maxwl	maximum word length above which words will be ignored.	DEFAULT_MAX_WORD_LENGTH = 0
mlt.maxqt	maximum number of query terms that will be included in any generated query.	DEFAULT_MAX_QUERY_TERMS = 25
mlt.qf	Query fields and their boosts. These fields must also be specified in mlt.fl	N/A

8 Developer Manual

8.1 Background

In this section, we introduce the relevant background for the developers to know more about the key modules in the architecture of our framework and important components that would be deployed in the architecture. The key modules include HBase [19], Apache Lucene [8], Solr [6] and Lily HBase Indexer [21]. These key modules help us understand the workflow of our system.

8.1.1 HBase

HBase is an open-source, non-relational, column-family-oriented, key-value-based database management system. HBase runs on top of HDFS (Hadoop distributed file system). Typically, an HBase system includes individual tables. Each table in HBase contains rows and columns. A primary key has to be defined in each table. For example, we could use Twitter ID as a primary key for each table in the tweet collections. We could use webpage ID as a primary key for webpage collections. All the operations and access to HBase must use the predefined primary key.

A column in the HBase table represents an attribute of an object. Using the Twitter collection as an example, each row represents a single tweet from one user. The column represents the user ID from the tweets, or the date, or the IP address. HBase allows for many attributes to be grouped together into column families. The column families are widely used in HBase tables. The elements of a column family are all stored together in HDFS.

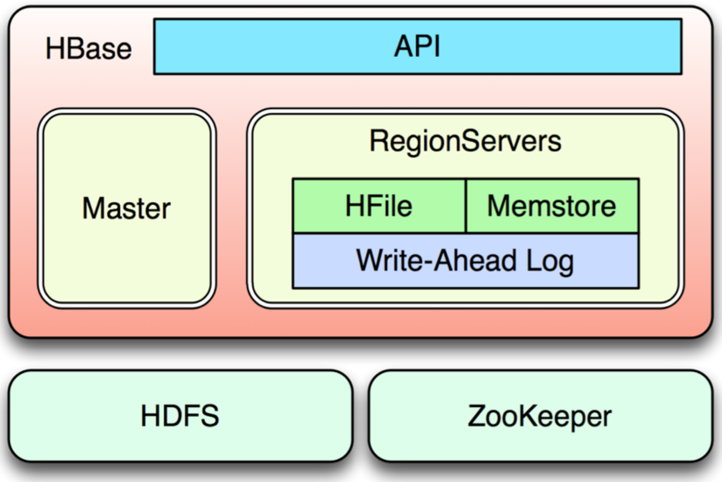


Figure 9 Architecture of HBase

Figure 9 [19] shows the architecture of HBase. HBase is built on top of HDFS, which is a Hadoop based file system. ZooKeeper is a centralized service for system-level configuration. ZooKeeper is used to maintain configuration information and naming. ZooKeeper also provides distributed synchronization as well as group services. Building on top of HDFS, HBase could provide database-related APIs for users to interact with the database.

HBase has two run modes: “standalone” and “distributed”. In standalone mode, HBase uses the local filesystem instead of HDFS. In distributed mode, HBase requires an instance of HDFS. The distributed mode can be further divided into pseudo-distributed- and fully-distributed-mode. Pseudo-distributed-mode allows all the daemons and services running on a single node. Fully-distributed-mode allows all the daemons spread across all the cluster nodes.

8.1.2 Apache Lucene

Apache Lucene is a high-performance, full-featured text search engine library. The Apache Lucene library is implemented in Java, which supports full-text search across different platforms. Our Solr implementation is built on top of Apache Lucene to utilize the existing searching and querying functions. The core of Apache Lucene is called Lucene core, which provides Java-based indexing, search technology and spellchecking. Solr is a standalone pre-configured product, which uses Lucene APIs.

8.1.3 Solr

Apache Solr is an open source enterprise search from the Apache Lucene project. It is scalable and supports flexible search queries (e.g., keyword search and complex query searches). In our project, we will utilize the Solr framework to realize a list of different queries. For example, we could use Solr to achieve keyword matching (e.g., title:foo, title:"foo" AND body:"bar"), wildcard matching (e.g., title:foo*), and proximity matching (e.g., "foo bar"~4). Solr is highly reliable, scalable and fault tolerant. Apache Solr is widely used for indexing.

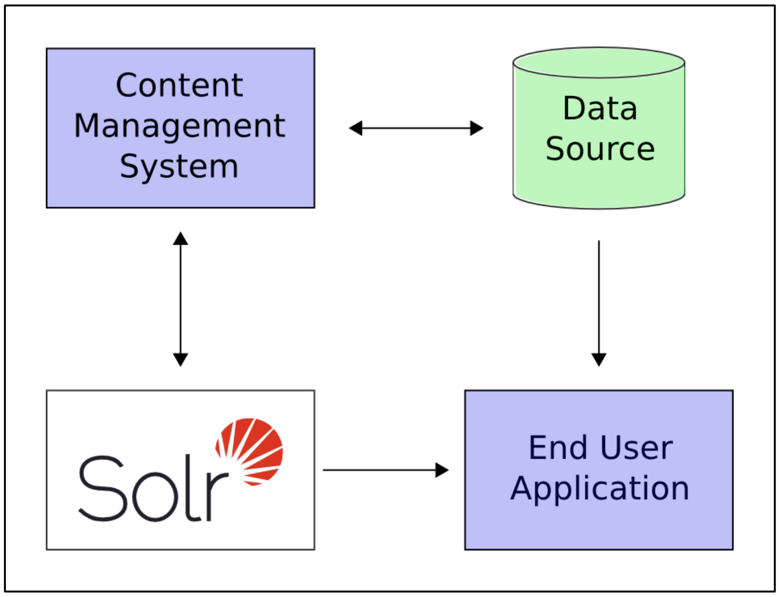


Figure 10 Interaction of Solr [20]

Figure 10 [20] shows a scenario where Solr runs alongside with other server applications. Solr enables interactions with other applications (e.g., end user application and content management application) by following these four steps:

1. Define a schema. The schema file in Solr explains the contents of documents that will be indexed.

2. Deploy Solr based on a particular schema.xml.

3. Feed Solr documents for which users will search.

4. Expose search functionality in a user application.

Moreover, Solr includes two important components that are the request handler and search component, which can be used to customize the functionalities of Solr. The two components will be explained in the following sections.

8.1.4 Request Handler

Every request handler is defined with a name and a class. The name of the request handler is referenced with the request to Solr, typically as a path. The primary request handler defined with Solr by default is the "SearchHandler", which handles search queries. The request handler is defined, and then a list of defaults for the handler are defined with a defaults list.

```
<requestHandler name="/select" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
  </lst>
</requestHandler>
```

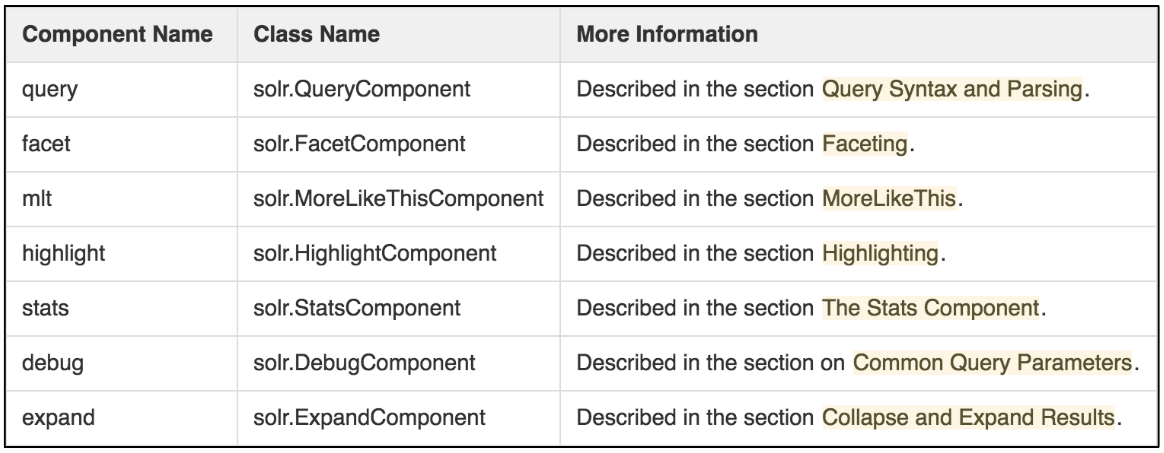
Figure 11 An Example of Request Handler

Figure 11 shows an example of the "select" request handler. It defines the rows parameter, which defines how many search results to return, e.g., "10". The "echoParams" parameter defines that the parameters defined in the query should be returned when debug information is returned. Note also that the way the defaults are defined in the list varies if the parameter is a string, an integer, or another type.

8.1.5 Search Component

Search components define the logic that is used by the SearchHandler to perform queries for users. There are several default search components that work with all SearchHandlers without any additional configuration. Table 7 shows the default search components.

Table 7 Default Search Components [29]



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.

Developers can register their own search components. However, if you register a new search component with one of these default names, the newly defined component will be used instead of the default.

8.1.6 Lily HBase Indexer

The Lily HBase indexer service allows users to query data stored in HBase with Search. The indexer service will index the stream of records being added to HBase tables. By using the Lily HBase indexer, we could easily and quickly index HBase rows into Solr.

Content stored in HBase needs to be indexed before we can search and query. There are two types of indexers declared in Cloudera: Lily HBase Batch Indexer and Lily HBase Near Real-time (NRT) indexer. The Lily HBase Batch Indexer can realize batch index tables in HBase by using MapReduce jobs. On the other hand, the Lily HBase Near Real-Time (NRT) indexer is able to process a continuous stream of HBase cell updates into live search indexers.

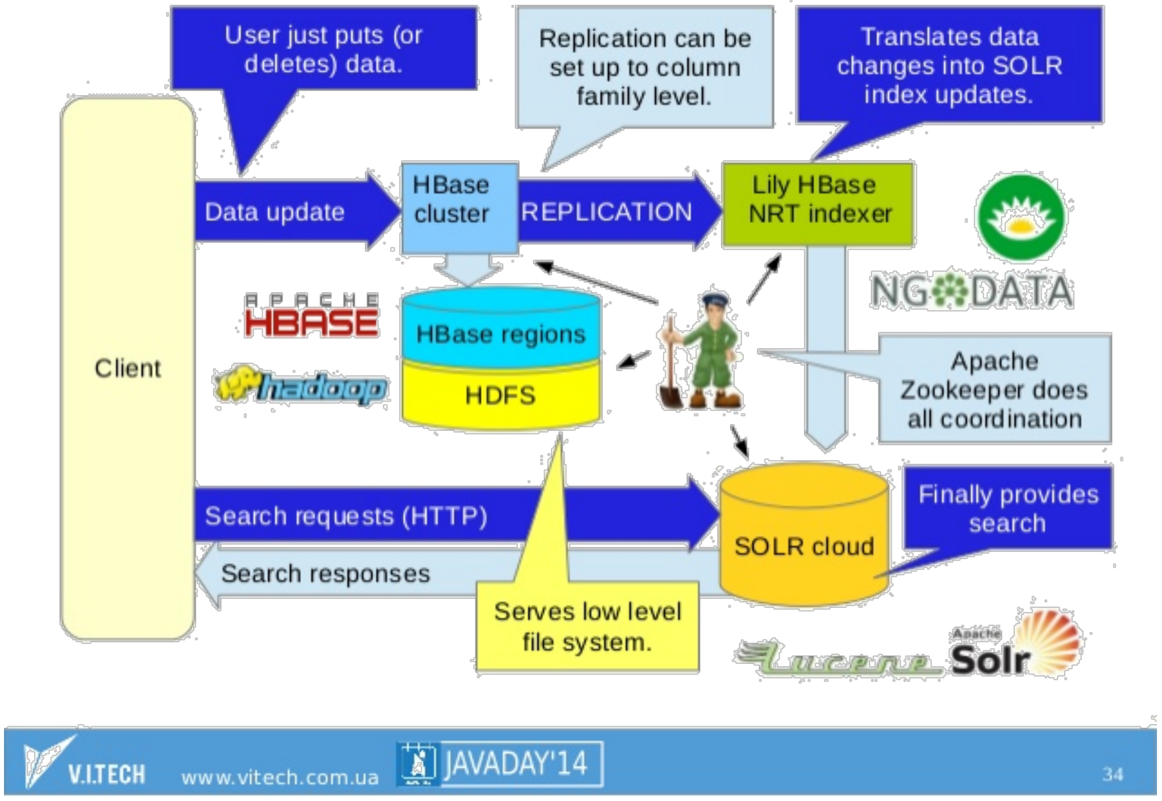


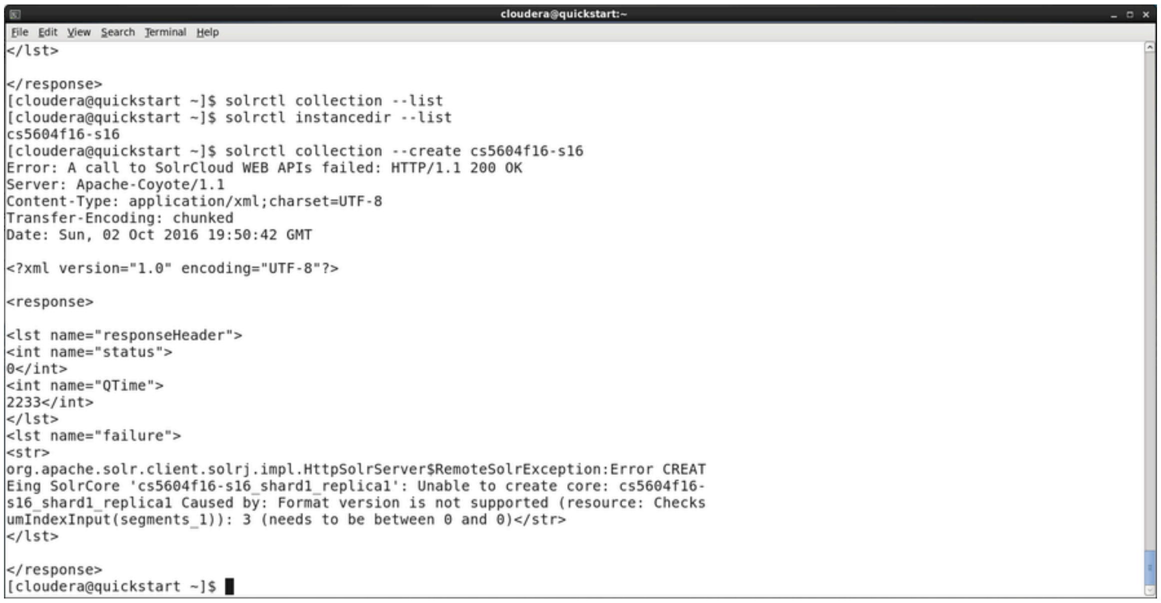
Figure 12 Lily Hbase Indexer with Solr [21]

Figure 12 [21] shows the workflow of a Lily HBase indexer with Solr. The Lily HBase indexer will translate data changes into Solr index updates. The Solr cloud uses the results from the Lily HBase indexer for searching and querying.

8.2 Tutorials for Basic Indexing in Virtual Cloudera

In this section, we show how to successfully load data into our local Virtual Cloudera and realize the indexing with Solr. By following these steps, we can easily deploy our framework in Virtual Cloudera.

To make building and testing easier, we strongly recommend the developers install the same version of Solr in both Virtual Cloudera and Hadoop Cluster. If you get a format error while doing the same operation on the above machines, please check the version of Solr. Figure 13 shows the version conflict in Solr.



```
</lst>

</response>
[cloudera@quickstart ~]$ solrctl collection --list
[cloudera@quickstart ~]$ solrctl instancedir --list
cs5604f16-s16
[cloudera@quickstart ~]$ solrctl collection --create cs5604f16-s16
Error: A call to SolrCloud WEB APIs failed: HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 02 Oct 2016 19:50:42 GMT

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

<response>

<lst name="responseHeader">
<int name="status">
0</int>
<int name="QTime">
2233</int>
</lst>
<lst name="failure">
<str>
org.apache.solr.client.solrj.impl.HttpSolrServer$RemoteSolrException:Error CREAT
Eing SolrCore 'cs5604f16-s16_shard1_replica1': Unable to create core: cs5604f16-
s16_shard1_replica1 Caused by: Format version is not supported (resource: Checks
umIndexInput(segments_1)): 3 (needs to be between 0 and 0)</str>
</lst>

</response>
[cloudera@quickstart ~]$
```

Figure 13 Version Conflict in Solr

8.2.1 Preparation

Download Virtual Cloudera and set up the environment. We download the Virtual Cloudera 5.8 from the official website. The Solr version is compatible with the one on the Hadoop Cluster.

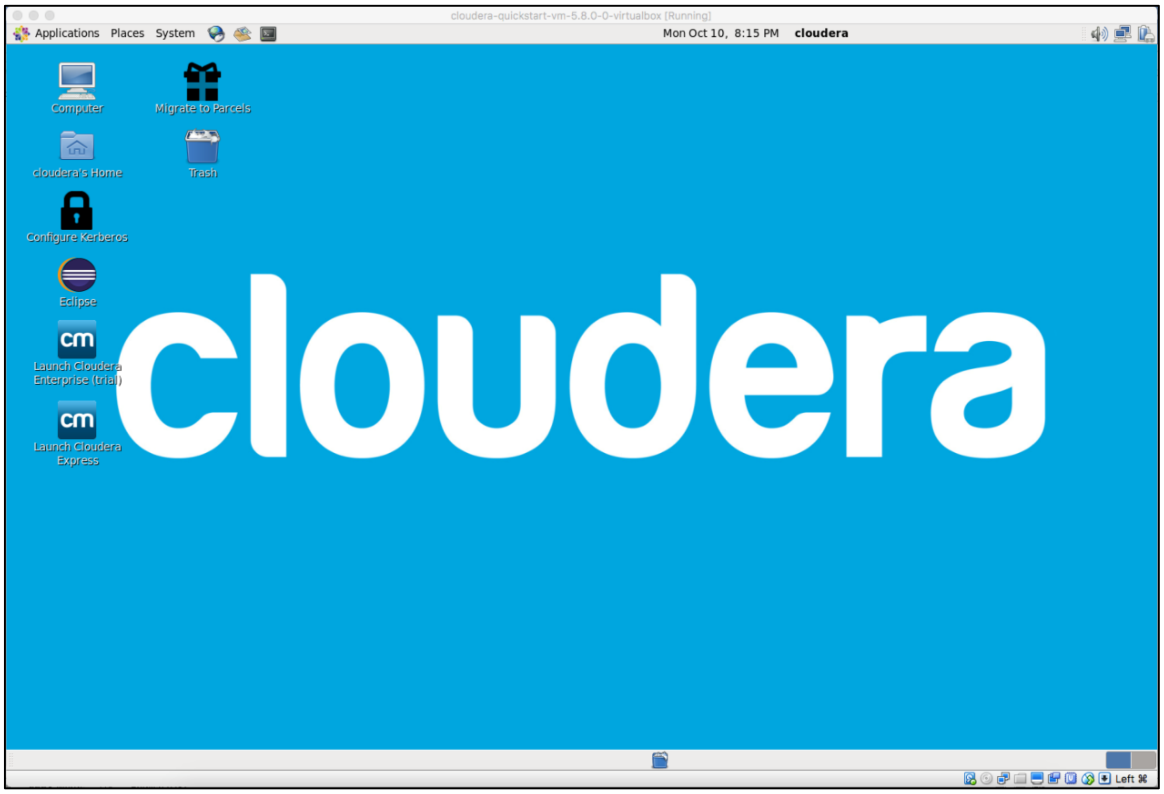
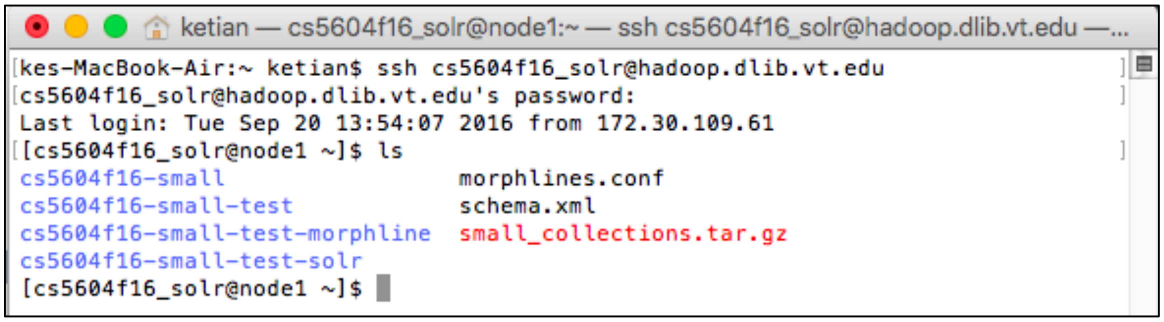


Figure 14 Interface of Virtual Cloudera

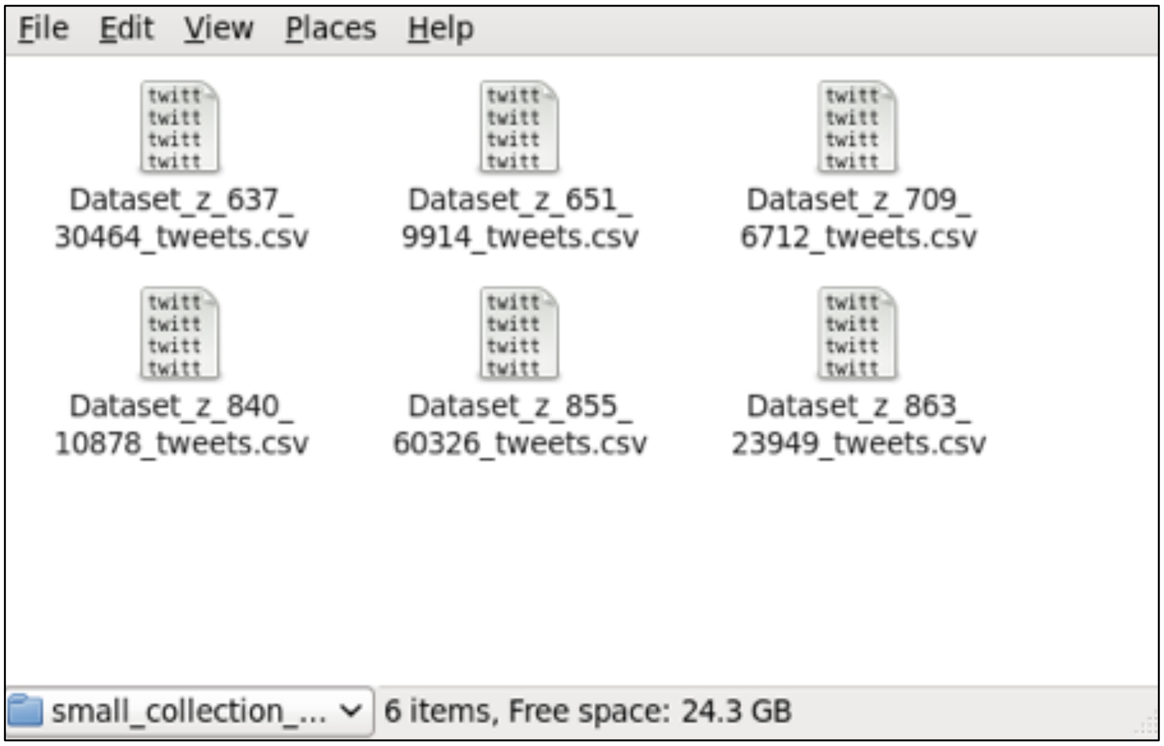
Collect the small Twitter collection for testing. We use SSH to access the cs5604f16 data collections in the Hadoop Cluster. Figure 15 shows how we connect to hadoop.dlib.vt.edu and identify the test file. Specifically, we use the small_collection.tar.gz in the virtual machine for testing.


ketian — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu —...

```
kes-MacBook-Air:~ ketian$ ssh cs5604f16_solr@hadoop.dlib.vt.edu
cs5604f16_solr@hadoop.dlib.vt.edu's password:
Last login: Tue Sep 20 13:54:07 2016 from 172.30.109.61
[cs5604f16_solr@node1 ~]$ ls
cs5604f16-small                  morphlines.conf
cs5604f16-small-test             schema.xml
cs5604f16-small-test-morphline   small_collections.tar.gz
cs5604f16-small-test-solr
[cs5604f16_solr@node1 ~]$
```

Figure 15 User Directory in the Hadoop Cluster

Download and extract the data file in Virtual Cloudera. We use scp to download the small Twitter collection into our Virtual Cloudera and extract it into a list of CSV files. Figure 16 shows the extracted CSV files from the small collection. We will import them into HBase and use Solr to index them.



Dataset_z_637_30464_tweets.csv
Dataset_z_651_9914_tweets.csv
Dataset_z_709_6712_tweets.csv
Dataset_z_840_10878_tweets.csv
Dataset_z_855_60326_tweets.csv
Dataset_z_863_23949_tweets.csv

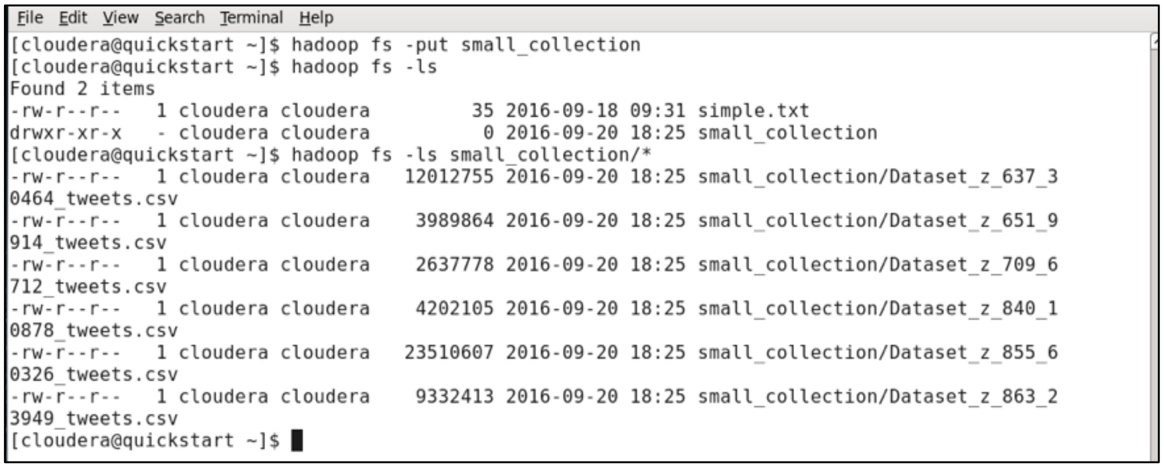
small_collection_...
6 items, Free space: 24.3 GB

Figure 16 Small Collection in the Hadoop Cluster

8.2.2 Import Data into HDFS and HBase

Upload the list of CSV files into HDFS with Hadoop fs -put commands. Now, we are able to find these files in our local HDFS. Figure 17 shows commands for putting local files into HDFS. Obviously, we can also use similar commands to import webpage data.

Key Command(s)
hadoop fs -put small_collection hadoop fs -put dataset_test.csv

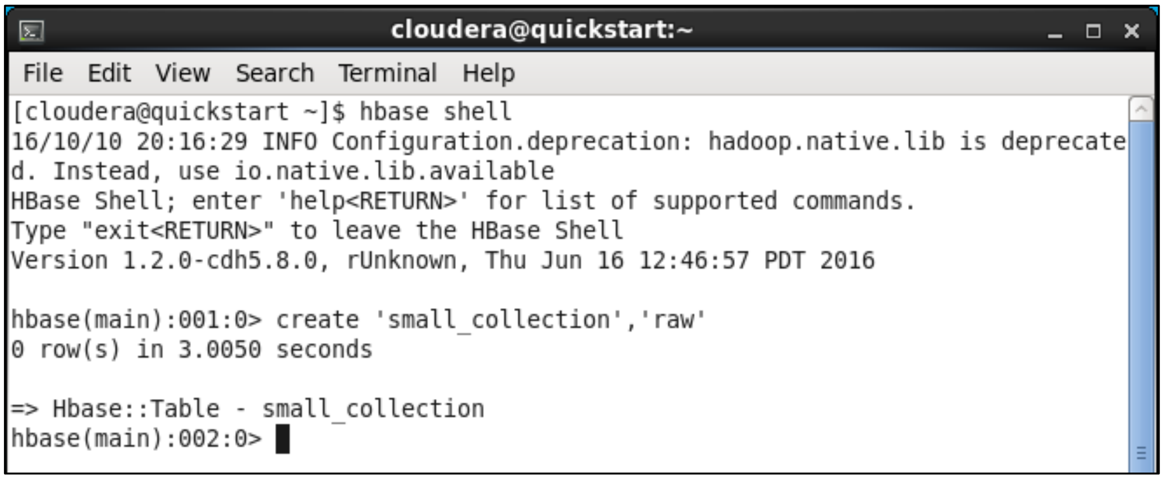


```
[cloudera@quickstart ~]$ hadoop fs -put small_collection
[cloudera@quickstart ~]$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 cloudera cloudera         35 2016-09-18 09:31 simple.txt
drwxr-xr-x   - cloudera cloudera          0 2016-09-20 18:25 small_collection
[cloudera@quickstart ~]$ hadoop fs -ls small_collection/*
-rw-r--r--   1 cloudera cloudera   12012755 2016-09-20 18:25 small_collection/Dataset_z_637_3
0464_tweets.csv
-rw-r--r--   1 cloudera cloudera    3989864 2016-09-20 18:25 small_collection/Dataset_z_651_9
914_tweets.csv
-rw-r--r--   1 cloudera cloudera    2637778 2016-09-20 18:25 small_collection/Dataset_z_709_6
712_tweets.csv
-rw-r--r--   1 cloudera cloudera    4202105 2016-09-20 18:25 small_collection/Dataset_z_840_1
0878_tweets.csv
-rw-r--r--   1 cloudera cloudera   23510607 2016-09-20 18:25 small_collection/Dataset_z_855_6
0326_tweets.csv
-rw-r--r--   1 cloudera cloudera    9332413 2016-09-20 18:25 small_collection/Dataset_z_863_2
3949_tweets.csv
[cloudera@quickstart ~]$ 
```

Figure 17 Import Local Files into HDFS in Virtual Cloudera

Create HBase table for the test file of tweets. We use HBase shell commands to create a table. For example, we use the “create” command to create a new table. We use the “scan/describe” command to list existing tables in HBase. Figure 18 shows the commands to create a HBase table.

Key Command(s)
create ‘small_collection’, ‘raw’ disable ‘small_collection’ drop ‘small_collection’

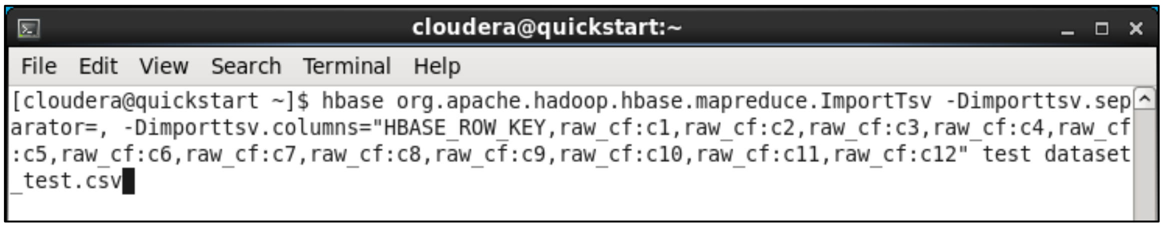


```
[cloudera@quickstart ~]$ hbase shell
16/10/10 20:16:29 INFO Configuration.deprecation: hadoop.native.lib is deprecate
d. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.8.0, rUnknown, Thu Jun 16 12:46:57 PDT 2016

hbase(main):001:0> create 'small_collection','raw'
0 row(s) in 3.0050 seconds

=> Hbase::Table - small_collection
hbase(main):002:0> 
```

Figure 18 Create a HBase Table in Virtual Cloudera

Import CSV files into HBase tables. We run importtsv MapReduce to import the test file into HBase tables. Figure 19 shows the command to import Twitter data files.

Key Command(s)
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY,raw_cf:c1,raw_cf:c2,raw_cf:c3,raw_cf:c4,raw_cf:c5,raw_cf:c6,raw_cf:c7,raw_cf:c8,raw_cf:c9,raw_cf:c10,raw_cf:c11,raw_cf:c12" test dataset_test.csv



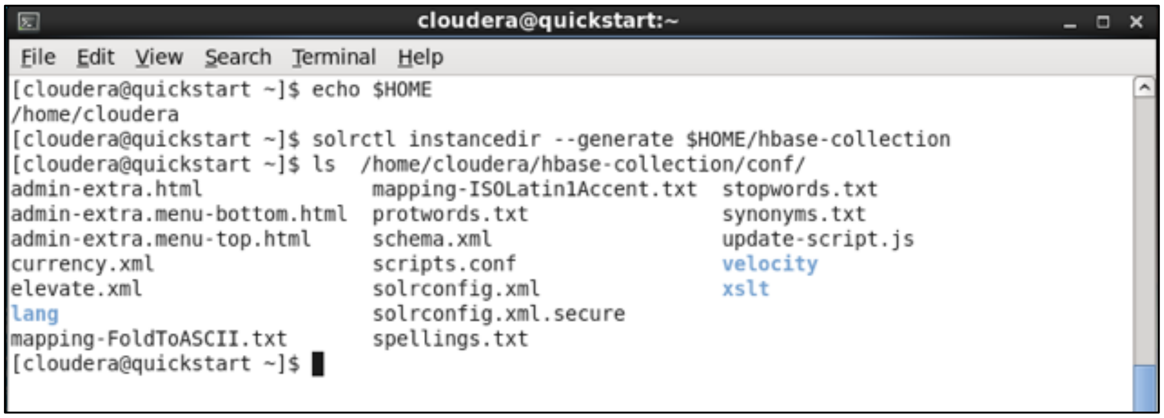
```
[cloudera@quickstart ~]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.sep
arator=, -Dimporttsv.columns="HBASE_ROW_KEY,raw_cf:c1,raw_cf:c2,raw_cf:c3,raw_cf:c4,raw_cf
:c5,raw_cf:c6,raw_cf:c7,raw_cf:c8,raw_cf:c9,raw_cf:c10,raw_cf:c11,raw_cf:c12" test dataset
_test.csv
```

Figure 19 Import Data into HBase in Virtual Cloudera

8.2.3 Create Solr Collection

Generate a Solr collection. Before using Solr for indexing, we need to generate a particular collection to hold the index. Configuration files including schema.xml, solrconfig.xml and other helper files for a collection are managed as part of the instance directory. Figure 20 shows how to initialize a new Solr directory. The schema.xml file is located in $HOME/hbase-collection/conf.

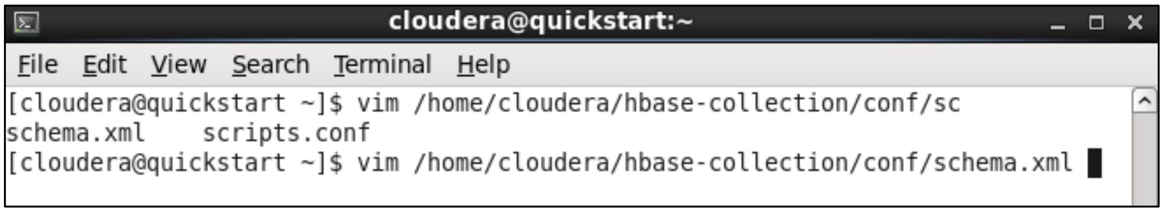
Key Command(s)
solrctl instancedir --generate $HOME/hbase_collection solrctl instancedir --create hbase_collection $HOME/hbase_collection solrctl collection --create hbase_collection



```
[cloudera@quickstart ~]$ echo $HOME
/home/cloudera
[cloudera@quickstart ~]$ solrctl instancedir --generate $HOME/hbase-collection
[cloudera@quickstart ~]$ ls  /home/cloudera/hbase-collection/conf/
admin-extra.html              mapping-ISOLatin1Accent.txt  stopwords.txt
admin-extra.menu-bottom.html  protwords.txt                synonyms.txt
admin-extra.menu-top.html     schema.xml                   update-script.js
currency.xml                  scripts.conf                 velocity
elevate.xml                   solrconfig.xml               xslt
lang                          solrconfig.xml.secure
mapping-FoldToASCII.txt       spellings.txt
[cloudera@quickstart ~]$
```

Figure 20 Create a Solr Directory in Virtual Cloudera

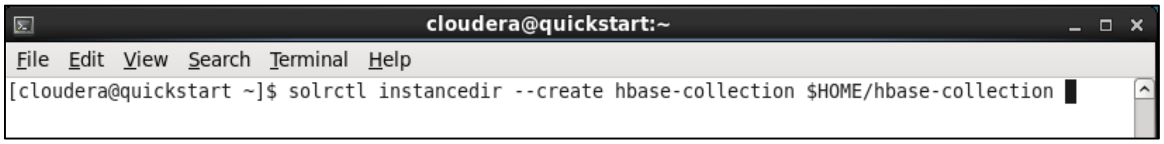
Edit Schema file with our customization. We need to edit the schema file for multiple requirements from other teams.



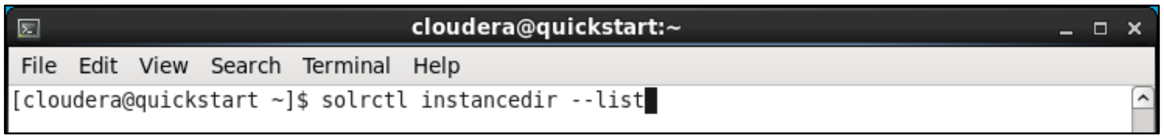
```
[cloudera@quickstart ~]$ vim /home/cloudera/hbase-collection/conf/sc
schema.xml    scripts.conf
[cloudera@quickstart ~]$ vim /home/cloudera/hbase-collection/conf/schema.xml
```

Figure 21 Customize the schema.xml in Virtual Cloudera

Upload a configuration file to ZooKeeper. We use the solrctl command to upload the configuration file into ZooKeeper. After uploading, the configuration file is available for Solr to use. Figure 22 shows the command we use for uploading the configuration file. We also use --list command to verify our uploading process.



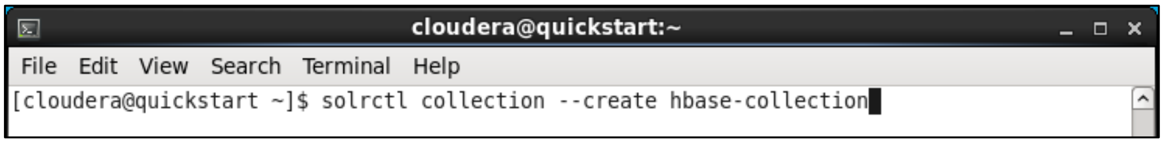
```
[cloudera@quickstart ~]$ solrctl instancedir --create hbase-collection $HOME/hbase-collection
```



```
[cloudera@quickstart ~]$ solrctl instancedir --list
```

Figure 22 Upload the Configuration File in Virtual Cloudera

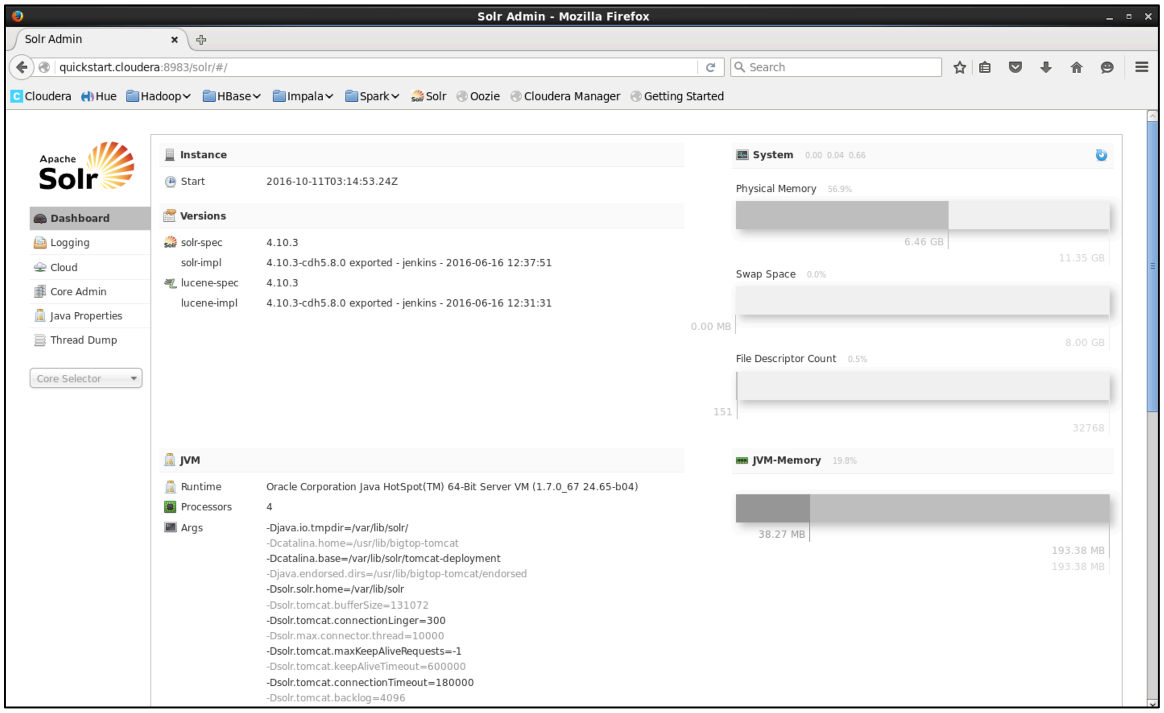
Create Solr collection. We then create our Solr collection using the command line as:



```
[cloudera@quickstart ~]$ solrctl collection --create hbase-collection
```

Figure 23 Create a Solr Collection in Virtual Cloudera

Then, we can find a new collection has been built in the Solr Admin UI.



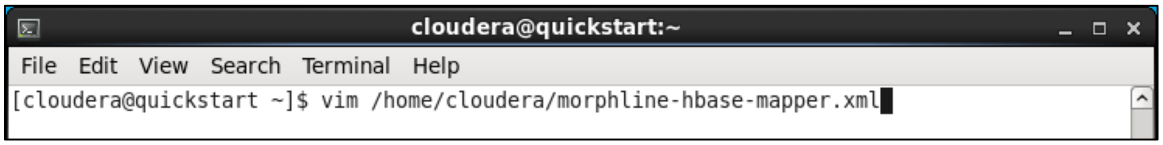
```
Instance
Start           2016-10-11T03:14:53.24Z

Versions
solr-spec       4.10.3
solr-impl       4.10.3-cdh5.8.0 exported - jenkins - 2016-06-16 12:37:51
lucene-spec     4.10.3
lucene-impl     4.10.3-cdh5.8.0 exported - jenkins - 2016-06-16 12:31:31

JVM
Runtime         Oracle Corporation Java HotSpot(TM) 64-Bit Server VM (1.7.0_67 24.65-b04)
Processors      4
Args            -Djava.io.tmpdir=/var/lib/solr/
                -Dcatalina.home=/usr/lib/bigtop-tomcat
                -Dcatalina.base=/var/lib/solr/tomcat-deployment
                -Djava.endorsed.dirs=/usr/lib/bigtop-tomcat/endorsed
                -Dsolr.solr.home=/var/lib/solr
                -Dsolr.tomcat.bufferSize=131072
                -Dsolr.tomcat.connectionLinger=300
                -Dsolr.max.connector.thread=10000
                -Dsolr.tomcat.maxKeepAliveRequests=-1
                -Dsolr.tomcat.keepAliveTimeout=600000
                -Dsolr.tomcat.connectionTimeout=180000
                -Dsolr.tomcat.backlog=4096
```

Figure 24 Verify the Solr Collection in Virtual Cloudera

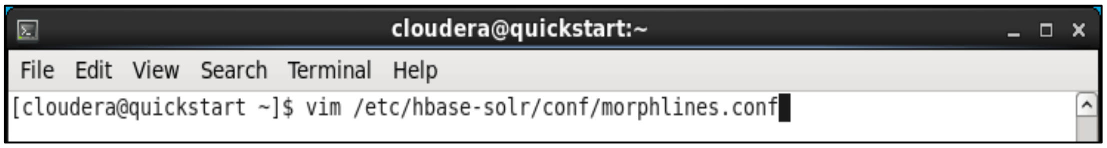
8.2.4 Use Lily Indexer for Data Indexing (Live Mode)

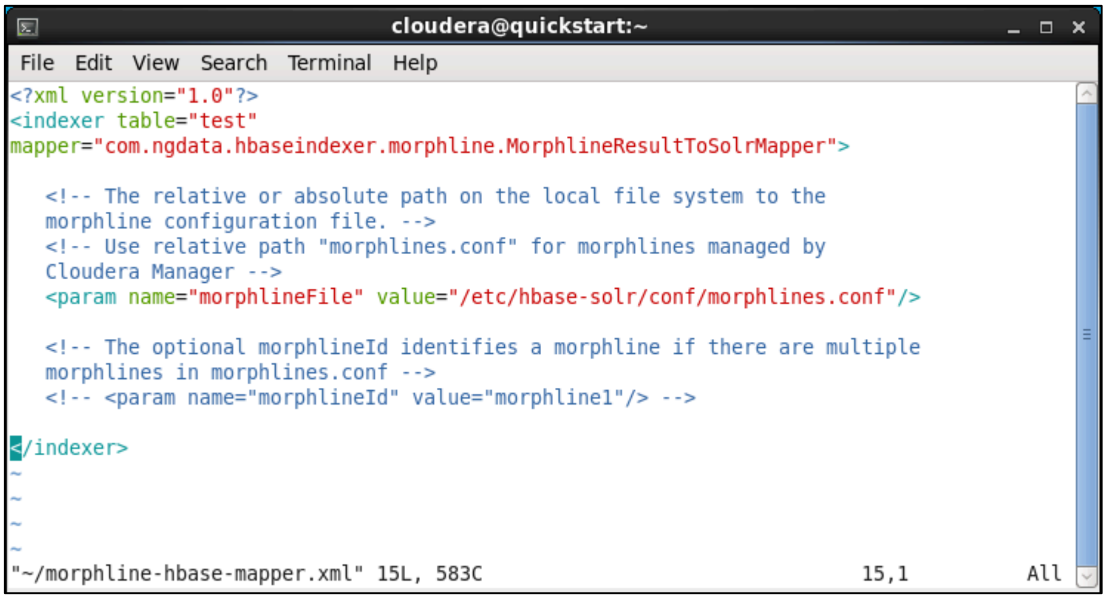
Create a Lily Indexer configuration file. We create a Lily indexer configuration file as morphline-hbase-mapper.xml. This XML file is referred to as the implementation of MorphlineResultSolrMapper. The morphline configuration file explains the morphline commands and the mapping between HBase columns and Solr indexes. Make sure that the table name and the morphline file path are right.

Note: the morphlines.conf must be created in the directory /etc/hbase-solr/conf in the virtual Cloudera 5.8.



```
[cloudera@quickstart ~]$ vim /home/cloudera/morphline-hbase-mapper.xml
```


```
[cloudera@quickstart ~]$ vim /etc/hbase-solr/conf/morphlines.conf
```


```
<?xml version="1.0"?>
<indexer table="test"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">

   <!-- The relative or absolute path on the local file system to the
   morphline configuration file. -->
   <!-- Use relative path "morphlines.conf" for morphlines managed by
   Cloudera Manager -->
   <param name="morphlineFile" value="/etc/hbase-solr/conf/morphlines.conf"/>

   <!-- The optional morphlineId identifies a morphline if there are multiple
   morphlines in morphlines.conf -->
   <!-- <param name="morphlineId" value="morphline1"/> -->

</indexer>
~
~
~
~
"~/morphline-hbase-mapper.xml" 15L, 583C                         15,1          All
```

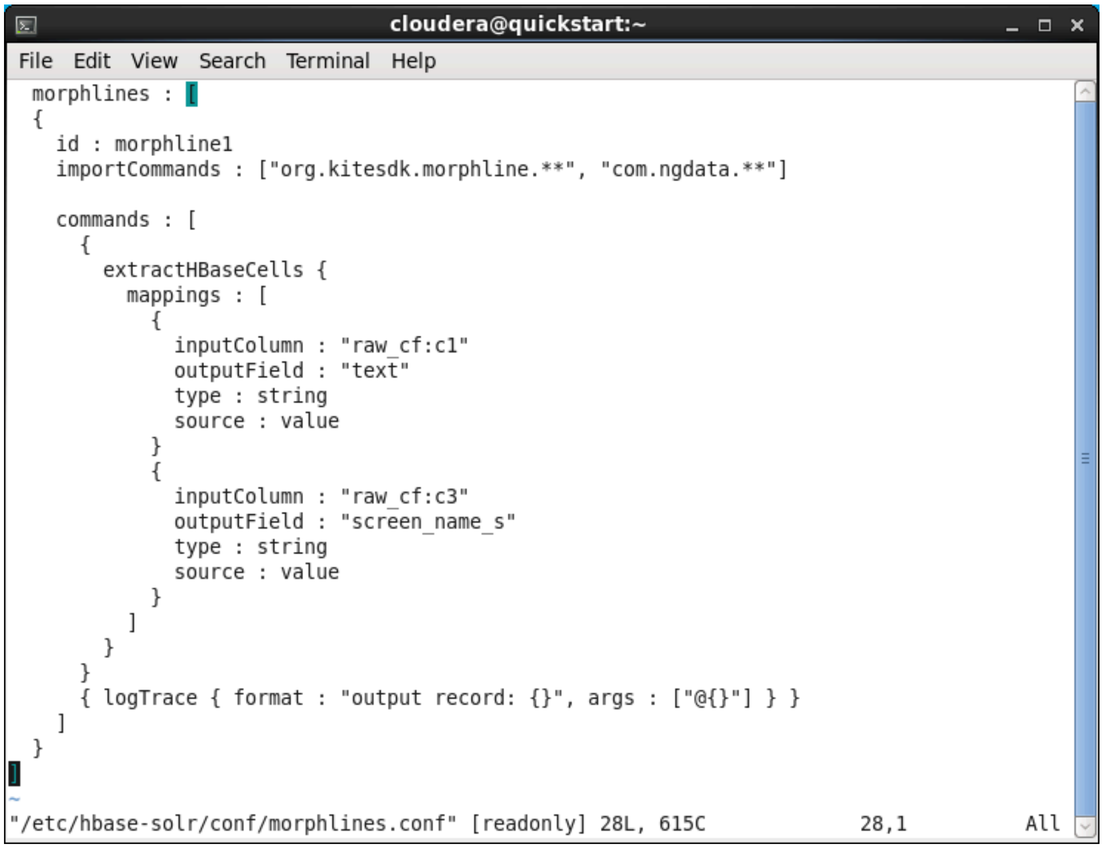
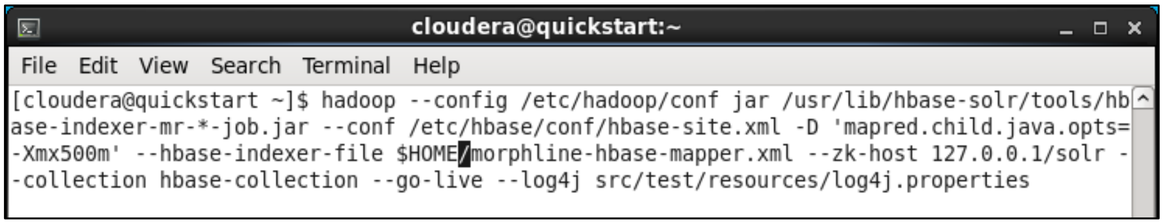

```
  morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.morphline.**", "com.ngdata.**"]

    commands : [
      {
        extractHBaseCells {
          mappings : [
            {
              inputColumn : "raw_cf:c1"
              outputField : "text"
              type : string
              source : value
            }
            {
              inputColumn : "raw_cf:c3"
              outputField : "screen_name_s"
              type : string
              source : value
            }
          ]
        }
      }
      { logTrace { format : "output record: {}", args : ["@{}"] } }
    ]
  }
]
~
"/etc/hbase-solr/conf/morphlines.conf" [readonly] 28L, 615C      28,1          All
```

Figure 25 Modify Key Morphline Files in Virtual Cloudera

Run HBaseMapReduceIndexer tool. We index the HBase table using MapReduce in live mode. Figure 26 shows the command for generating the index files with live mode.

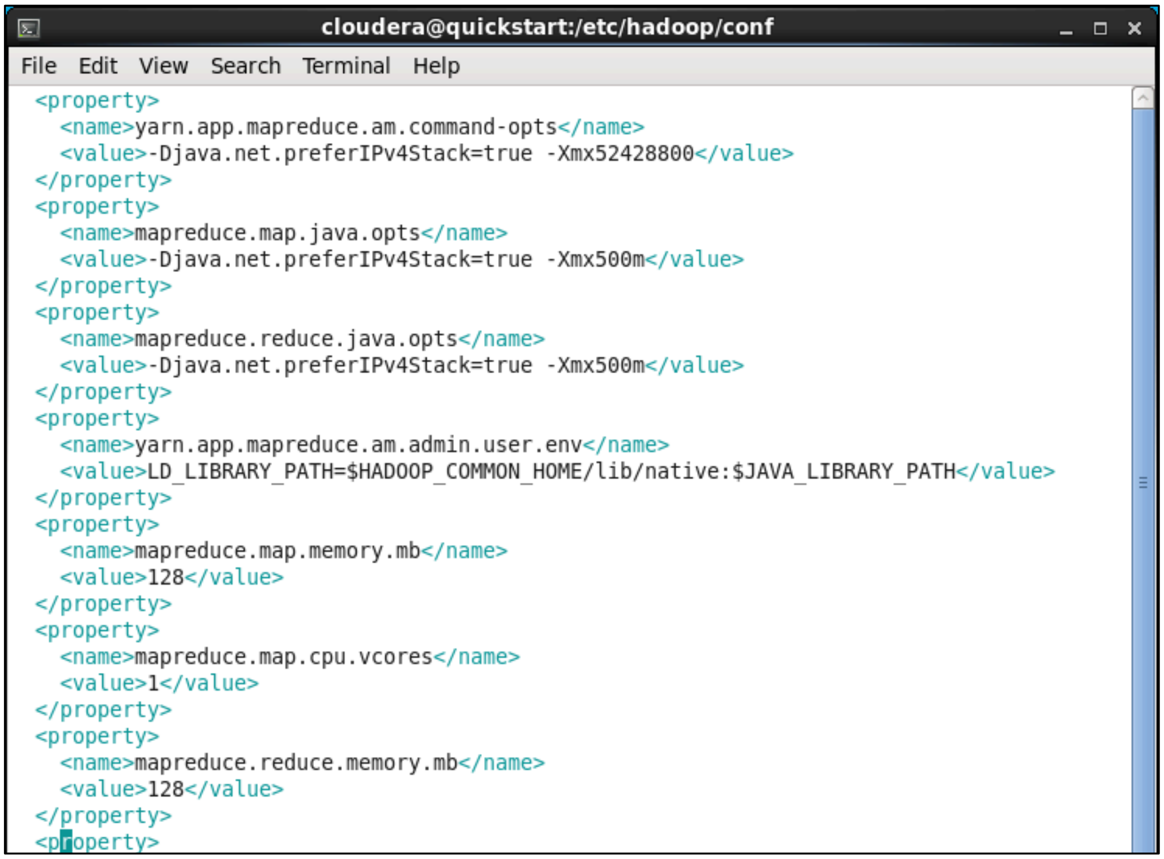
Key Command(s)
hadoop --config /etc/hadoop/conf jar /usr/lib/hbase-solr/tools/hbase-indexer-mr-*-job.jar --conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx500m' --hbase-indexer-file [LOCAL_DIR]/morphline-hbase-mapper.xml --zk-host 127.0.0.1/solr --collection [COLLECTION_NAME] --go-live --log4j [LOCAL_DIR]/log4j.properties



```
[cloudera@quickstart ~]$ hadoop --config /etc/hadoop/conf jar /usr/lib/hbase-solr/tools/hb
ase-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-collection --go-live --log4j src/test/resources/log4j.properties
```

Figure 26 Generate the Index Files with Live Mode in Virtual Cloudera

Note: If you get a Java heap space error, adjust the values of the mapreduce.map.java.opts and mapreduce.reduce.java.opts in the file /etc/hadoop/conf/mapred-site.xml.


cloudera@quickstart:/etc/hadoop/conf
File Edit View Search Terminal Help

```
  <property>
    <name>yarn.app.mapreduce.am.command-opts</name>
    <value>-Djava.net.preferIPv4Stack=true -Xmx52428800</value>
  </property>
  <property>
    <name>mapreduce.map.java.opts</name>
    <value>-Djava.net.preferIPv4Stack=true -Xmx500m</value>
  </property>
  <property>
    <name>mapreduce.reduce.java.opts</name>
    <value>-Djava.net.preferIPv4Stack=true -Xmx500m</value>
  </property>
  <property>
    <name>yarn.app.mapreduce.am.admin.user.env</name>
    <value>LD_LIBRARY_PATH=$HADOOP_COMMON_HOME/lib/native:$JAVA_LIBRARY_PATH</value>
  </property>
  <property>
    <name>mapreduce.map.memory.mb</name>
    <value>128</value>
  </property>
  <property>
    <name>mapreduce.map.cpu.vcores</name>
    <value>1</value>
  </property>
  <property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>128</value>
  </property>
  <property>
```

Figure 27 Fix the Java Heap Error in Virtual Cloudera

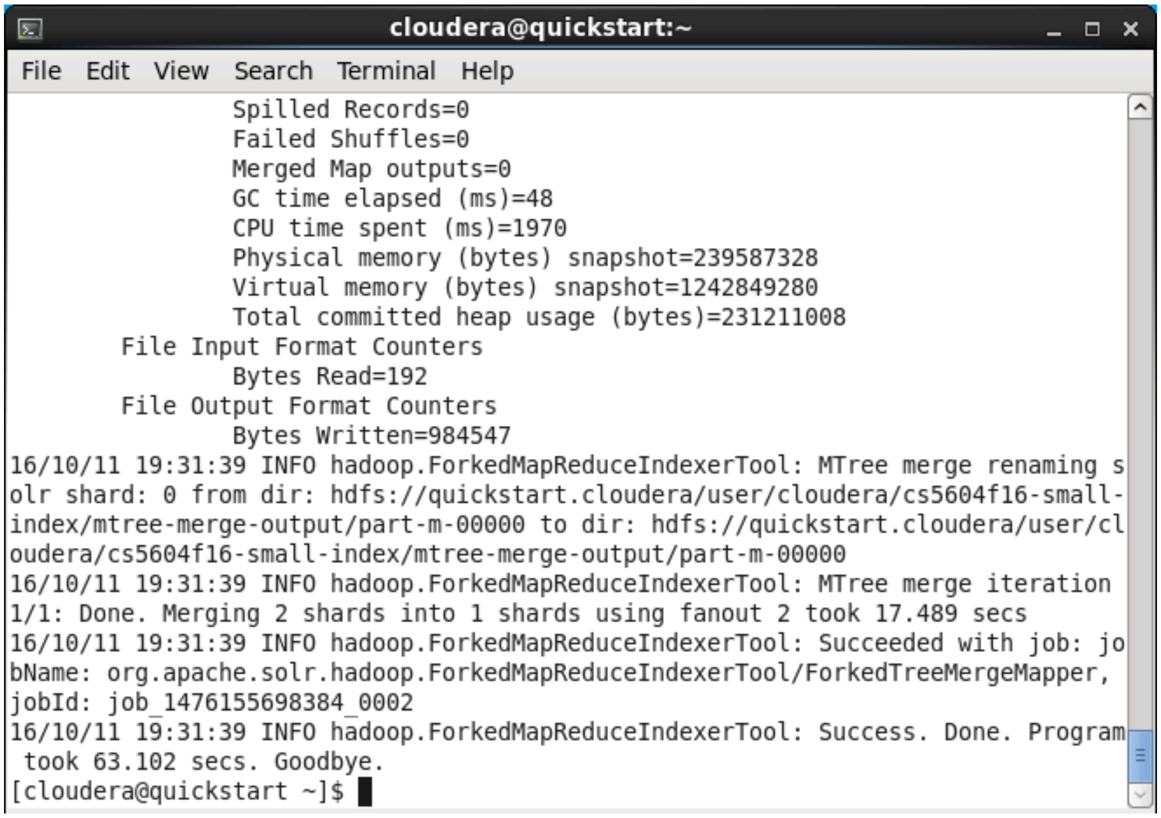
8.2.5 Use Lily Indexer for Data Indexing (Batch Mode)

The live mode of Lily Indexer is simple and efficient, because it makes full use of the memory. However, it cannot deal with big data. The batch mode of Lily indexer is helpful for indexing a large dataset.

Create a Lily Indexer configuration file. This step is the same as the one in live mode.

Run HBaseMapReduceIndexer tool. We index the HBase table using MapReduce in batch mode. The following command helps us to output the indexes into HDFS.

Key Command(s)
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 [LOCAL_DIR]/morphline-hbase-mapper.xml --zk-host 127.0.0.1/solr --log4j [LOCAL_DIR]/log4j.properties --collection [COLLECTION_NAME] --verbose --output-dir hdfs://quickstart.cloudera/user/cloudera/cs5604f16-small-index --overwrite-output-dir --shard 1

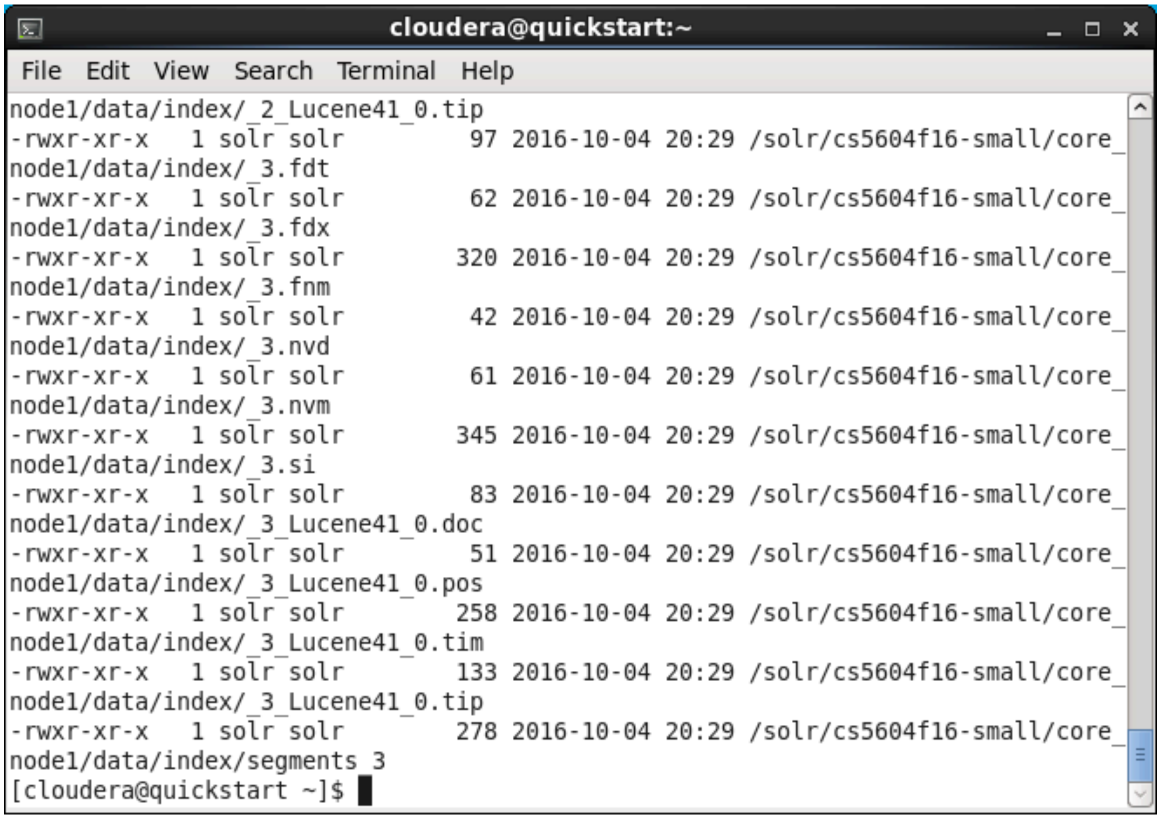


```
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=48
                CPU time spent (ms)=1970
                Physical memory (bytes) snapshot=239587328
                Virtual memory (bytes) snapshot=1242849280
                Total committed heap usage (bytes)=231211008
        File Input Format Counters
                Bytes Read=192
        File Output Format Counters
                Bytes Written=984547
16/10/11 19:31:39 INFO hadoop.ForkedMapReduceIndexerTool: MTree merge renaming s
olr shard: 0 from dir: hdfs://quickstart.cloudera/user/cloudera/cs5604f16-small-
index/mtree-merge-output/part-m-00000 to dir: hdfs://quickstart.cloudera/user/cl
oudera/cs5604f16-small-index/mtree-merge-output/part-m-00000
16/10/11 19:31:39 INFO hadoop.ForkedMapReduceIndexerTool: MTree merge iteration
1/1: Done. Merging 2 shards into 1 shards using fanout 2 took 17.489 secs
16/10/11 19:31:39 INFO hadoop.ForkedMapReduceIndexerTool: Succeeded with job: jo
bName: org.apache.solr.hadoop.ForkedMapReduceIndexerTool/ForkedTreeMergeMapper,
jobId: job_1476155698384_0002
16/10/11 19:31:39 INFO hadoop.ForkedMapReduceIndexerTool: Success. Done. Program
 took 63.102 secs. Goodbye.
[cloudera@quickstart ~]$ 
```

Figure 28 Generate the Index Files with Batch Mode in Virtual Cloudera

Check the files in HDFS. If the MapReduce job goes successfully, then multiple indexing files have already been generated in HDFS.

Key Command(s)
hadoop fs -ls /solr/cs5604f16-small/core_node1/data/index



```
node1/data/index/_2_Lucene41_0.tip
-rwxr-xr-x   1 solr solr         97 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3.fdt
-rwxr-xr-x   1 solr solr         62 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3.fdx
-rwxr-xr-x   1 solr solr        320 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3.fnm
-rwxr-xr-x   1 solr solr         42 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3.nvd
-rwxr-xr-x   1 solr solr         61 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3.nvm
-rwxr-xr-x   1 solr solr        345 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3.si
-rwxr-xr-x   1 solr solr         83 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3_Lucene41_0.doc
-rwxr-xr-x   1 solr solr         51 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3_Lucene41_0.pos
-rwxr-xr-x   1 solr solr        258 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3_Lucene41_0.tim
-rwxr-xr-x   1 solr solr        133 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/_3_Lucene41_0.tip
-rwxr-xr-x   1 solr solr        278 2016-10-04 20:29 /solr/cs5604f16-small/core_
node1/data/index/segments_3
[cloudera@quickstart ~]$ 
```

Figure 29 Check the Index Files in HDFS in Virtual Cloudera

Move the indexing files from HDFS to OS. Since we have already obtained those indexing files, we can easily move them from HDFS to OS, then it is possible to transfer those files to another machine.

Key Command(s)
hadoop fs -get /solr/cs5604f16-small/core_node1/data/index index-export

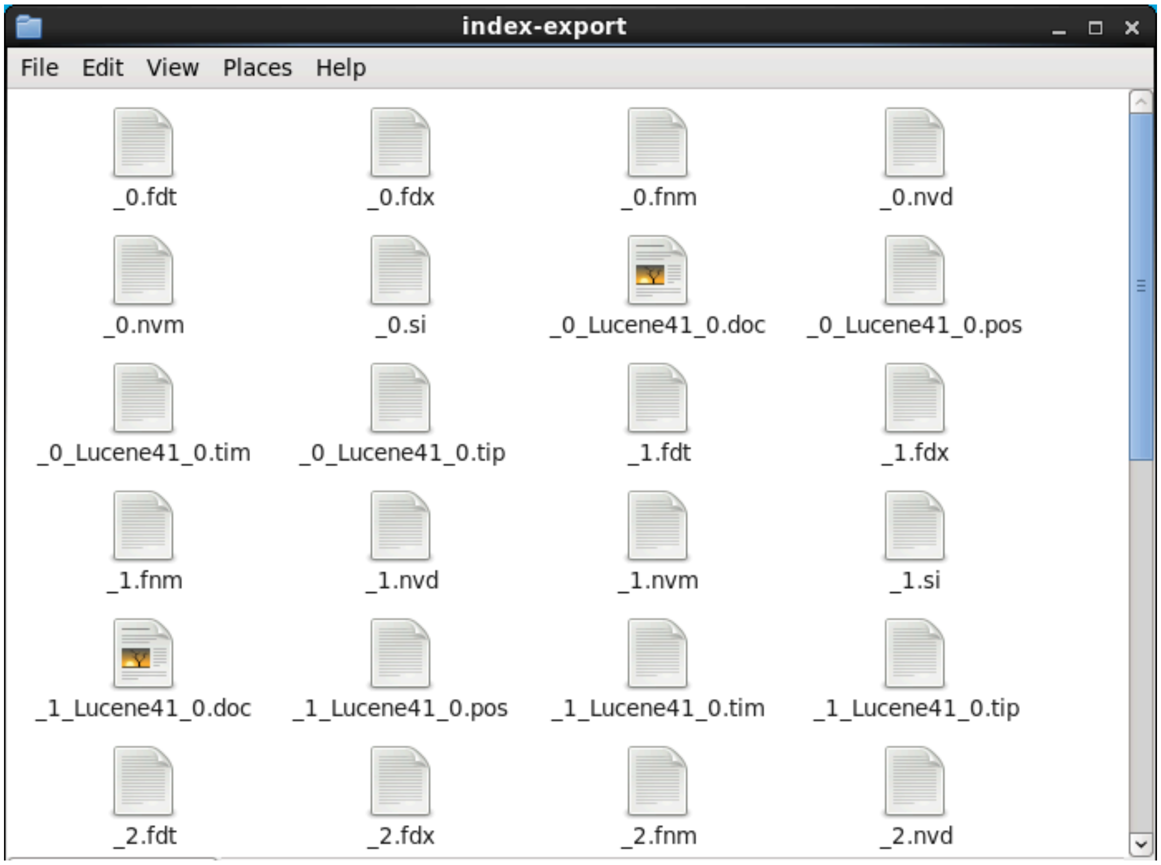
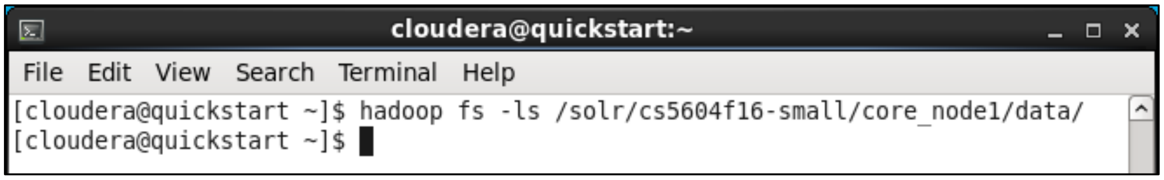


Figure 30 Move the Index Files from HDFS to OS in Virtual Cloudera

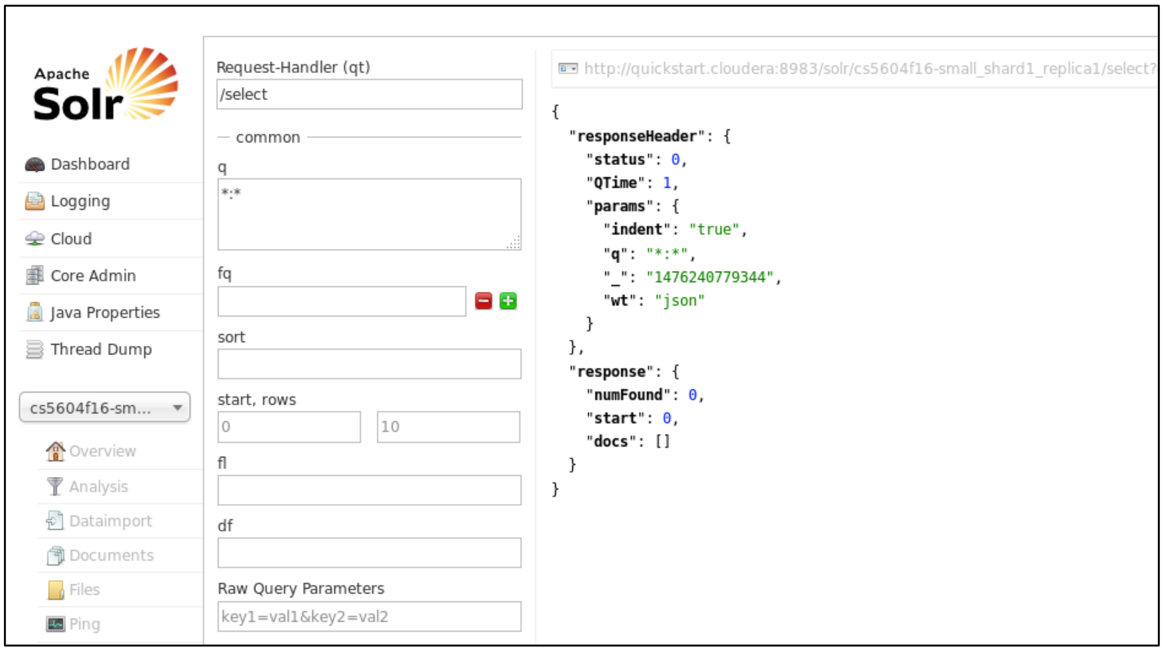
Clean a Solr collection. Now we try to use these offline indexing files to redo the indexing job. Before that, we need to remove all documents from the existing Solr collection.

Key Command(s)
sudo -u hdfs hadoop fs -rm -r -skipTrash /solr/[COLLECTION_NAME]/core_node1/data/index sudo -u hdfs hadoop fs -rm -r -skipTrash /solr/[COLLECTION_NAME]/core_node1/data/tlog



```
[cloudera@quickstart ~]$ hadoop fs -ls /solr/cs5604f16-small/core_node1/data/
[cloudera@quickstart ~]$ 
```

Figure 31 Clean a Solr Collection in Virtual Cloudera

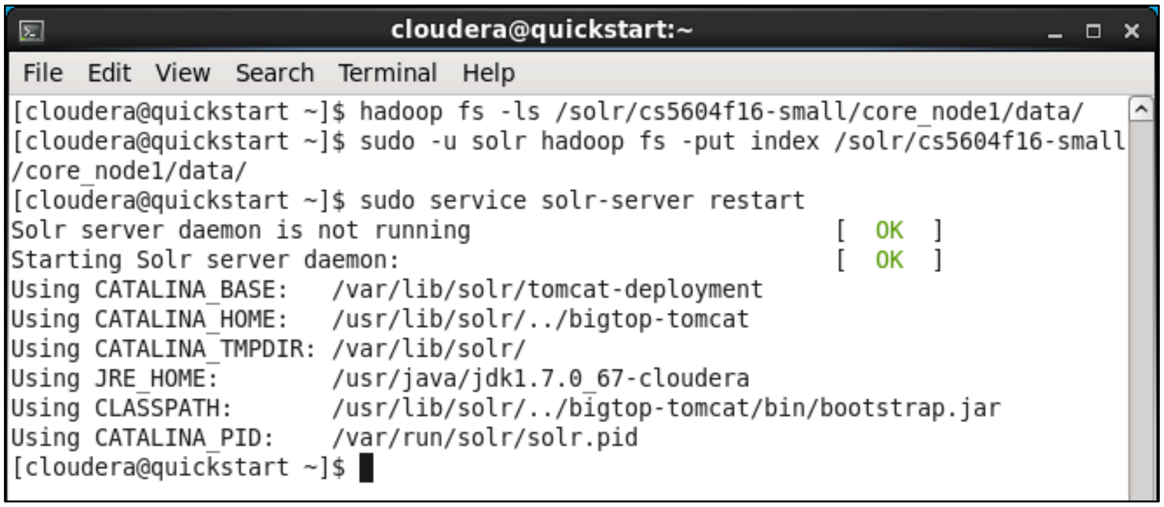


```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "*:*",
      "_": "1476240779344",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 0,
    "start": 0,
    "docs": []
  }
}
```

Figure 32 Verify the Clean Solr Collection in Virtual Cloudera

Put the offline indexing files into the Solr collection and restart the Solr service. Please notice that we use the role Solr to put into /solr.

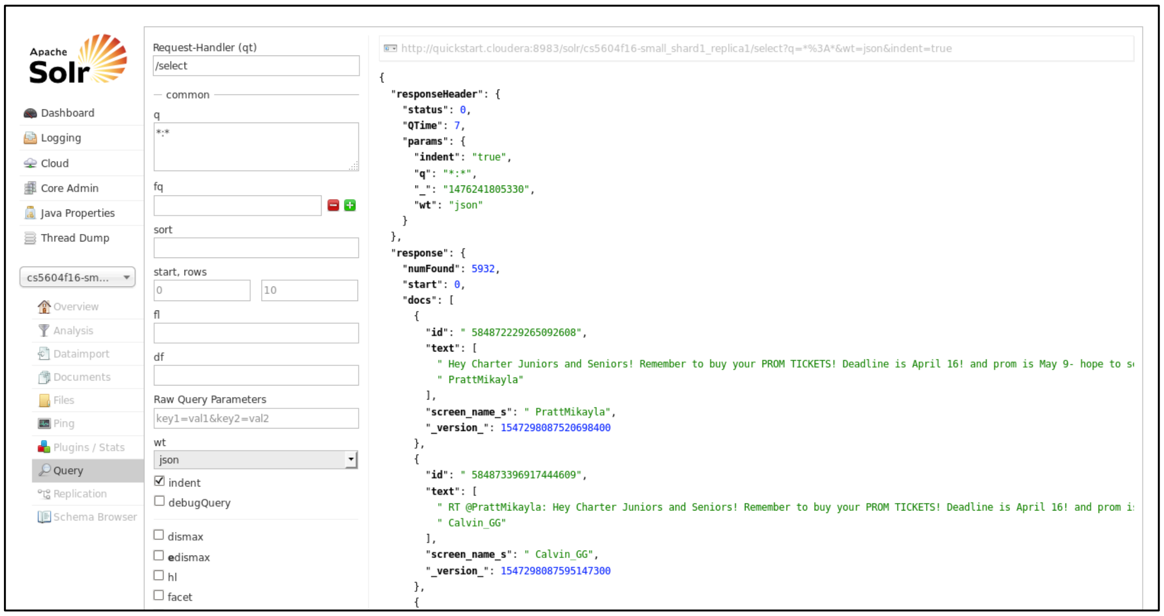
Key Command(s)
sudo -u solr hadoop fs -put index /solr/[COLLECTION_NAME]/core_node1/data/ sudo service solr-server restart



```
[cloudera@quickstart ~]$ hadoop fs -ls /solr/cs5604f16-small/core_node1/data/
[cloudera@quickstart ~]$ sudo -u solr hadoop fs -put index /solr/cs5604f16-small
/core_node1/data/
[cloudera@quickstart ~]$ sudo service solr-server restart
Solr server daemon is not running                          [  OK  ]
Starting Solr server daemon:                               [  OK  ]
Using CATALINA_BASE:   /var/lib/solr/tomcat-deployment
Using CATALINA_HOME:   /usr/lib/solr/../bigtop-tomcat
Using CATALINA_TMPDIR: /var/lib/solr/
Using JRE_HOME:        /usr/java/jdk1.7.0_67-cloudera
Using CLASSPATH:       /usr/lib/solr/../bigtop-tomcat/bin/bootstrap.jar
Using CATALINA_PID:    /var/run/solr/solr.pid
[cloudera@quickstart ~]$ 
```

Figure 33 Move the Index Files from OS to Solr in Virtual Cloudera

Note: if you get a Java heap size error, please set solr.hdfs.blockcache.enabled as false in the solrconfig.xml and then restart the Solr service again.



Request-Handler (qt): /select
q: *:*
fq:
sort:
start, rows: 0 10
fl:
df:
Raw Query Parameters: key1=val1&key2=val2
wt: json
☑ indent
☐ debugQuery
☐ dismax
☐ edismax
☐ hl
☐ facet

```
http://quickstart.cloudera:8983/solr/cs5604f16-small_shard1_replica1/select?q=*%3A*&wt=json&indent=true
{
  "responseHeader": {
    "status": 0,
    "QTime": 7,
    "params": {
      "indent": "true",
      "q": "*:*",
      "_": "1476241805330",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 5932,
    "start": 0,
    "docs": [
      {
        "id": " 584872229265092608",
        "text": [
          " Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is April 16! and prom is May 9- hope to s
          " PrattMikayla"
        ],
        "screen_name_s": " PrattMikayla",
        "_version_": 1547298087520698400
      },
      {
        "id": " 584873396917444609",
        "text": [
          " RT @PrattMikayla: Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is April 16! and prom i
          " Calvin_GG"
        ],
        "screen_name_s": " Calvin_GG",
        "_version_": 1547298087595147300
      },
      {
```

Figure 34 Basic Indexing in Virtual Cloudera (Batch Mode)

8.3 Tutorial for Basic Indexing in Hadoop Cluster

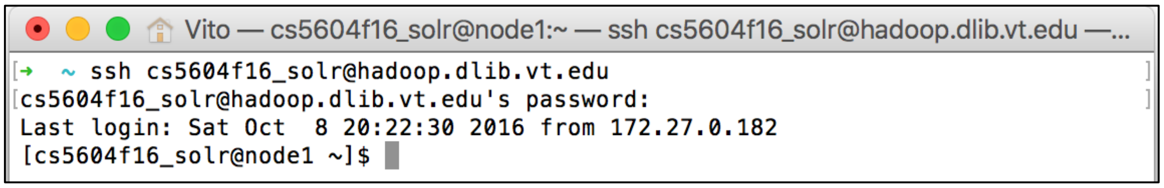
The main steps of basic indexing in the Hadoop Cluster are similar to those in Virtual Cloudera, but you should notice that the address of the Solr server and some library and configuration files are different from the above.

8.3.1 Import Data into HDFS and HBase

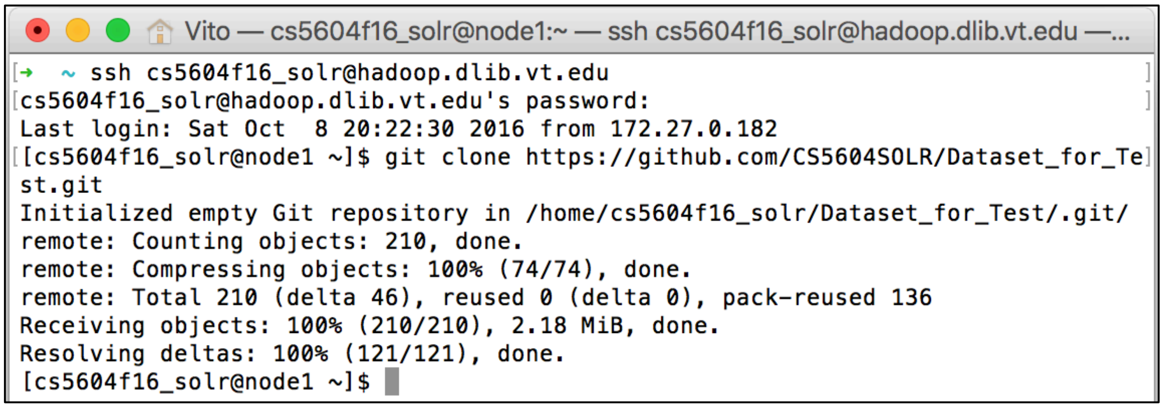
In this part, we download our test file from GitHub and then put it in the HDFS in the Hadoop Cluster.

Upload the list of CSV files into HDFS with Hadoop fs -put commands. We use SSH to access the Hadoop Cluster and then clone the test file on GitHub to the server. After that, assuming the files are stored in the Dataset_for_Test directory, you can use the cd command to go to that directory and put the test file into HDFS.

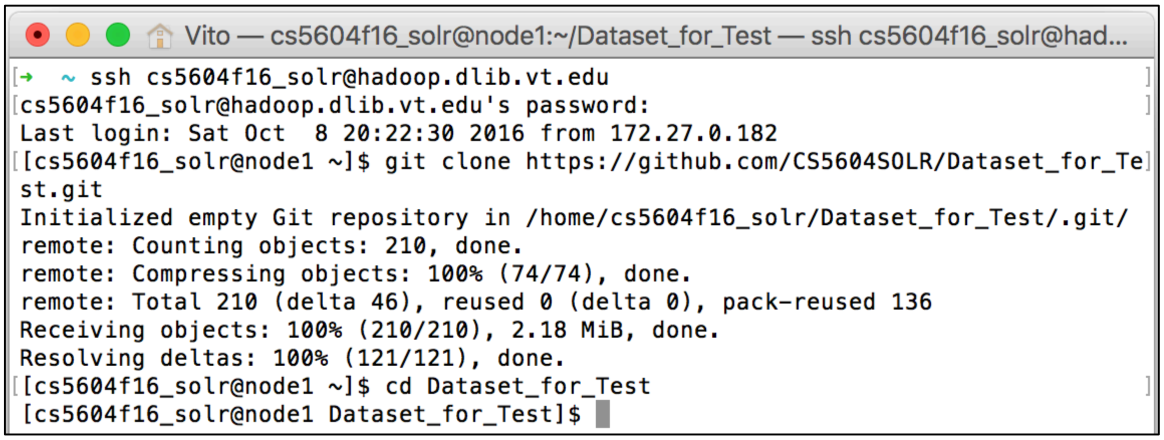
Key Command(s)
hadoop fs –put dataset_test.csv


Vito — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu —...

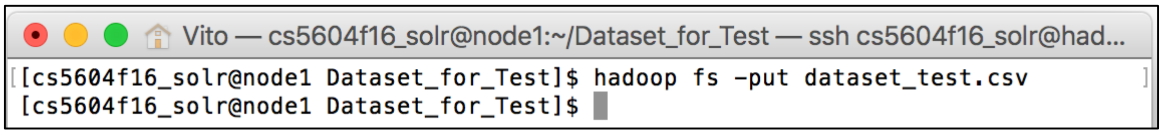
```
→  ~ ssh cs5604f16_solr@hadoop.dlib.vt.edu
cs5604f16_solr@hadoop.dlib.vt.edu's password:
Last login: Sat Oct  8 20:22:30 2016 from 172.27.0.182
[cs5604f16_solr@node1 ~]$
```


Vito — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu —...

```
→  ~ ssh cs5604f16_solr@hadoop.dlib.vt.edu
cs5604f16_solr@hadoop.dlib.vt.edu's password:
Last login: Sat Oct  8 20:22:30 2016 from 172.27.0.182
[cs5604f16_solr@node1 ~]$ git clone https://github.com/CS5604SOLR/Dataset_for_Te
st.git
Initialized empty Git repository in /home/cs5604f16_solr/Dataset_for_Test/.git/
remote: Counting objects: 210, done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 210 (delta 46), reused 0 (delta 0), pack-reused 136
Receiving objects: 100% (210/210), 2.18 MiB, done.
Resolving deltas: 100% (121/121), done.
[cs5604f16_solr@node1 ~]$
```


Vito — cs5604f16_solr@node1:~/Dataset_for_Test — ssh cs5604f16_solr@had...

```
→  ~ ssh cs5604f16_solr@hadoop.dlib.vt.edu
cs5604f16_solr@hadoop.dlib.vt.edu's password:
Last login: Sat Oct  8 20:22:30 2016 from 172.27.0.182
[cs5604f16_solr@node1 ~]$ git clone https://github.com/CS5604SOLR/Dataset_for_Te
st.git
Initialized empty Git repository in /home/cs5604f16_solr/Dataset_for_Test/.git/
remote: Counting objects: 210, done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 210 (delta 46), reused 0 (delta 0), pack-reused 136
Receiving objects: 100% (210/210), 2.18 MiB, done.
Resolving deltas: 100% (121/121), done.
[cs5604f16_solr@node1 ~]$ cd Dataset_for_Test
[cs5604f16_solr@node1 Dataset_for_Test]$
```

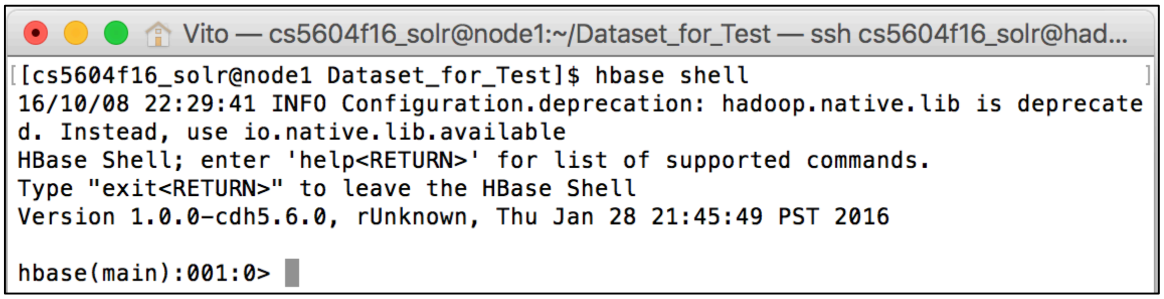

Vito — cs5604f16_solr@node1:~/Dataset_for_Test — ssh cs5604f16_solr@had...

```
[cs5604f16_solr@node1 Dataset_for_Test]$ hadoop fs -put dataset_test.csv
[cs5604f16_solr@node1 Dataset_for_Test]$
```

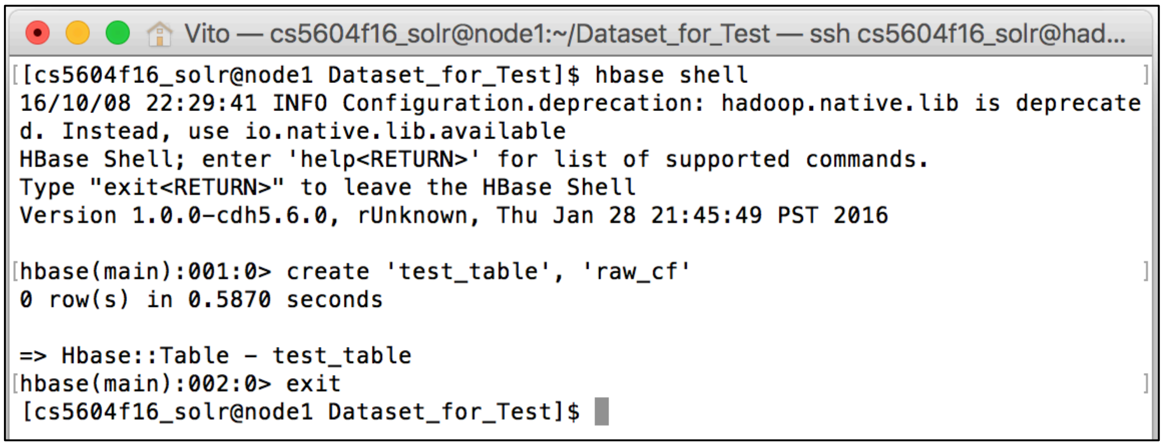
Figure 35 Import Local Files into HDFS in Hadoop Cluster

Create HBase table for the test file of tweets. For the next step, use the HBase shell to create a table named "test_table" in HBase.

Key Command(s)
create 'test_table', 'raw_cf' disable 'test_table' drop 'test_table'


Vito — cs5604f16_solr@node1:~/Dataset_for_Test — ssh cs5604f16_solr@had...

```
[cs5604f16_solr@node1 Dataset_for_Test]$ hbase shell
16/10/08 22:29:41 INFO Configuration.deprecation: hadoop.native.lib is deprecate
d. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.0.0-cdh5.6.0, rUnknown, Thu Jan 28 21:45:49 PST 2016

hbase(main):001:0>
```

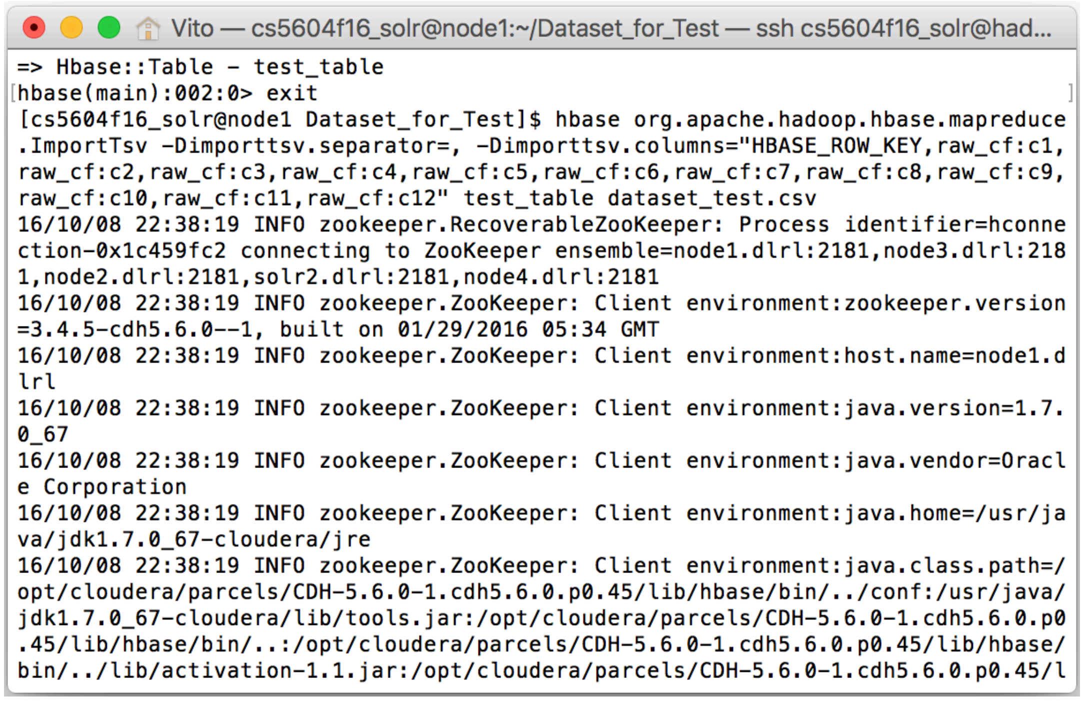

Vito — cs5604f16_solr@node1:~/Dataset_for_Test — ssh cs5604f16_solr@had...

```
[cs5604f16_solr@node1 Dataset_for_Test]$ hbase shell
16/10/08 22:29:41 INFO Configuration.deprecation: hadoop.native.lib is deprecate
d. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.0.0-cdh5.6.0, rUnknown, Thu Jan 28 21:45:49 PST 2016

hbase(main):001:0> create 'test_table', 'raw_cf'
0 row(s) in 0.5870 seconds

=> Hbase::Table - test_table
hbase(main):002:0> exit
[cs5604f16_solr@node1 Dataset_for_Test]$
```

Figure 36 Create a HBase Table in Hadoop Cluster

Import CSV files into HBase tables. We run importtsv MapReduce to import the test file into HBase tables. The following screenshot shows the command to import Twitter data files.

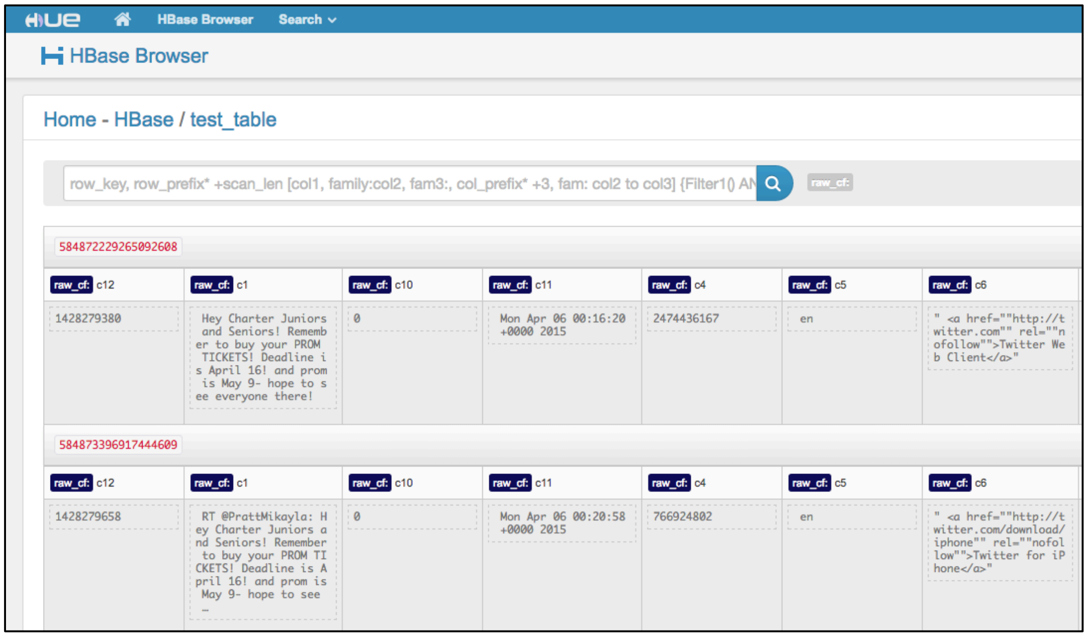
Key Command(s)
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY,raw_cf:c1,raw_cf:c2,raw_cf:c3,raw_cf:c4,raw_cf:c5,raw_cf:c6,raw_cf:c7,raw_cf:c8,raw_cf:c9,raw_cf:c10,raw_cf:c11,raw_cf:c12" test_table dataset_test.csv


Vito — cs5604f16_solr@node1:~/Dataset_for_Test — ssh cs5604f16_solr@had...

```
=> Hbase::Table - test_table
hbase(main):002:0> exit
[cs5604f16_solr@node1 Dataset_for_Test]$ hbase org.apache.hadoop.hbase.mapreduce
.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY,raw_cf:c1,
raw_cf:c2,raw_cf:c3,raw_cf:c4,raw_cf:c5,raw_cf:c6,raw_cf:c7,raw_cf:c8,raw_cf:c9,
raw_cf:c10,raw_cf:c11,raw_cf:c12" test_table dataset_test.csv
16/10/08 22:38:19 INFO zookeeper.RecoverableZooKeeper: Process identifier=hconne
ction-0x1c459fc2 connecting to ZooKeeper ensemble=node1.dlrl:2181,node3.dlrl:218
1,node2.dlrl:2181,solr2.dlrl:2181,node4.dlrl:2181
16/10/08 22:38:19 INFO zookeeper.ZooKeeper: Client environment:zookeeper.version
=3.4.5-cdh5.6.0--1, built on 01/29/2016 05:34 GMT
16/10/08 22:38:19 INFO zookeeper.ZooKeeper: Client environment:host.name=node1.d
lrl
16/10/08 22:38:19 INFO zookeeper.ZooKeeper: Client environment:java.version=1.7.
0_67
16/10/08 22:38:19 INFO zookeeper.ZooKeeper: Client environment:java.vendor=Oracl
e Corporation
16/10/08 22:38:19 INFO zookeeper.ZooKeeper: Client environment:java.home=/usr/ja
va/jdk1.7.0_67-cloudera/jre
16/10/08 22:38:19 INFO zookeeper.ZooKeeper: Client environment:java.class.path=/
opt/cloudera/parcels/CDH-5.6.0-1.cdh5.6.0.p0.45/lib/hbase/bin/../conf:/usr/java/
jdk1.7.0_67-cloudera/lib/tools.jar:/opt/cloudera/parcels/CDH-5.6.0-1.cdh5.6.0.p0
.45/lib/hbase/bin/..:/opt/cloudera/parcels/CDH-5.6.0-1.cdh5.6.0.p0.45/lib/hbase/
bin/../lib/activation-1.1.jar:/opt/cloudera/parcels/CDH-5.6.0-1.cdh5.6.0.p0.45/l
```

Figure 37 Import Data into HBase in Hadoop Cluster

If the whole process goes successfully, you are able to check the data in the Hue interface.


HUE
HBase Browser
Search
HBase Browser
Home - HBase / test_table
row_key, row_prefix* +scan_len [col1, family:col2, fam3:, col_prefix* +3, fam: col2 to col3] {Filter1() AN
raw_cf:

584872229265092608

raw_cf: c12	raw_cf: c1	raw_cf: c10	raw_cf: c11	raw_cf: c4	raw_cf: c5	raw_cf: c6
1428279380	Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is April 16! and prom is May 9- hope to see everyone there!	0	Mon Apr 06 00:16:20 +0000 2015	2474436167	en	" <a href=""http://twitter.com"" rel=""nofollow"">Twitter Web Client</a>"

584873396917444609

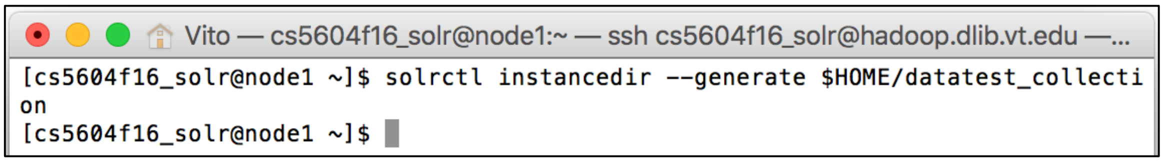
raw_cf: c12	raw_cf: c1	raw_cf: c10	raw_cf: c11	raw_cf: c4	raw_cf: c5	raw_cf: c6
1428279658	RT @PrattMikayla: Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is April 16! and prom is May 9- hope to see …	0	Mon Apr 06 00:20:58 +0000 2015	766924802	en	" <a href=""http://twitter.com/download/iphone"" rel=""nofollow"">Twitter for iPhone</a>"

Figure 38 Verify the Data in HBase in Hadoop Cluster

8.3.2 Create Solr Collection

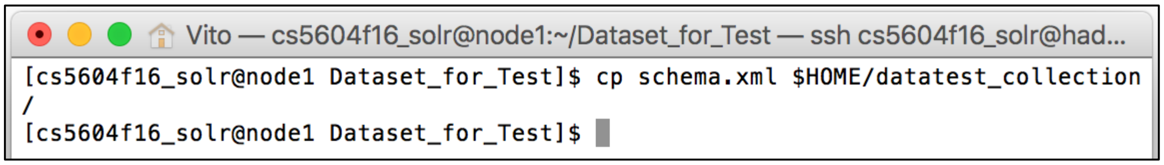
Here, we create a Solr collection named “datatest_collection”. The following screenshots show the steps for the process.

Key Command(s)
solrctl instancedir --generate $HOME/datatest_collection solrctl instancedir --create datatest_collection $HOME/datatest_collection solrctl collection --create datatest_collection



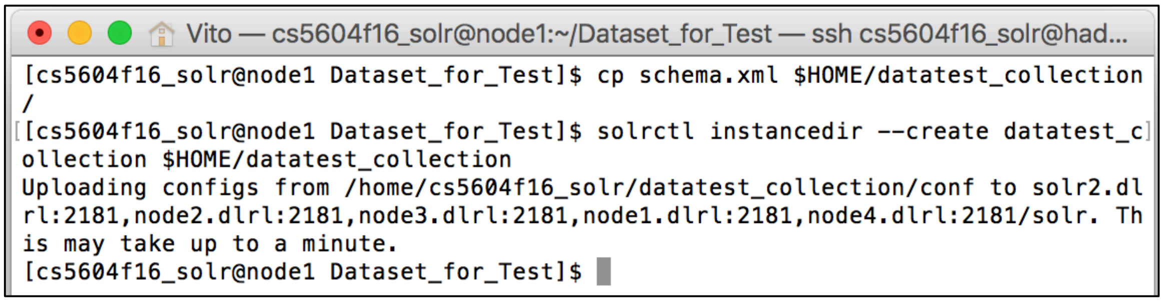
```
[cs5604f16_solr@node1 ~]$ solrctl instancedir --generate $HOME/datatest_collecti
on
[cs5604f16_solr@node1 ~]$
```

Figure 39 Create a Solr Directory in Hadoop Cluster



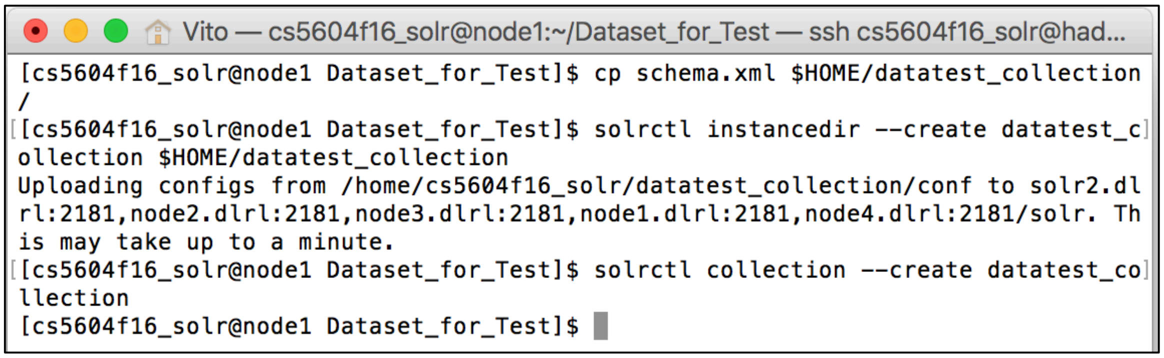
```
[cs5604f16_solr@node1 Dataset_for_Test]$ cp schema.xml $HOME/datatest_collection
/
[cs5604f16_solr@node1 Dataset_for_Test]$
```

Figure 40 Customize the schema.xml in Hadoop Cluster



```
[cs5604f16_solr@node1 Dataset_for_Test]$ cp schema.xml $HOME/datatest_collection
/
[cs5604f16_solr@node1 Dataset_for_Test]$ solrctl instancedir --create datatest_c
ollection $HOME/datatest_collection
Uploading configs from /home/cs5604f16_solr/datatest_collection/conf to solr2.dl
rl:2181,node2.dlrl:2181,node3.dlrl:2181,node1.dlrl:2181,node4.dlrl:2181/solr. Th
is may take up to a minute.
[cs5604f16_solr@node1 Dataset_for_Test]$
```

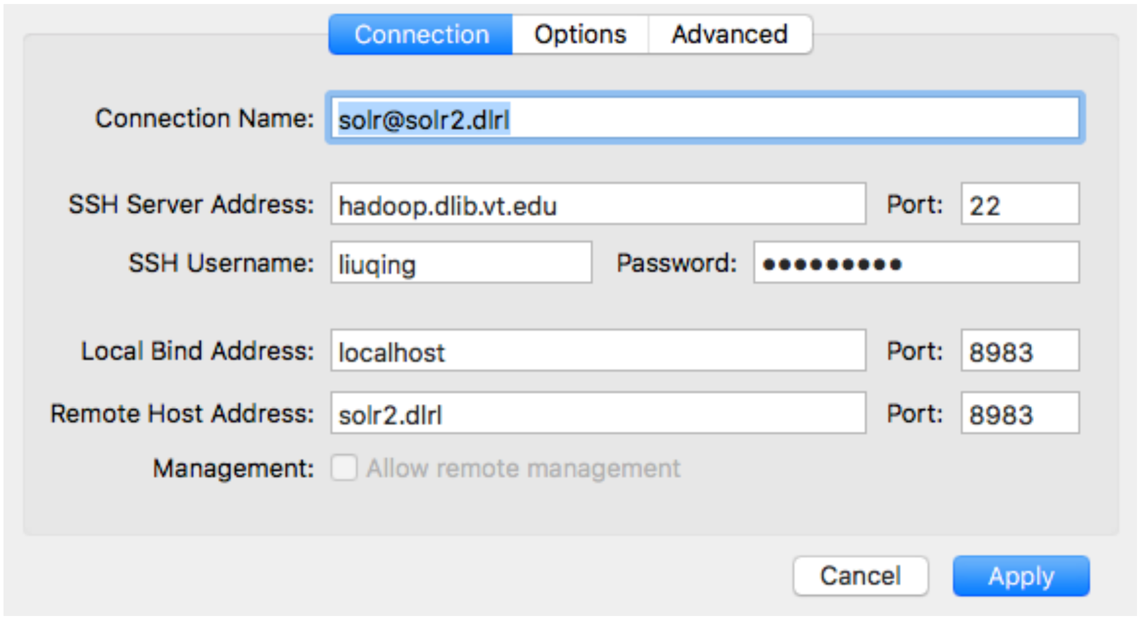
Figure 41 Upload the Configuration File in Hadoop Cluster



```
[cs5604f16_solr@node1 Dataset_for_Test]$ cp schema.xml $HOME/datatest_collection
/
[cs5604f16_solr@node1 Dataset_for_Test]$ solrctl instancedir --create datatest_c
ollection $HOME/datatest_collection
Uploading configs from /home/cs5604f16_solr/datatest_collection/conf to solr2.dl
rl:2181,node2.dlrl:2181,node3.dlrl:2181,node1.dlrl:2181,node4.dlrl:2181/solr. Th
is may take up to a minute.
[cs5604f16_solr@node1 Dataset_for_Test]$ solrctl collection --create datatest_co
llection
[cs5604f16_solr@node1 Dataset_for_Test]$
```

Figure 42 Create a Solr Collection in Hadoop Cluster

Now you can check the new collection in the Solr Admin UI. To start a remote connection to the Solr server in the Hadoop Cluster, there are several approaches. For Mac OS, install Secure Pipes and then modify the connection settings as in Figure 43.



Connection Name: solr@solr2.dlrl

SSH Server Address: hadoop.dlib.vt.edu Port: 22

SSH Username: liuqing Password: ●●●●●●●●●

Local Bind Address: localhost Port: 8983

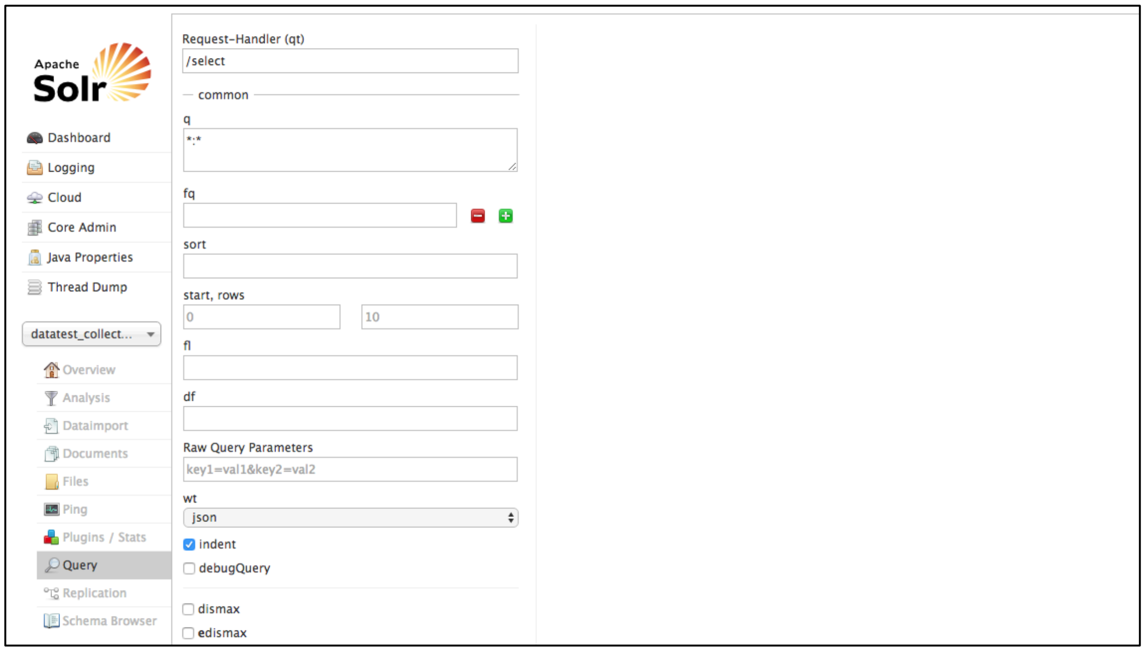
Remote Host Address: solr2.dlrl Port: 8983

Management: ☐ Allow remote management

Figure 43 Remote Connection to Solr in Hadoop Cluster

For other systems, use SSH to connect to the Solr server in the Hadoop Cluster.

Key Command(s)
ssh -L 9983:solr2.dlrl:8983 [username]@hadoop.dlib.vt.edu



Request-Handler (qt): /select

common

q: *:*

fq

sort

start, rows: 0 10

fl

df

Raw Query Parameters: key1=val1&key2=val2

wt: json

☑ indent

☐ debugQuery

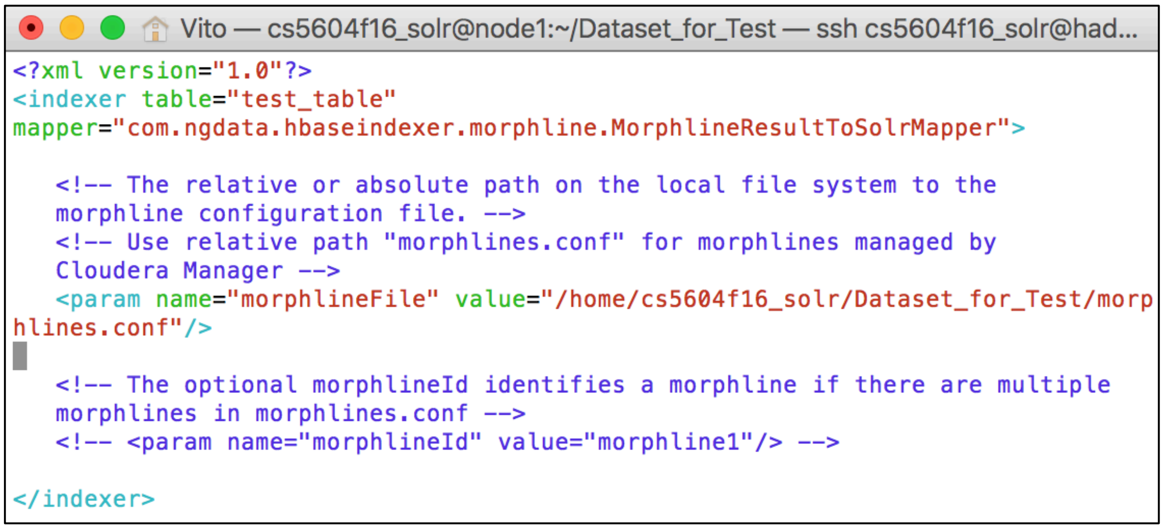
☐ dismax

☐ edismax

Figure 44 Verify the Solr Collection in Hadoop Cluster

8.3.3 Use Lily Indexer for Data Indexing (Live Mode)

Create a Lily Indexer configuration file. This step is the same as the one in Virtual Cloudera.

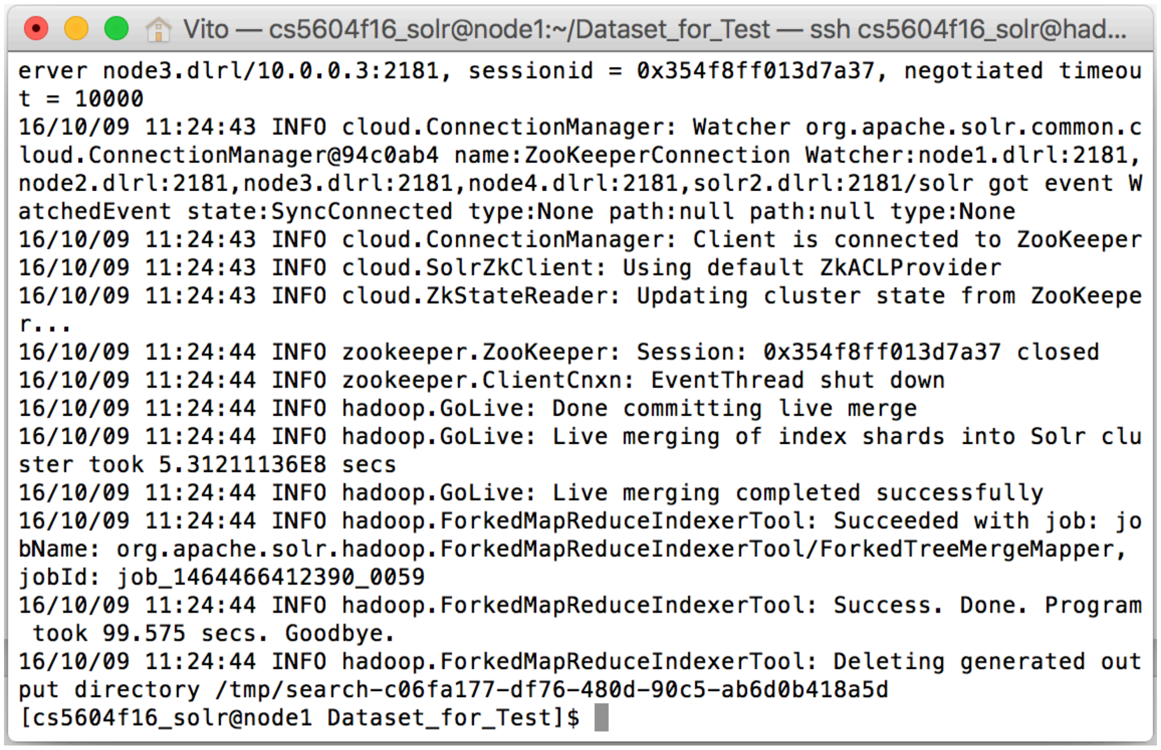

Vito — cs5604f16_solr@node1:~/Dataset_for_Test — ssh cs5604f16_solr@had...

```
<?xml version="1.0"?>
<indexer table="test_table"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper">

   <!-- The relative or absolute path on the local file system to the
   morphline configuration file. -->
   <!-- Use relative path "morphlines.conf" for morphlines managed by
   Cloudera Manager -->
   <param name="morphlineFile" value="/home/cs5604f16_solr/Dataset_for_Test/morp
hlines.conf"/>

   <!-- The optional morphlineId identifies a morphline if there are multiple
   morphlines in morphlines.conf -->
   <!-- <param name="morphlineId" value="morphline1"/> -->

</indexer>
```

Figure 45 Modify Key Morphline Files in Hadoop Cluster

Run HBaseMapReduceIndexer tool. After that, we can index the data with the command below. [LOCAL_DIR] and [COLLECTION_NAME] should be replaced with the correct names, and notice the address of the server is different from the one in Virtual Cloudera (i.e., 127.0.0.1/solr).

Key Command(s)
hadoop --config /etc/hadoop/conf jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-1.5-cdh5.6.0-job.jar --conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx1024m' --hbase-indexer-file [LOCAL_DIR]/morphline-hbase-mapper.xml --zk-host node1.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node4.dlrl:2181,solr2.dlrl:2181/solr --collection [COLLECTION_NAME] --go-live --log4j [LOCAL_DIR]/log4j.properties

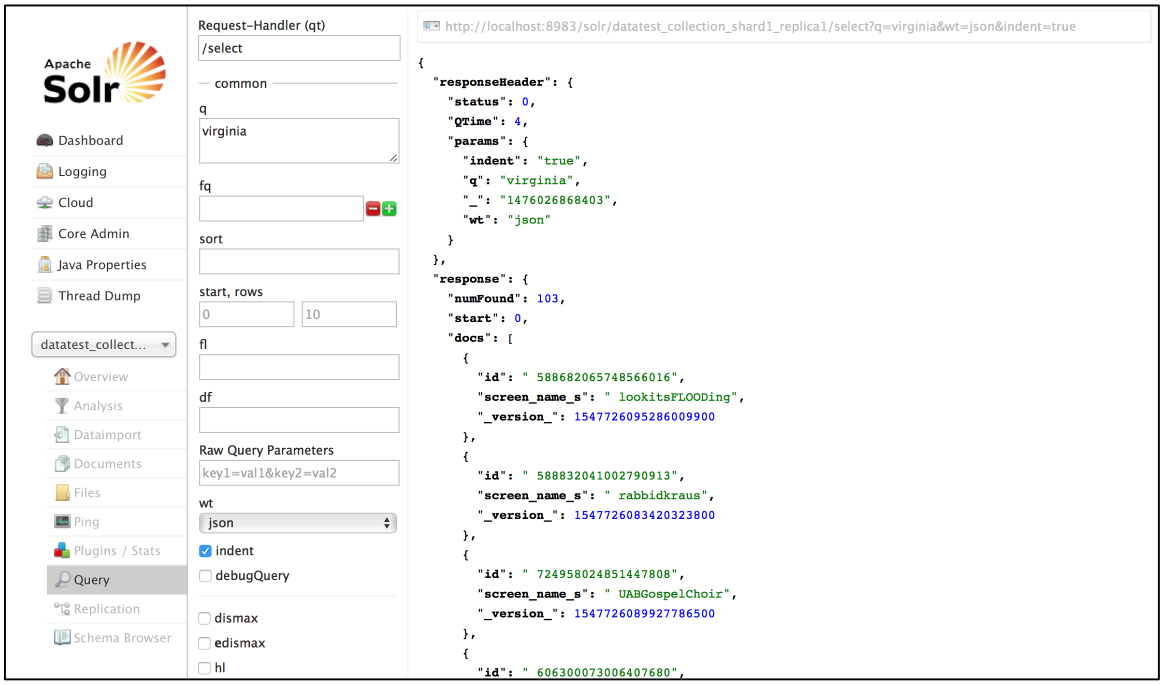


Vito — cs5604f16_solr@node1:~/Dataset_for_Test — ssh cs5604f16_solr@had...

```
erver node3.dlrl/10.0.0.3:2181, sessionid = 0x354f8ff013d7a37, negotiated timeou
t = 10000
16/10/09 11:24:43 INFO cloud.ConnectionManager: Watcher org.apache.solr.common.c
loud.ConnectionManager@94c0ab4 name:ZooKeeperConnection Watcher:node1.dlrl:2181,
node2.dlrl:2181,node3.dlrl:2181,node4.dlrl:2181,solr2.dlrl:2181/solr got event W
atchedEvent state:SyncConnected type:None path:null path:null type:None
16/10/09 11:24:43 INFO cloud.ConnectionManager: Client is connected to ZooKeeper
16/10/09 11:24:43 INFO cloud.SolrZkClient: Using default ZkACLProvider
16/10/09 11:24:43 INFO cloud.ZkStateReader: Updating cluster state from ZooKeepe
r...
16/10/09 11:24:44 INFO zookeeper.ZooKeeper: Session: 0x354f8ff013d7a37 closed
16/10/09 11:24:44 INFO zookeeper.ClientCnxn: EventThread shut down
16/10/09 11:24:44 INFO hadoop.GoLive: Done committing live merge
16/10/09 11:24:44 INFO hadoop.GoLive: Live merging of index shards into Solr clu
ster took 5.31211136E8 secs
16/10/09 11:24:44 INFO hadoop.GoLive: Live merging completed successfully
16/10/09 11:24:44 INFO hadoop.ForkedMapReduceIndexerTool: Succeeded with job: jo
bName: org.apache.solr.hadoop.ForkedMapReduceIndexerTool/ForkedTreeMergeMapper,
jobId: job_1464466412390_0059
16/10/09 11:24:44 INFO hadoop.ForkedMapReduceIndexerTool: Success. Done. Program
 took 99.575 secs. Goodbye.
16/10/09 11:24:44 INFO hadoop.ForkedMapReduceIndexerTool: Deleting generated out
put directory /tmp/search-c06fa177-df76-480d-90c5-ab6d0b418a5d
[cs5604f16_solr@node1 Dataset_for_Test]$ 
```

Figure 46 Generate the Index Files with Live Mode in Hadoop Cluster

Now you can do a query search in the Solr Admin UI.



Request-Handler (qt): /select

common

q: virginia

fq:

sort:

start, rows: 0 10

fl:

df:

Raw Query Parameters: key1=val1&key2=val2

wt: json

[✓ indent]

debugQuery

dismax

edismax

hl

http://localhost:8983/solr/datatest_collection_shard1_replica1/select?q=virginia&wt=json&indent=true

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 4,
    "params": {
      "indent": "true",
      "q": "virginia",
      "_": "1476026868403",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 103,
    "start": 0,
    "docs": [
      {
        "id": " 588682065748566016",
        "screen_name_s": " lookitsFLOODing",
        "_version_": 1547726095286009900
      },
      {
        "id": " 588832041002790913",
        "screen_name_s": " rabbidkraus",
        "_version_": 1547726083420323800
      },
      {
        "id": " 724958024851447808",
        "screen_name_s": " UABGospelChoir",
        "_version_": 1547726089927786500
      },
      {
        "id": " 606300073006407680",
```

Figure 47 Basic Indexing in Hadoop Cluster (Live Mode)

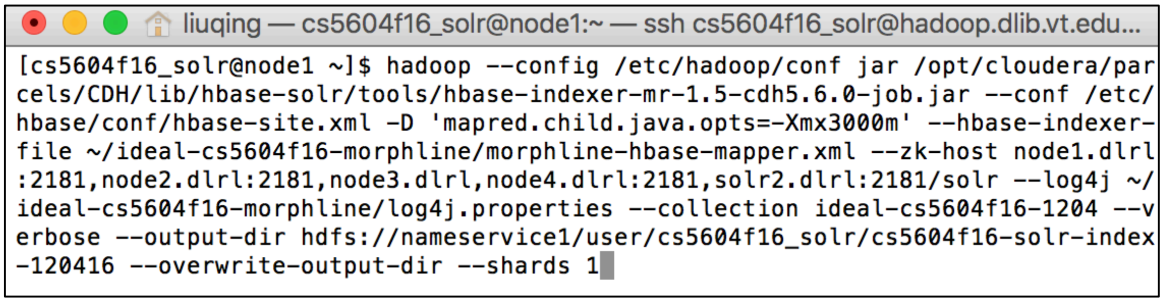
8.3.4 Use Lily Indexer for Data Indexing (Batch Mode)

In batch mode, the Lily Indexer generates the index, and then we need to copy the generated index to the collection index directory in HDFS.

Create a Lily Indexer configuration file. This step is the same as the one in live mode.

Run HBaseMapReduceIndexer tool. We index the HBase table using MapReduce in batch mode. The following command helps us to output the indexes into HDFS.

Key Command(s)
hadoop --config /etc/hadoop/conf jar /opt/cloudera/parcels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-1.5-cdh5.6.0-job.jar --conf /etc/hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx3000m' --hbase-indexer-file [LOCAL_DIR]/morphline-hbase-mapper.xml --zk-host node1.dlrl:2181,node2.dlrl:2181,node3.dlrl,node4.dlrl:2181,solr2.dlrl:2181/solr --log4j [LOCAL_DIR]/log4j.properties --collection [COLLECTION_NAME] --verbose --output-dir [OUTPUT_DIR] --overwrite-output-dir --shards 1

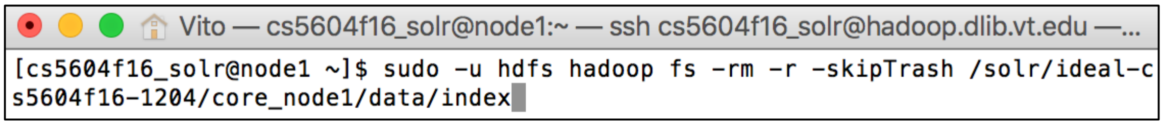


```
[cs5604f16_solr@node1 ~]$ hadoop --config /etc/hadoop/conf jar /opt/cloudera/par
cels/CDH/lib/hbase-solr/tools/hbase-indexer-mr-1.5-cdh5.6.0-job.jar --conf /etc/
hbase/conf/hbase-site.xml -D 'mapred.child.java.opts=-Xmx3000m' --hbase-indexer-
file ~/ideal-cs5604f16-morphline/morphline-hbase-mapper.xml --zk-host node1.dlrl
:2181,node2.dlrl:2181,node3.dlrl,node4.dlrl:2181,solr2.dlrl:2181/solr --log4j ~/
ideal-cs5604f16-morphline/log4j.properties --collection ideal-cs5604f16-1204 --v
erbose --output-dir hdfs://nameservice1/user/cs5604f16_solr/cs5604f16-solr-index
-120416 --overwrite-output-dir --shards 1
```

Figure 48 Generate the Index Files with Batch Mode in Hadoop Cluster

Clean a Solr collection. Now we try to use these offline indexing files to redo the indexing job. As mentioned above, we need to remove all documents from the existing Solr collection. The Solr should be turned off for a more stable process.

Key Command(s)
sudo -u hdfs hadoop fs -rm -r -skipTrash /solr/ideal-cs5604f16-1204/core_node1/data/index

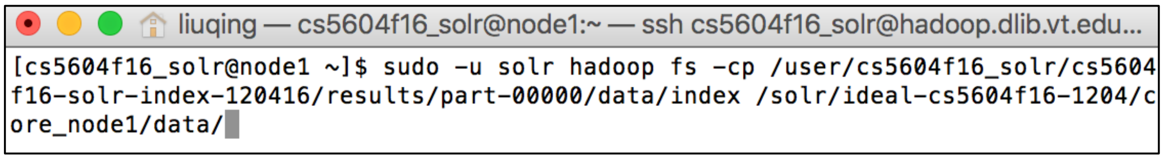


```
[cs5604f16_solr@node1 ~]$ sudo -u hdfs hadoop fs -rm -r -skipTrash /solr/ideal-c
s5604f16-1204/core_node1/data/index
```

Figure 49 Clean a Solr Collection in Hadoop Cluster

Put the offline indexing files into the Solr collection and restart the Solr service. Please notice that we use the role Solr to put into /solr.

Key Command(s)
sudo -u solr hadoop fs -cp /user/cs5604f16_solr/cs5604f16-solr-index-120416/results/part-00000/data/index /solr/ideal-cs5604f16-1204/core_node1/data/

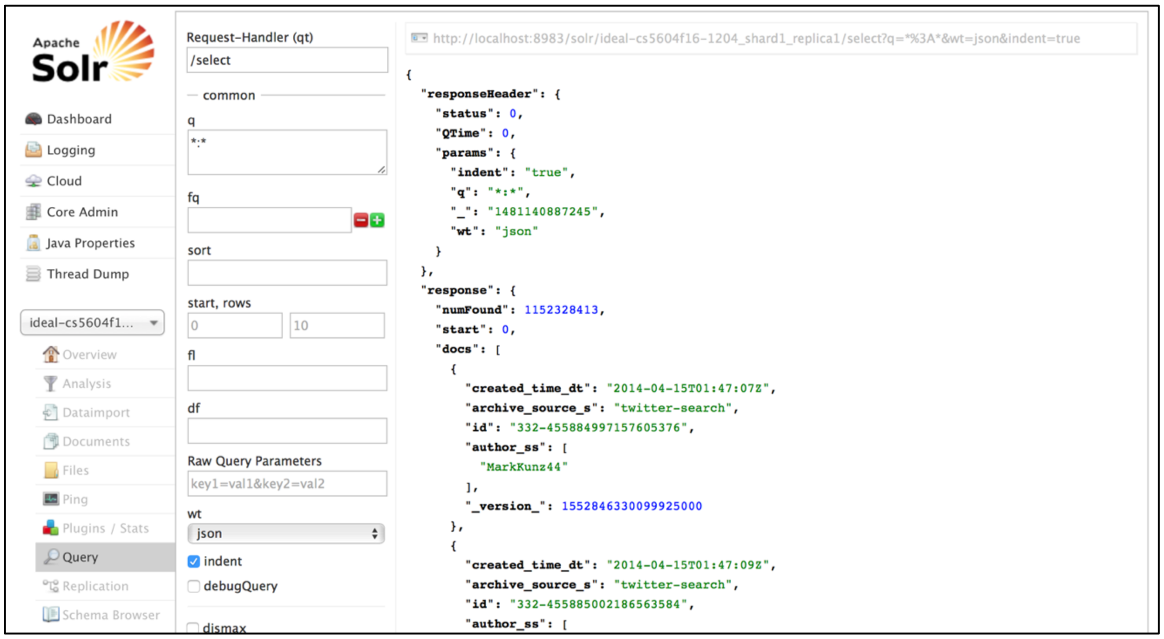


```
[cs5604f16_solr@node1 ~]$ sudo -u solr hadoop fs -cp /user/cs5604f16_solr/cs5604
f16-solr-index-120416/results/part-00000/data/index /solr/ideal-cs5604f16-1204/c
ore_node1/data/
```

liuqing — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu...

Figure 50 Move the Index Files from User Directory to Solr in Hadoop Cluster

After the copying is finished, we need to restart Solr. Then, we can check the collection in the Solr Admin UI.



Request-Handler (qt): /select
q: *:*
fq:
sort:
start, rows: 0, 10
fl:
df:
Raw Query Parameters: key1=val1&key2=val2
wt: json
[✓ indent]
debugQuery

http://localhost:8983/solr/ideal-cs5604f16-1204_shard1_replica1/select?q=*%3A*&wt=json&indent=true

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "indent": "true",
      "q": "*:*",
      "_": "1481140887245",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 1152328413,
    "start": 0,
    "docs": [
      {
        "created_time_dt": "2014-04-15T01:47:07Z",
        "archive_source_s": "twitter-search",
        "id": "332-455884997157605376",
        "author_ss": [
          "MarkKunz44"
        ],
        "_version_": 1552846330099925000
      },
      {
        "created_time_dt": "2014-04-15T01:47:09Z",
        "archive_source_s": "twitter-search",
        "id": "332-455885002186563584",
        "author_ss": [
```

Figure 51 Basic Indexing in Hadoop Cluster (Batch Mode)

8.4 Tutorial for Incremental Indexing in Virtual Cloudera

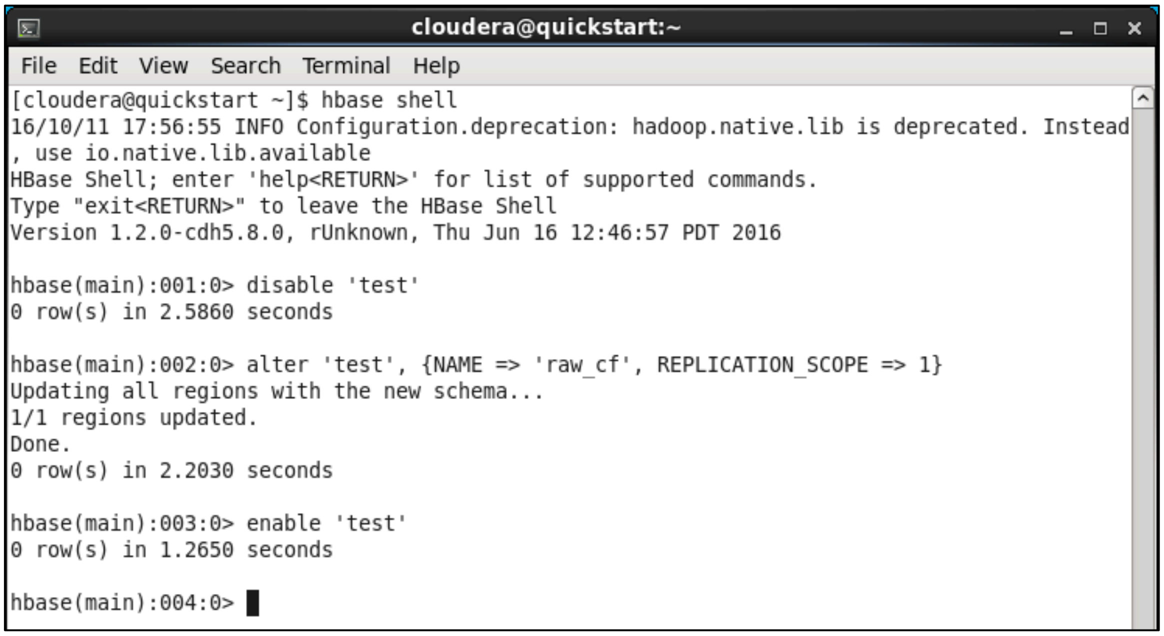
Now we know that we can use the Lily Indexer for data indexing in both live mode and batch mode. To index a small size of data, live mode is a simpler and more efficient approach. However, it cannot deal with big data. Instead of using the live mode, the batch mode is a good choice to index a huge amount of data, though it is complex and takes more time. Therefore, there are some limitations while applying the batch mode of Lily indexer. Specifically, people need to apply frequent inserts, updates, and deletes to HBase table cells. If we still use the batch mode to index all the data, it seems an endless job. In this case, Lily HBase NRT indexer can be used to process a continuous stream of HBase cell updates into live search indexes.

Our incremental indexing job is based on the basic indexing. The following steps present how to deploy the NRT indexer in Virtual Cloudera.

8.4.1 Enable replication on HBase column families

Ensure that cluster-wide HBase replication is enabled. Use the HBase shell to define column-family replication settings. For every existing table, set the REPLICATION_SCOPE on every column family that needs to be indexed.

Key Command(s)
disable ‘test’ alter 'test', {NAME => 'raw_cf', REPLICATION_SCOPE => 1} enable ‘test’



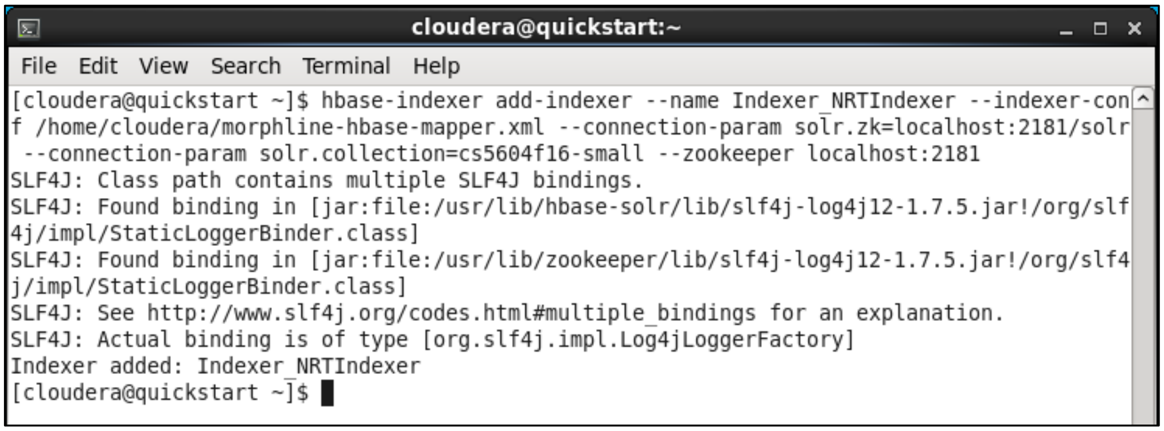
```
[cloudera@quickstart ~]$ hbase shell
16/10/11 17:56:55 INFO Configuration.deprecation: hadoop.native.lib is deprecated. Instead
, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.8.0, rUnknown, Thu Jun 16 12:46:57 PDT 2016

hbase(main):001:0> disable 'test'
0 row(s) in 2.5860 seconds

hbase(main):002:0> alter 'test', {NAME => 'raw_cf', REPLICATION_SCOPE => 1}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.2030 seconds

hbase(main):003:0> enable 'test'
0 row(s) in 1.2650 seconds

hbase(main):004:0>
```

Figure 52 Enable HBase Replication in Virtual Cloudera

8.4.2 Register a Lily HBase Indexer

Once the content of the Lily HBase Indexer configuration XML file is satisfactory, register it with the Lily HBase Indexer Service.

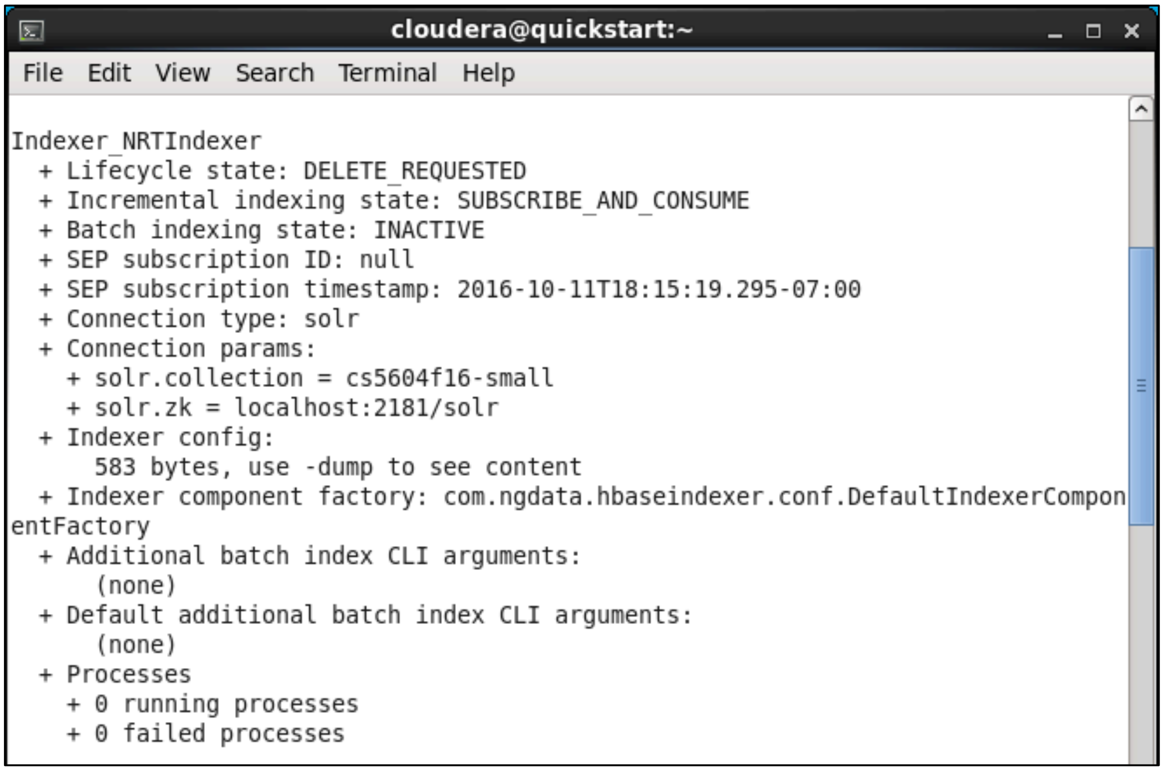
Key Command(s)
hbase-indexer add-indexer --name Indexer_NRTIndexer --indexer-conf [LOCAL_DIR]/morphline-hbase-mapper.xml --connection-param solr.zk=localhost:2181/solr --connection-param solr.collection=[COLLECTION_NAME] --zookeeper localhost:2181 hbase-indexer list-indexers



```
[cloudera@quickstart ~]$ hbase-indexer add-indexer --name Indexer_NRTIndexer --indexer-con
f /home/cloudera/morphline-hbase-mapper.xml --connection-param solr.zk=localhost:2181/solr
 --connection-param solr.collection=cs5604f16-small --zookeeper localhost:2181
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hbase-solr/lib/slf4j-log4j12-1.7.5.jar!/org/slf
4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4
j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Indexer added: Indexer_NRTIndexer
[cloudera@quickstart ~]$ 
```

Figure 53 Register a Lily HBase Indexer in Virtual Cloudera

Verify that the indexer was successfully created as follows:



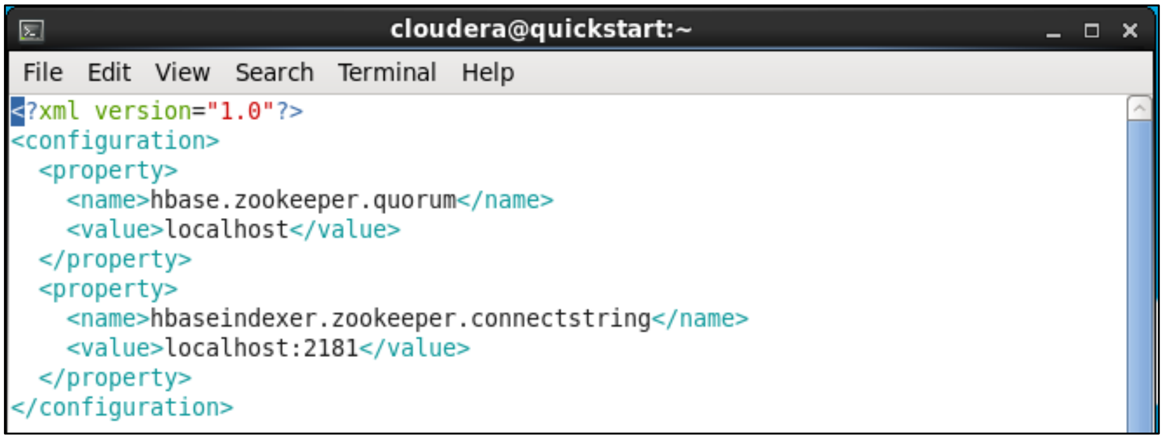
```
Indexer_NRTIndexer
  + Lifecycle state: DELETE_REQUESTED
  + Incremental indexing state: SUBSCRIBE_AND_CONSUME
  + Batch indexing state: INACTIVE
  + SEP subscription ID: null
  + SEP subscription timestamp: 2016-10-11T18:15:19.295-07:00
  + Connection type: solr
  + Connection params:
    + solr.collection = cs5604f16-small
    + solr.zk = localhost:2181/solr
  + Indexer config:
      583 bytes, use -dump to see content
  + Indexer component factory: com.ngdata.hbaseindexer.conf.DefaultIndexerCompon
entFactory
  + Additional batch index CLI arguments:
      (none)
  + Default additional batch index CLI arguments:
      (none)
  + Processes
    + 0 running processes
    + 0 failed processes
```

Figure 54 Verify a Lily HBase Indexer in Virtual Cloudera

8.4.3 Point a Lily HBase NRT Indexer Service

Configure individual Lily HBase NRT Indexer Services with the location of a ZooKeeper ensemble that is used for the target HBase cluster. This must be done before starting Lily HBase NRT Indexer Services. Add the following property to /etc/hbase-solr/conf/hbase-indexer-site.xml. Remember to replace hbase-cluster-zookeeper with the actual ensemble string as found in the hbase-site.xml configuration file.

Key Command(s)
sudo vim /etc/hbase-solr/conf/hbase-indexer-site.xml



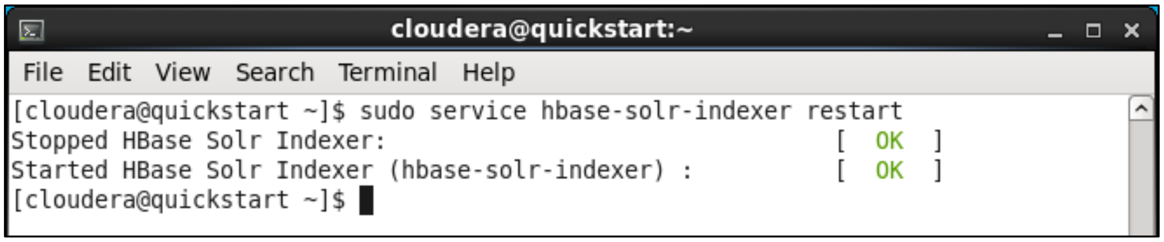
```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>localhost</value>
  </property>
  <property>
    <name>hbaseindexer.zookeeper.connectstring</name>
    <value>localhost:2181</value>
  </property>
</configuration>
```

Figure 55 Point a Lily HBase NRT Indexer Service in Virtual Cloudera

8.4.4 Start a Lily HBase NRT Indexer Service, and Testing

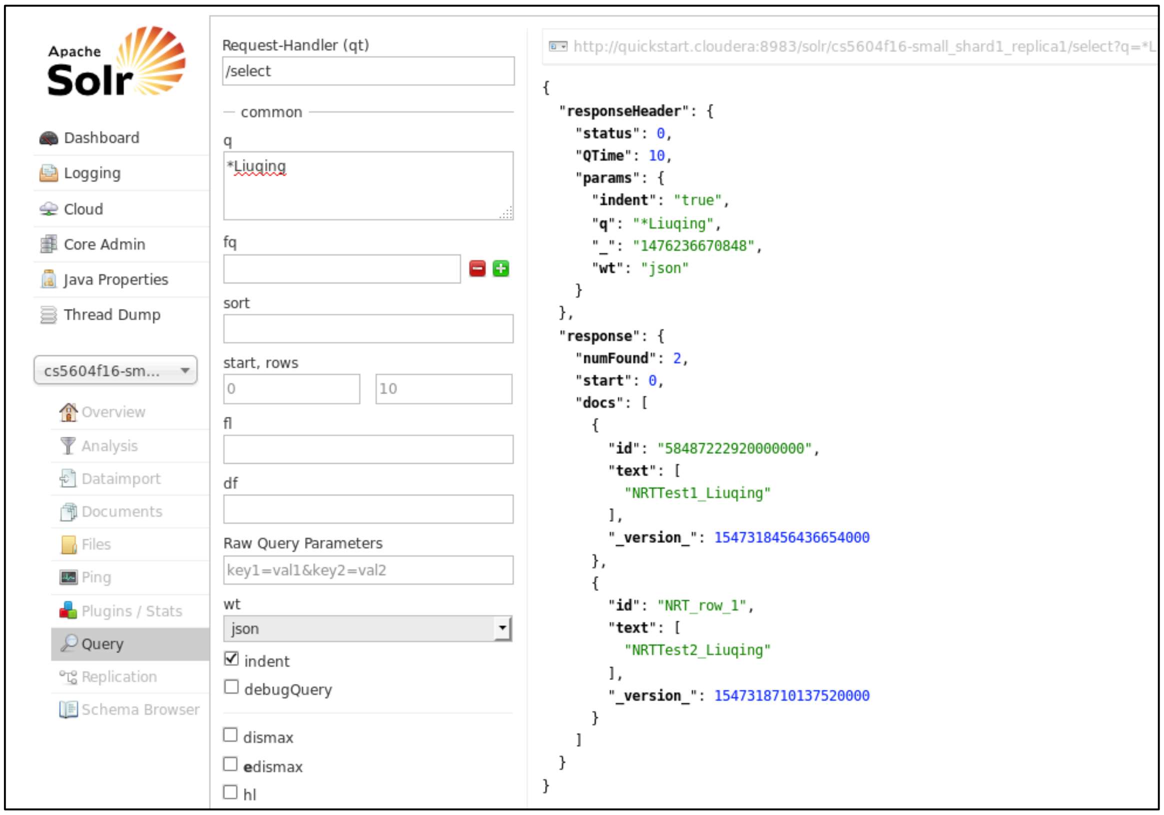
Now, we can restart the indexer service and manually insert some records into HBase. The incremental data can be indexed and retrieved from the Solr Admin UI.

Key Command(s)
sudo service hbase-solr-indexer restart



```
[cloudera@quickstart ~]$ sudo service hbase-solr-indexer restart
Stopped HBase Solr Indexer:                                [  OK  ]
Started HBase Solr Indexer (hbase-solr-indexer) :          [  OK  ]
[cloudera@quickstart ~]$ 
```

Figure 56 Start a Lily HBase NRT Indexer Service in Virtual Cloudera



Request-Handler (qt): /select
q: *Liuqing
start, rows: 0 10
wt: json
☑ indent

http://quickstart.cloudera:8983/solr/cs5604f16-small_shard1_replica1/select?q=*L

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 10,
    "params": {
      "indent": "true",
      "q": "*Liuqing",
      "_": "1476236670848",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 2,
    "start": 0,
    "docs": [
      {
        "id": "584872229200000000",
        "text": [
          "NRTTest1_Liuqing"
        ],
        "_version_": 1547318456436654000
      },
      {
        "id": "NRT_row_1",
        "text": [
          "NRTTest2_Liuqing"
        ],
        "_version_": 1547318710137520000
      }
    ]
  }
}
```

Figure 57 Incremental Indexing in Virtual Cloudera

8.5 Tutorial for Incremental Indexing in Hadoop Cluster

To configure the incremental indexing in the Hadoop Cluster is similar to the process in Virtual Cloudera. Since the commands are a bit different, we follow the same steps to make the procedure much more clear.

8.5.1 Enable replication on HBase column families

Ensure that cluster-wide HBase replication is enabled. Use the HBase shell to define column-family replication settings. For every existing table, set the REPLICATION_SCOPE on every column family that needs to be indexed.

Key Command(s)

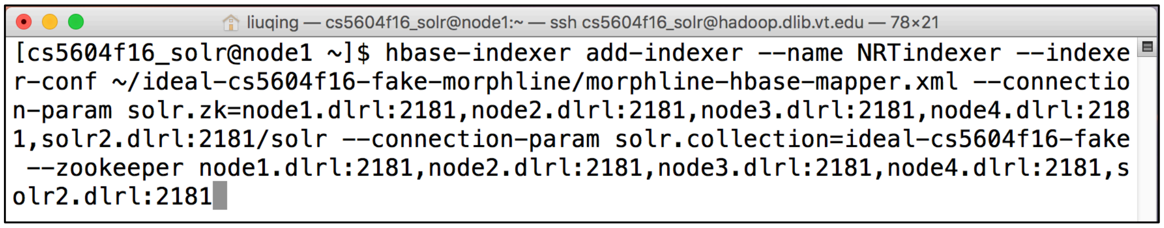
```
disable 'ideal-cs5604-fake'
alter 'ideal-cs5604-fake', {NAME => 'tweet', REPLICATION_SCOPE => 1}
alter 'ideal-cs5604-fake', {NAME => 'clean-tweet', REPLICATION_SCOPE => 1}
alter 'ideal-cs5604-fake', {NAME => 'tweet-topic', REPLICATION_SCOPE => 1}
alter 'ideal-cs5604-fake', {NAME => 'tweet-cluster', REPLICATION_SCOPE => 1}
alter 'ideal-cs5604-fake', {NAME => 'webpage', REPLICATION_SCOPE => 1}
enable 'ideal-cs5604-fake'
```

Figure 58 Enable HBase Replication in Hadoop Cluster

8.5.2 Register a Lily HBase Indexer

Once the content of the Lily HBase Indexer configuration XML file is satisfactory, register it with the Lily HBase Indexer Service.

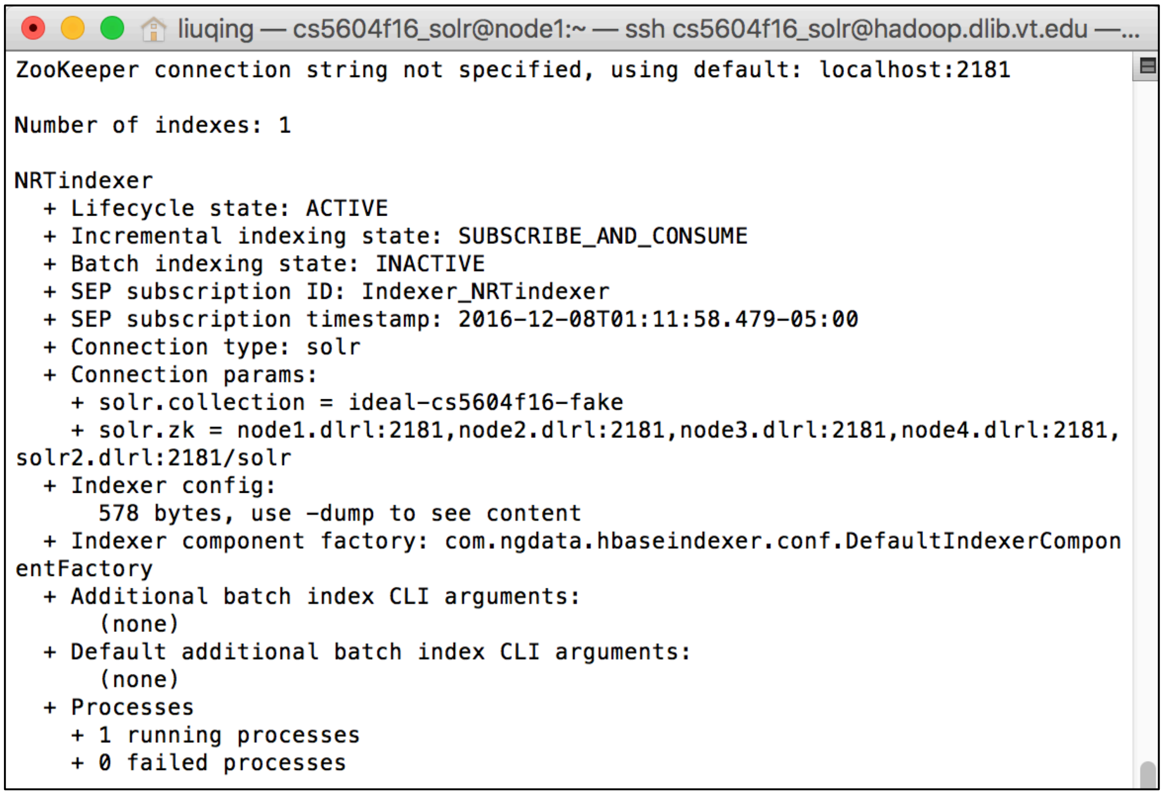
Key Command(s)
hbase-indexer add-indexer --name NRTindexer --indexer-conf ~/ideal-cs5604f16-fake-morphline/morphline-hbase-mapper.xml --connection-param solr.zk=node1.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node4.dlrl:2181,solr2.dlrl:2181/solr --connection-param solr.collection=ideal-cs5604f16-fake --zookeeper node1.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node4.dlrl:2181,solr2.dlrl:2181


liuqing — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu — 78×21

```
[cs5604f16_solr@node1 ~]$ hbase-indexer add-indexer --name NRTindexer --indexe
r-conf ~/ideal-cs5604f16-fake-morphline/morphline-hbase-mapper.xml --connectio
n-param solr.zk=node1.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node4.dlrl:218
1,solr2.dlrl:2181/solr --connection-param solr.collection=ideal-cs5604f16-fake
 --zookeeper node1.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node4.dlrl:2181,s
olr2.dlrl:2181
```

Figure 59 Register a Lily HBase Indexer in Hadoop Cluster

Verify that the indexer was successfully created as follows:


liuqing — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu —...

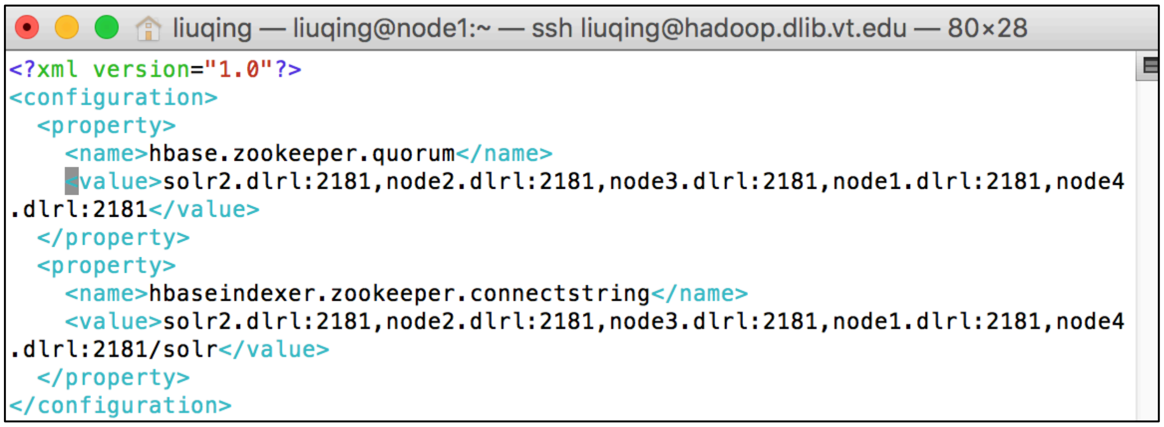
```
ZooKeeper connection string not specified, using default: localhost:2181

Number of indexes: 1

NRTindexer
  + Lifecycle state: ACTIVE
  + Incremental indexing state: SUBSCRIBE_AND_CONSUME
  + Batch indexing state: INACTIVE
  + SEP subscription ID: Indexer_NRTindexer
  + SEP subscription timestamp: 2016-12-08T01:11:58.479-05:00
  + Connection type: solr
  + Connection params:
    + solr.collection = ideal-cs5604f16-fake
    + solr.zk = node1.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node4.dlrl:2181,
solr2.dlrl:2181/solr
  + Indexer config:
      578 bytes, use -dump to see content
  + Indexer component factory: com.ngdata.hbaseindexer.conf.DefaultIndexerCompon
entFactory
  + Additional batch index CLI arguments:
      (none)
  + Default additional batch index CLI arguments:
      (none)
  + Processes
    + 1 running processes
    + 0 failed processes
```

Figure 60 Verify a Lily HBase Indexer in Hadoop Cluster

8.5.3 Point a Lily HBase NRT Indexer Service

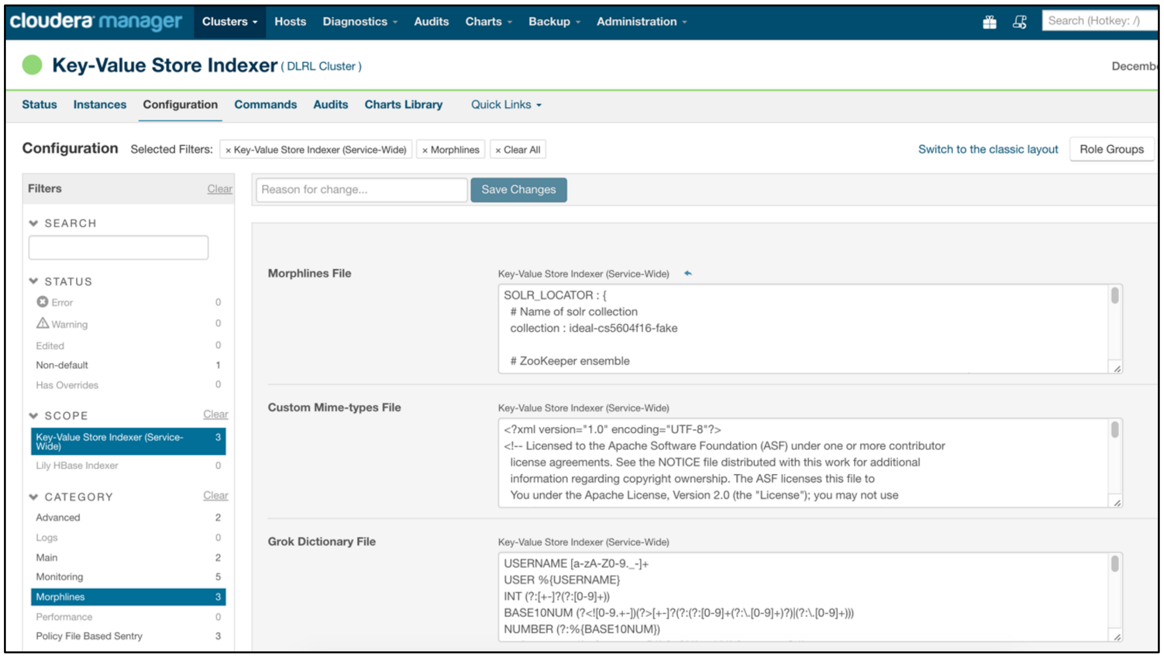
Configure individual Lily HBase NRT Indexer Services with the location of a ZooKeeper ensemble that is used for the target HBase cluster. This must be done before starting Lily HBase NRT Indexer Services. Add the following property to /etc/hbase-solr/conf/hbase-indexer-site.xml. We need to modify the above file in HeadNode. Remember to replace hbase-cluster-zookeeper with the actual ensemble string as found in the hbase-site.xml configuration file.

Key Command(s)
sudo vim /etc/hbase-solr/conf/hbase-indexer-site.xml


liuqing — liuqing@node1:~ — ssh liuqing@hadoop.dlib.vt.edu — 80×28

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>solr2.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node1.dlrl:2181,node4
.dlrl:2181</value>
  </property>
  <property>
    <name>hbaseindexer.zookeeper.connectstring</name>
    <value>solr2.dlrl:2181,node2.dlrl:2181,node3.dlrl:2181,node1.dlrl:2181,node4
.dlrl:2181/solr</value>
  </property>
</configuration>
```

Here, we need to complete one more step. In the Cloudera Manager, we need to overwrite the same morphlines.conf into the Key-Value Store Indexer.



Morphlines File — Key-Value Store Indexer (Service-Wide)

```
SOLR_LOCATOR : {
  # Name of solr collection
  collection : ideal-cs5604f16-fake

  # ZooKeeper ensemble
```

Custom Mime-types File — Key-Value Store Indexer (Service-Wide)

```
<?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
```

Grok Dictionary File — Key-Value Store Indexer (Service-Wide)

```
USERNAME [a-zA-Z0-9._-]+
USER %{USERNAME}
INT (?:[+-]?(?:[0-9]+))
BASE10NUM (?<![0-9.+-])(?>[+-]?(?:(?:[0-9]+(?:\.[0-9]+)?)|(?:\.[0-9]+)))
NUMBER (?:%{BASE10NUM})
```

Figure 61 Point a Lily HBase NRT Indexer Service in Hadoop Cluster

8.5.4 Start a Lily HBase NRT Indexer Service and Testing

Now, we need to restart the indexer service through Cloudera Manager and manually insert some records into HBase. The incremental data can be indexed and retrieved from the Solr Admin UI.

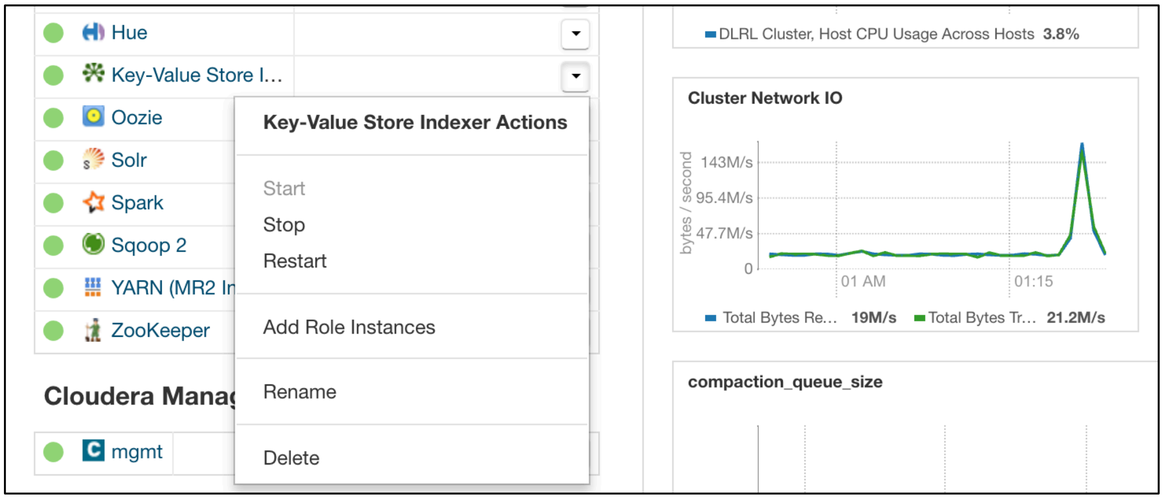
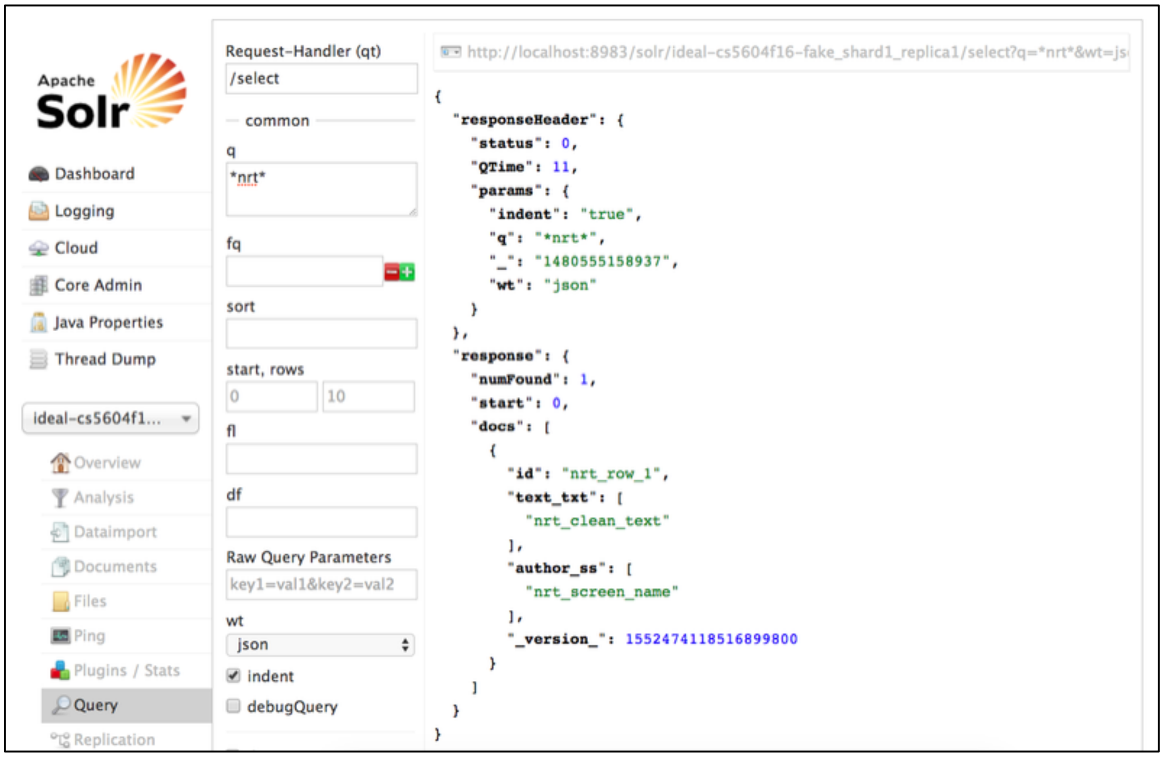


Figure 62 Start a Lily HBase NRT Indexer Service in Hadoop Cluster



Request-Handler (qt): /select
q: *nrt*
fq:
sort:
start, rows: 0 10
fl:
df:
Raw Query Parameters: key1=val1&key2=val2
wt: json
indent ☑
debugQuery ☐

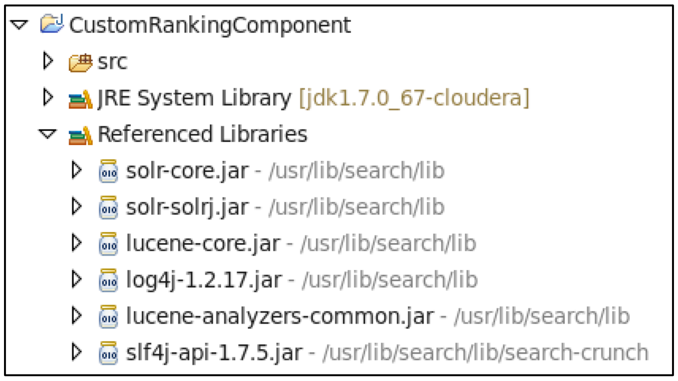
```
http://localhost:8983/solr/ideal-cs5604f16-fake_shard1_replica1/select?q=*nrt*&wt=js
{
  "responseHeader": {
    "status": 0,
    "QTime": 11,
    "params": {
      "indent": "true",
      "q": "*nrt*",
      "_": "1480555158937",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 1,
    "start": 0,
    "docs": [
      {
        "id": "nrt_row_1",
        "text_txt": [
          "nrt_clean_text"
        ],
        "author_ss": [
          "nrt_screen_name"
        ],
        "_version_": 1552474118516899800
      }
    ]
  }
}
```

Figure 63 Incremental Indexing in Hadoop Cluster

8.6 Tutorial for Custom Ranking

8.6.1 Build and copy jar file into the Hadoop Cluster

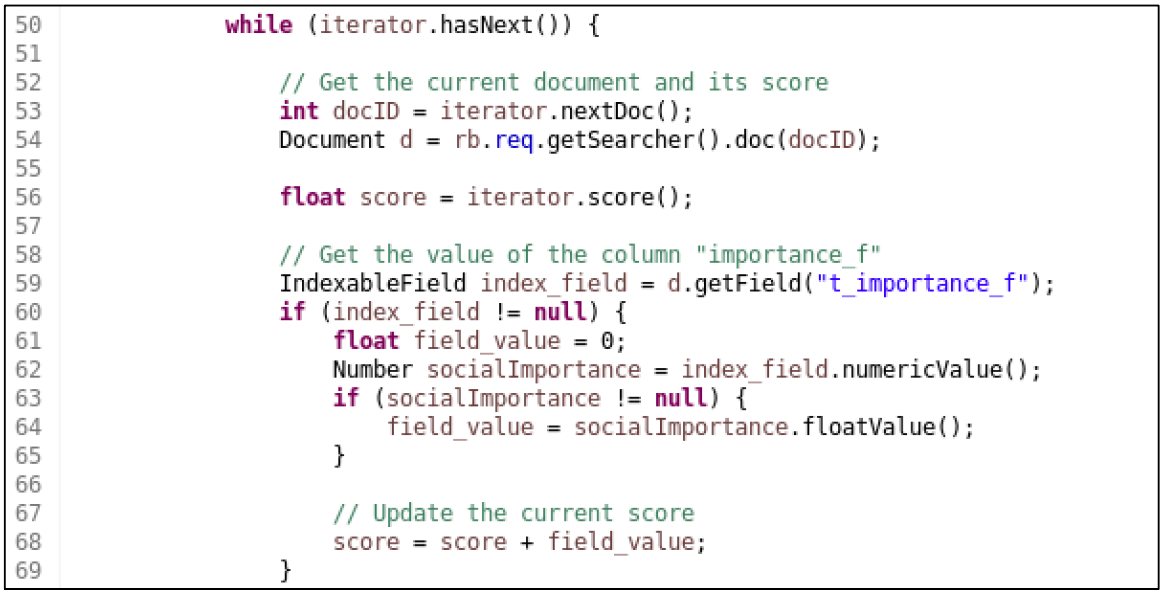
To build the jar file, we need to create a Java project in Virtual Cloudera through Eclipse. Then, based on the developer's requirement, some dependent libraries should be imported into the same project. For the current custom ranking function, we import the following libraries that are shown in Figure 64.



```
CustomRankingComponent
  src
  JRE System Library [jdk1.7.0_67-cloudera]
  Referenced Libraries
    solr-core.jar - /usr/lib/search/lib
    solr-solrj.jar - /usr/lib/search/lib
    lucene-core.jar - /usr/lib/search/lib
    log4j-1.2.17.jar - /usr/lib/search/lib
    lucene-analyzers-common.jar - /usr/lib/search/lib
    slf4j-api-1.7.5.jar - /usr/lib/search/lib/search-crunch
```

Figure 64 External Libraries for Custom Ranking

After doing that, we can design and implement our own Search Component. The importance value of each tweet could be caught and added to the original Solr score. The jar file will be exported after finishing the coding job.



```java
            while (iterator.hasNext()) {

                // Get the current document and its score
                int docID = iterator.nextDoc();
                Document d = rb.req.getSearcher().doc(docID);

                float score = iterator.score();

                // Get the value of the column "importance_f"
                IndexableField index_field = d.getField("t_importance_f");
                if (index_field != null) {
                    float field_value = 0;
                    Number socialImportance = index_field.numericValue();
                    if (socialImportance != null) {
                        field_value = socialImportance.floatValue();
                    }

                    // Update the current score
                    score = score + field_value;
                }
```

Figure 65 Code Fragment of Custom Ranking

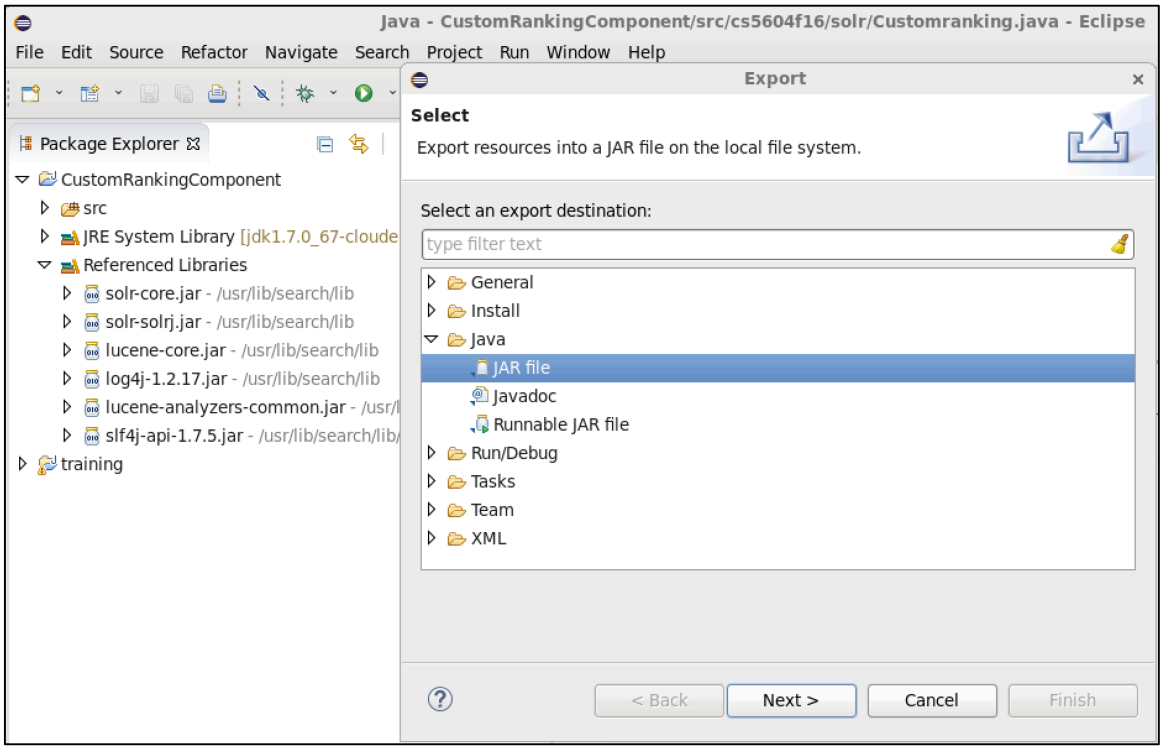
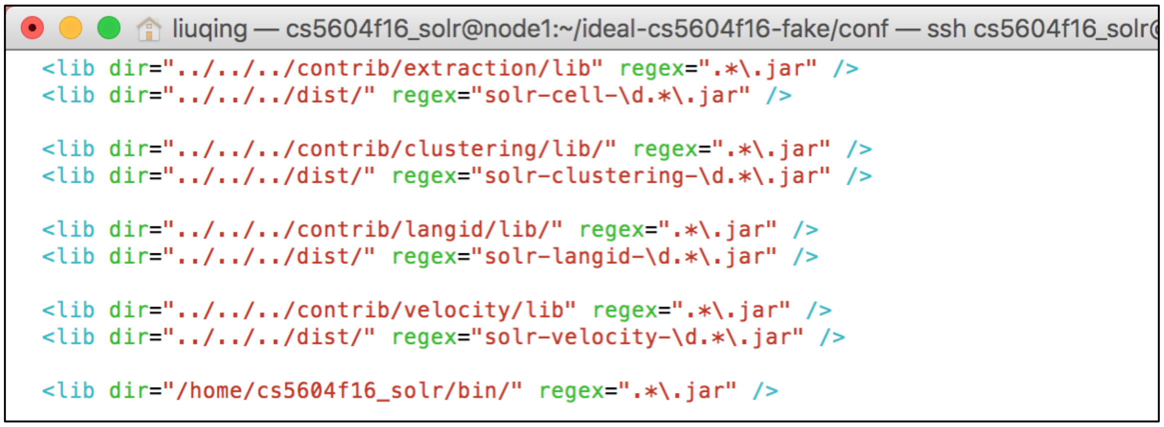

Java - CustomRankingComponent/src/cs5604f16/solr/Customranking.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
CustomRankingComponent
src
JRE System Library [jdk1.7.0_67-cloude
Referenced Libraries
solr-core.jar - /usr/lib/search/lib
solr-solrj.jar - /usr/lib/search/lib
lucene-core.jar - /usr/lib/search/lib
log4j-1.2.17.jar - /usr/lib/search/lib
lucene-analyzers-common.jar - /usr/l
slf4j-api-1.7.5.jar - /usr/lib/search/lib/
training
Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
Install
Java
JAR file
Javadoc
Runnable JAR file
Run/Debug
Tasks
Team
XML
< Back
Next >
Cancel
Finish

Figure 66 Export JAR File for Custom Ranking

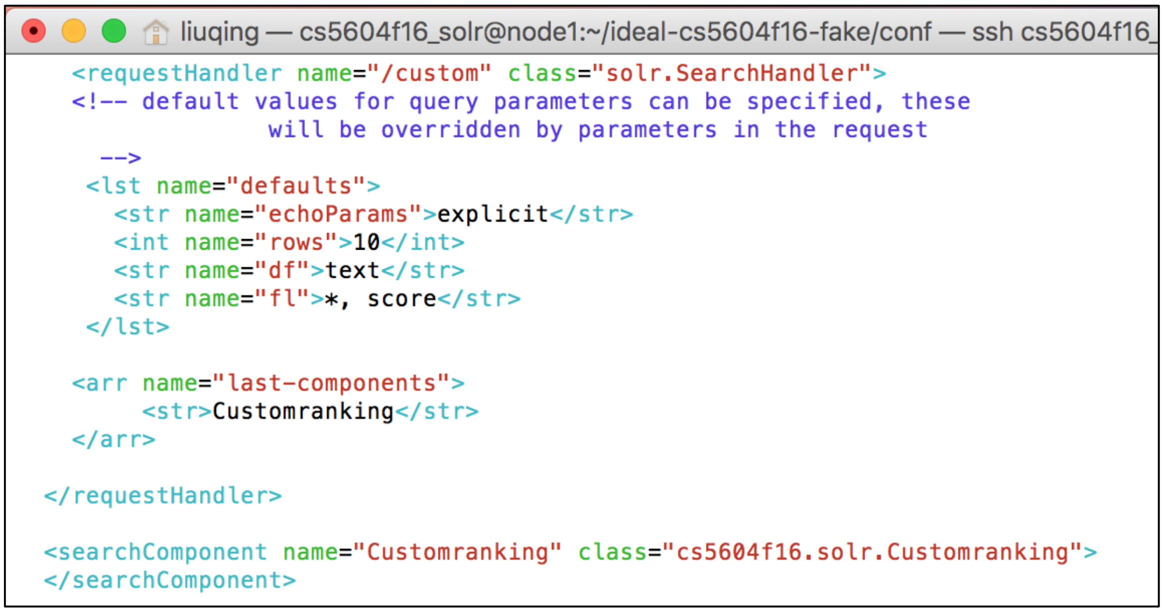
Then we use the scp command to copy the jar file into the Hadoop Cluster (HeadNode and Solr node).

8.6.2 Modify the solrconfig.xml

We have to modify the solrconfig.xml to locate the jar file and create a new search component for the custom ranking.


liuqing — cs5604f16_solr@node1:~/ideal-cs5604f16-fake/conf — ssh cs5604f16_solr@

```
  <lib dir="../../../contrib/extraction/lib" regex=".*\.jar" />
  <lib dir="../../../dist/" regex="solr-cell-\d.*\.jar" />

  <lib dir="../../../contrib/clustering/lib/" regex=".*\.jar" />
  <lib dir="../../../dist/" regex="solr-clustering-\d.*\.jar" />

  <lib dir="../../../contrib/langid/lib/" regex=".*\.jar" />
  <lib dir="../../../dist/" regex="solr-langid-\d.*\.jar" />

  <lib dir="../../../contrib/velocity/lib" regex=".*\.jar" />
  <lib dir="../../../dist/" regex="solr-velocity-\d.*\.jar" />

  <lib dir="/home/cs5604f16_solr/bin/" regex=".*\.jar" />
```


liuqing — cs5604f16_solr@node1:~/ideal-cs5604f16-fake/conf — ssh cs5604f16_

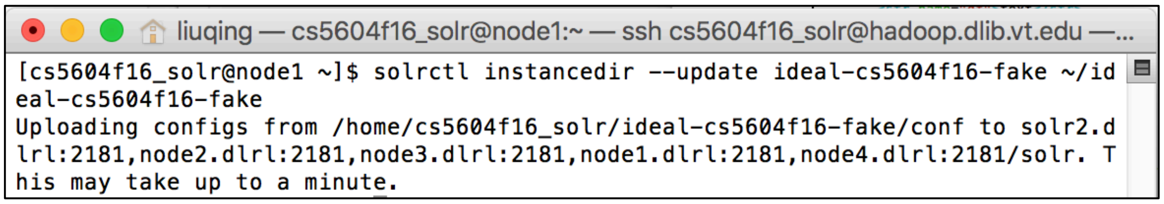
```
  <requestHandler name="/custom" class="solr.SearchHandler">
  <!-- default values for query parameters can be specified, these
               will be overridden by parameters in the request
    -->
   <lst name="defaults">
     <str name="echoParams">explicit</str>
     <int name="rows">10</int>
     <str name="df">text</str>
     <str name="fl">*, score</str>
   </lst>

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

</requestHandler>

<searchComponent name="Customranking" class="cs5604f16.solr.Customranking">
</searchComponent>
```

Figure 67 Add Custom Ranking Component in solrconfig.xml

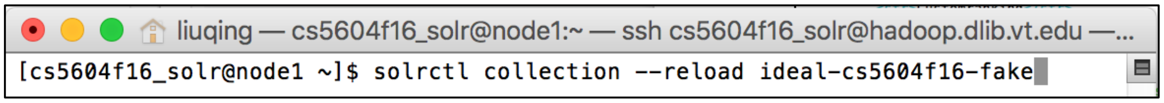
8.6.3 Update the instanceDir and reload the collection

Since the solrconfig.xml has been modified, we need to update the instanceDir to let ZooKeeper configure it in the Hadoop Cluster.

Key Command(s)
solrctl instancedir --update ideal-cs5604f16-fake ~/ideal-cs5604f16-fake solrctl collection --reload ideal-cs5604f16-fake


liuqing — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu —...

```
[cs5604f16_solr@node1 ~]$ solrctl instancedir --update ideal-cs5604f16-fake ~/id
eal-cs5604f16-fake
Uploading configs from /home/cs5604f16_solr/ideal-cs5604f16-fake/conf to solr2.d
lrl:2181,node2.dlrl:2181,node3.dlrl:2181,node1.dlrl:2181,node4.dlrl:2181/solr. T
his may take up to a minute.
```

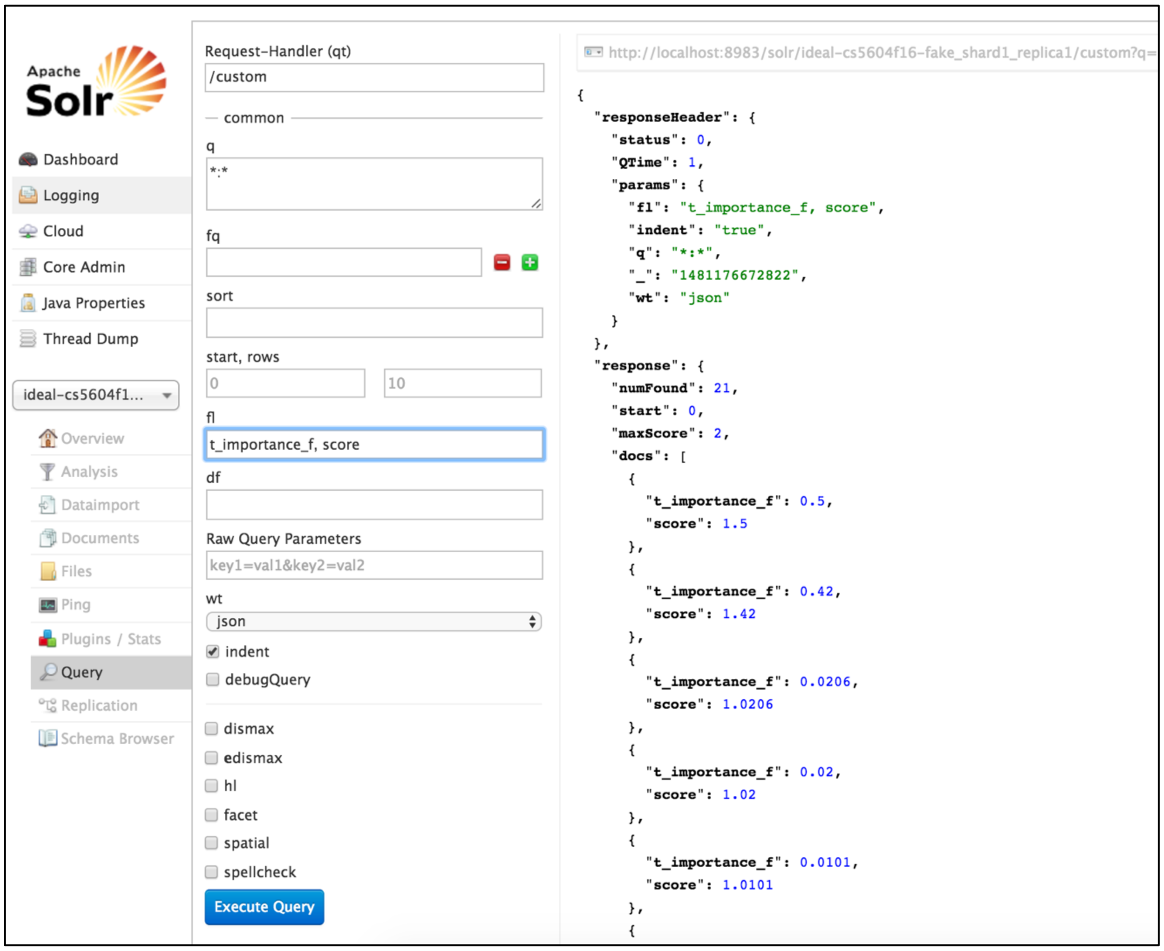

liuqing — cs5604f16_solr@node1:~ — ssh cs5604f16_solr@hadoop.dlib.vt.edu —...

```
[cs5604f16_solr@node1 ~]$ solrctl collection --reload ideal-cs5604f16-fake
```

Figure 68 Reload the Collection for Custom Ranking

8.6.4 Check the results in Solr Admin UI

Finally, we can use the “/custom” request handler to retrieve the results based on our custom ranking function.



Request-Handler (qt): /custom
common
q: `*:*`
fq:
sort:
start, rows: 0 10
fl: t_importance_f, score
df:
Raw Query Parameters: key1=val1&key2=val2
wt: json
[✓ indent]
debugQuery
dismax
edismax
hl
facet
spatial
spellcheck

http://localhost:8983/solr/ideal-cs5604f16-fake_shard1_replica1/custom?q=

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "fl": "t_importance_f, score",
      "indent": "true",
      "q": "*:*",
      "_": "1481176672822",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 21,
    "start": 0,
    "maxScore": 2,
    "docs": [
      {
        "t_importance_f": 0.5,
        "score": 1.5
      },
      {
        "t_importance_f": 0.42,
        "score": 1.42
      },
      {
        "t_importance_f": 0.0206,
        "score": 1.0206
      },
      {
        "t_importance_f": 0.02,
        "score": 1.02
      },
      {
        "t_importance_f": 0.0101,
        "score": 1.0101
      },
      {
```

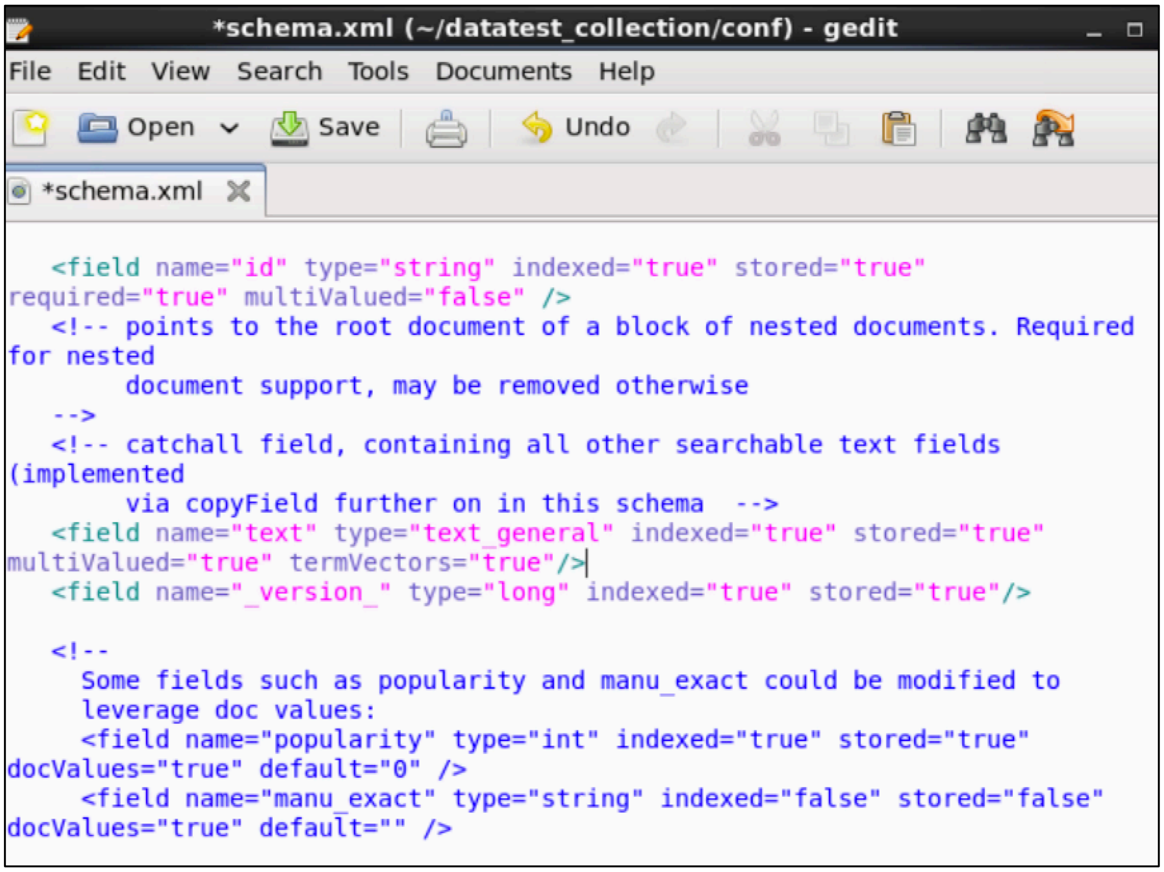
Figure 69 An Example of Custom Ranking

8.7 Tutorial for Document Recommendation

8.7.1 Configurations for MoreLikeThis request handler

The MLT request handler needs addition and modification to schema.xml.

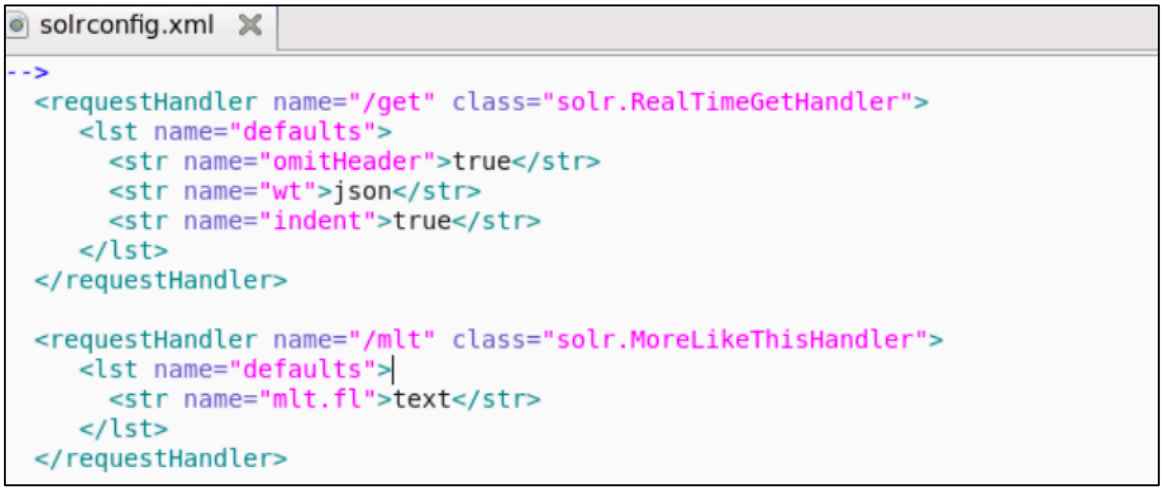
The MLT request handler use the stored term vectors of specified fields to compute similarity [26]. If no term vectors of fields are configured to be stored, MLT will generate term vectors, which takes time, so it is better to configure fields to store term vectors. Add a termVectors attribute to the field that you want to store term vectors, and set the attribute to “true”. In the configuration example below, the field named cat is configured to store term vectors, and in the process of indexing, the term vectors of the “text” field will be stored.


*schema.xml (~/datatest_collection/conf) - gedit

```
   <field name="id" type="string" indexed="true" stored="true"
required="true" multiValued="false" />
   <!-- points to the root document of a block of nested documents. Required
for nested
         document support, may be removed otherwise
   -->
   <!-- catchall field, containing all other searchable text fields
(implemented
         via copyField further on in this schema  -->
   <field name="text" type="text_general" indexed="true" stored="true"
multiValued="true" termVectors="true"/>
   <field name="_version_" type="long" indexed="true" stored="true"/>

   <!--
     Some fields such as popularity and manu_exact could be modified to
     leverage doc values:
     <field name="popularity" type="int" indexed="true" stored="true"
docValues="true" default="0" />
     <field name="manu_exact" type="string" indexed="false" stored="false"
docValues="true" default="" />
```

Figure 70 Add TermVectors in schema.xml for Recommendation

To make Solr able to handle an MLT request, you need to add a request handler configuration to solrconfig.xml. The online Apache Solr Reference Guide [27] explains how to add a request handler in solrconfig.xml.



```
-->
  <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>

  <requestHandler name="/mlt" class="solr.MoreLikeThisHandler">
     <lst name="defaults">
       <str name="mlt.fl">text</str>
     </lst>
  </requestHandler>
```

Figure 71 Add MLT Request Handler in solrconfig.xml for Recommendation

The name attribute of requestHandler defines how you use the HTTP API. For example, in the configuration above, the name attribute is “/mlt”, so in the HTTP API, you can use this request handler like http://localhost:8983/solr/mlt?..., The name before the question mark is the same as the name attribute. Inside the requestHandler element, there is a child element “lst”, whose name attribute is “default”. “Lst” actually mean “list”, and a lst with name attribute “defaults” defines the default parameters for the request handler. These default parameters can be overridden by the parameters given by a request using the HTTP API.

Table 8 explains the common parameters that can be used for MLT. Table 9 explains the parameters for the MoreLikeThisHandler.

Table 8 Parameters and Descriptions for MoreLikeThis [26]

Parameter	Description
mlt.fl	The fields to use for similarity. NOTE: if possible, these should have a stored TermVector
mlt.mintf	Minimum Term Frequency - the frequency below which terms will be ignored in the source doc.
mlt.mindf	Minimum Document Frequency - the frequency at which words will be ignored which do not occur in at least this many docs.
mlt.minwl	Minimum word length below which words will be ignored.
mlt.maxwl	Maximum word length above which words will be ignored.
mlt.maxqt	Maximum number of query terms that will be included in any generated query.
mlt.maxntp	Maximum number of tokens to parse in each example doc field that is not stored with TermVector support.
mlt.boost	[true/false] Set if the query will be boosted by the interesting term relevance.
mlt.qf	Query fields and their boosts using the same format as that used in DisMaxQParserPlugin. These fields must also be specified in mlt.fl.

Table 9 Parameters and Descriptions for MoreLikeThis Handler [26]

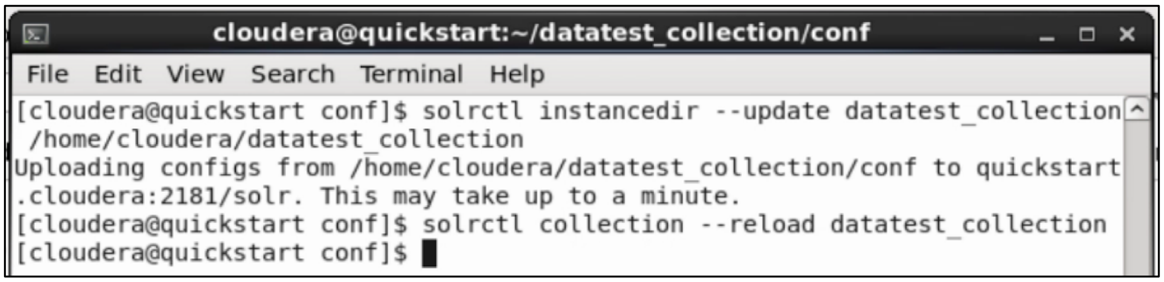
Parameter	Description
mlt.match.include	Should the response include the matched document? If false, the response will look exactly like a normal /select response
mlt.match.offset	By default, the MoreLikeThis query operates on the first result for 'q'
mlt.interestingTerms	One of: "list", "details", "none" -- this will show what "interesting" terms are used for the MoreLikeThis query. These are the top tf/idf terms. NOTE: if you select 'details', this shows you the term and boost used for each term. Unless mlt.boost=true, all terms will have boost=1.0

8.7.2 Update the instanceDir and reload the collection

After we change the configurations in schema.xml and solrconfig.xml, we need to do some operations on Solr to make the changed configurations take effect.

First we need to update the instance directory. Use the command below. In the command, we assume the instance name is datatest_collection and the instance path is /home/cloudera/datatest_collection. Then we can reload the collection to invoke the MLT.

Key Command(s)
solrctl instancedir --update datatest_collection /home/cloudera/datatest_collection solrctl collection --reload datatest_collection



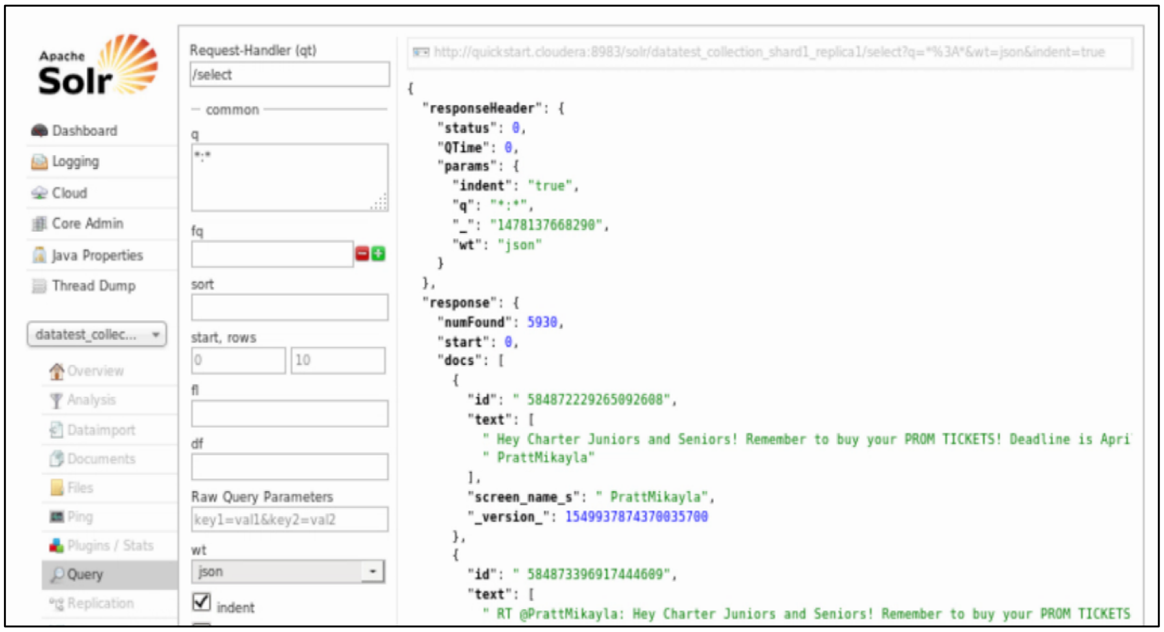
```
[cloudera@quickstart conf]$ solrctl instancedir --update datatest_collection
 /home/cloudera/datatest_collection
Uploading configs from /home/cloudera/datatest_collection/conf to quickstart
.cloudera:2181/solr. This may take up to a minute.
[cloudera@quickstart conf]$ solrctl collection --reload datatest_collection
[cloudera@quickstart conf]$ 
```

Figure 72 Reload the Collection for Recommendation

8.7.3 Check the results in Solr Admin UI

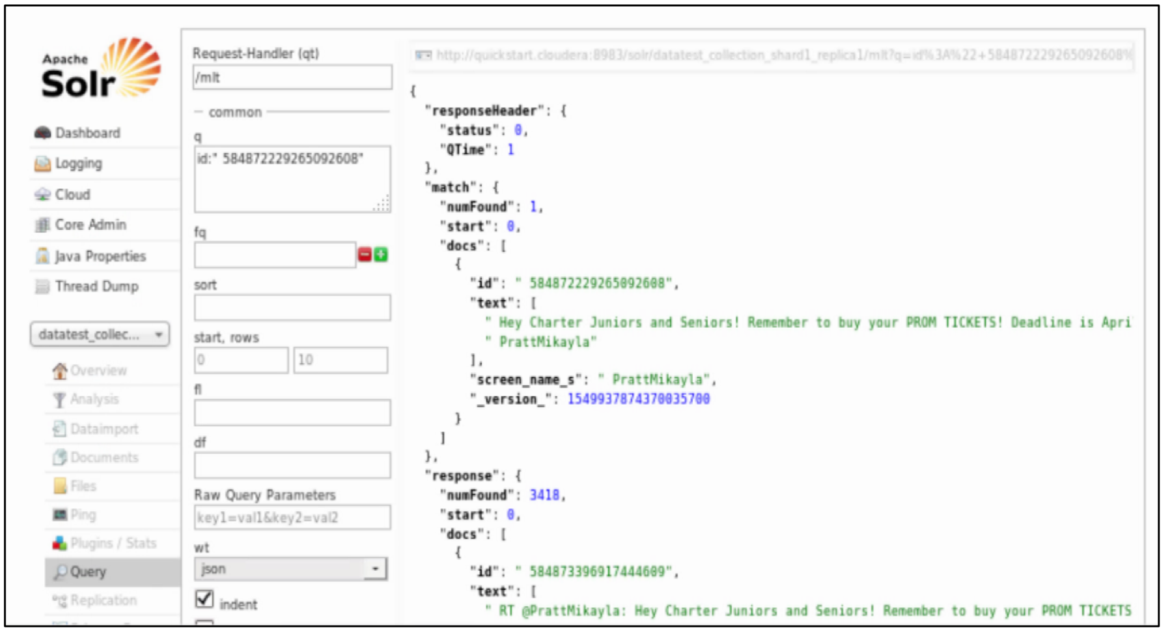
We can use the previous command to rebuild the index. Then, we can test the MoreLikeThis request handler in the Solr Admin UI.

First, we use the /select request handler to get the document list. From the response, we can know a document's ID. Then we use the first returned document to do the MoreLikeThis operation. We change the request handler from /select to /mlt. Then in the query box, we write the ID of the document for which we want to do recommendation. Next we execute the query and get the recommendation response.



Request-Handler (qt): /select
common
q: *:*
fq
sort
start, rows: 0 10
fl
df
Raw Query Parameters: key1=val1&key2=val2
wt: json
indent ✓

```
http://quickstart.cloudera:8983/solr/datatest_collection_shard1_replica1/select?q=*%3A*&wt=json&indent=true
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "indent": "true",
      "q": "*:*",
      "_": "1478137668290",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 5930,
    "start": 0,
    "docs": [
      {
        "id": " 584872229265092608",
        "text": [
          " Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is Apri
          " PrattMikayla"
        ],
        "screen_name_s": " PrattMikayla",
        "_version_": 1549937874370035700
      },
      {
        "id": " 584873396917444609",
        "text": [
          " RT @PrattMikayla: Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS
```



Request-Handler (qt): /mlt
common
q: id:" 584872229265092608"
fq
sort
start, rows: 0 10
fl
df
Raw Query Parameters: key1=val1&key2=val2
wt: json
indent ✓

```
http://quickstart.cloudera:8983/solr/datatest_collection_shard1_replica1/mlt?q=id%3A%22+584872229265092608%
{
  "responseHeader": {
    "status": 0,
    "QTime": 1
  },
  "match": {
    "numFound": 1,
    "start": 0,
    "docs": [
      {
        "id": " 584872229265092608",
        "text": [
          " Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS! Deadline is Apri
          " PrattMikayla"
        ],
        "screen_name_s": " PrattMikayla",
        "_version_": 1549937874370035700
      }
    ]
  },
  "response": {
    "numFound": 3418,
    "start": 0,
    "docs": [
      {
        "id": " 584873396917444609",
        "text": [
          " RT @PrattMikayla: Hey Charter Juniors and Seniors! Remember to buy your PROM TICKETS
```

Figure 73 An Example of Document Recommendation

8.8 Tutorial for Faceted Search

Faceted search [25], also called faceted navigation or faceted browsing, is a technique for accessing information organized according to a faceted field defined in Solr. Faceted search allows users to explore a collection of information by applying multiple filters.

Faceting is the arrangement of search results into categories based on indexed terms. Searchers are presented with the indexed terms, along with numerical counts of how many matching documents were found for each term. Facet queries only provide information (count of documents) and do not change the result documents. Faceting makes it easy for users to explore search results, narrowing in on exactly the results they are looking for. Table 10 presents some facet parameters and descriptions for the faceted search.

Table 10 Parameters and Descriptions for Faceted Search [25]

Parameter	Description
facet	If set to true, enables faceting.
facet.query	Specifies a Lucene query to generate a facet count.

Usually, we focus on two types of faceted search that are Field-Value Faceting and Range Faceting. For Field-Value Faceting, it is important to remember that "term" is a very specific concept. As shown in Table 11, several parameters can be used to trigger faceting based on the indexed terms in a field.

Table 11 Parameters and Descriptions for Field-Value Faceting [25]

Parameter	Description
facet.field	Identifies a field to be treated as a facet.
facet.prefix	Limits the terms used for faceting to those that begin with the specified prefix.
facet.contains	Limits the terms used for faceting to those that contain the specified substring.
facet.sort	Controls how faceted results are sorted.
facet.limit	Controls how many constraints should be returned for each facet.
facet.offset	Specifies an offset into the facet results at which to begin displaying facets.

facet.mincount	Specifies the minimum counts required for a facet field to be included in the response.
facet.missing	Controls whether Solr should compute a count of all matching results which have no value for the field, in addition to the term-based constraints of a facet field.
facet.method	Selects the algorithm or method Solr should use when faceting a field.
facet.exists	Caps facet counts by one. Available only for facet.method=enum as performance optimization.

Range Faceting can be used on any date field or any numeric field that supports range queries. This is particularly useful for stitching together a series of range queries for things like prices. Range Faceting is preferred over Date Faceting. Table 12 shows some parameters for Range Faceting.

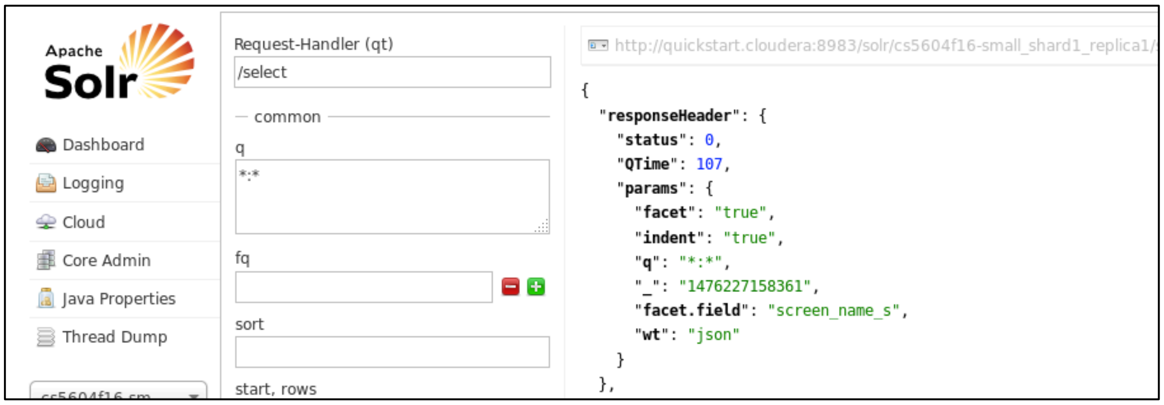
Table 12 Parameters and Descriptions for Range Faceting [25]

Parameter	Description
facet.range	Specifies the field to facet by range.
facet.range.start	Specifies the start of the facet range.
facet.range.end	Specifies the end of the facet range.
facet.range.gap	Specifies the span of the range as a value to be added to the lower bound.
facet.range.other	Specifies counts for Solr to compute in addition to the counts for each facet range constraint.

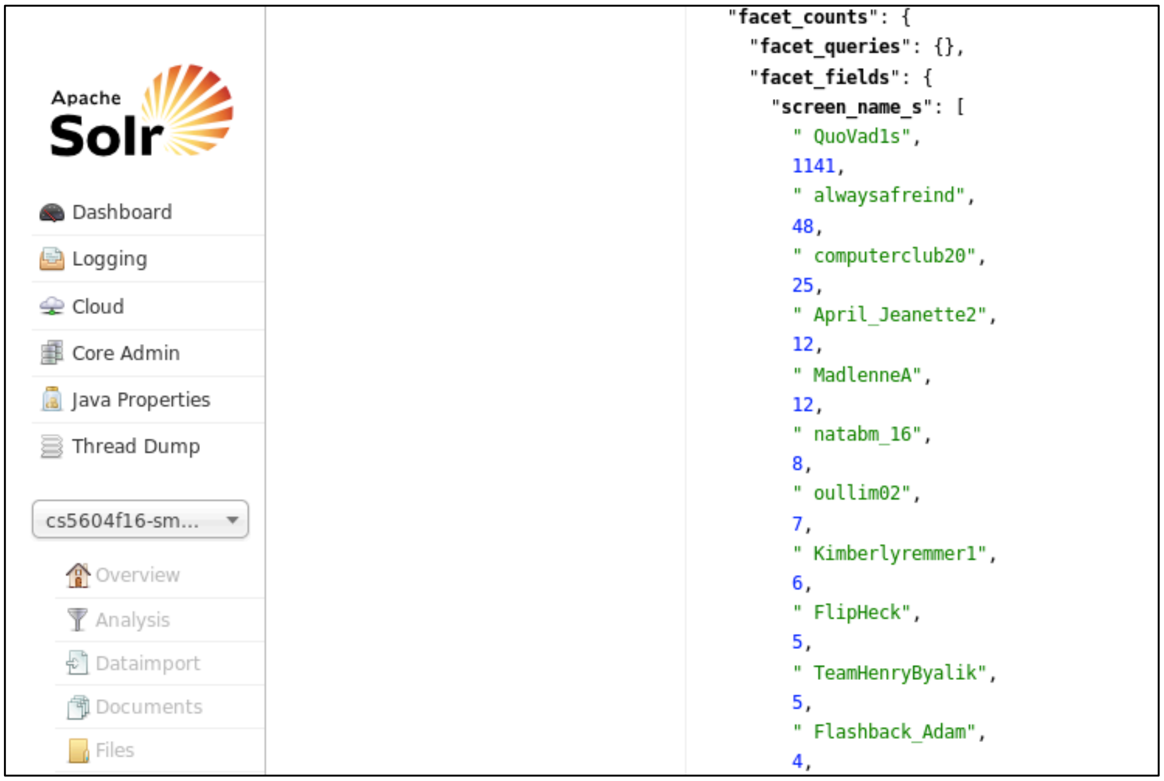
Based on our test file, we use the field ‘screen_name_s’ as the example to prove the feasibility of the faceted search. After the schema file is determined for our Hadoop Cluster, our faceted search will be able to support more functionalities. The following screenshot of schema.xml shows the modification of our tested schema file.

The following screenshot shows the query for a Field-Value Search. Especially, we set the facet as true to support the faceted search and the faceted field represents the field we are interested.

By default, the results are the sorted constraints by count (highest count first). The screenshot shows the result from the faceted search. From the results, we can find that the user whose screen name is QuoVadIs posted 1141 tweets.



```
/select
*:*
http://quickstart.cloudera:8983/solr/cs5604f16-small_shard1_replica1/
{
  "responseHeader": {
    "status": 0,
    "QTime": 107,
    "params": {
      "facet": "true",
      "indent": "true",
      "q": "*:*",
      "_": "1476227158361",
      "facet.field": "screen_name_s",
      "wt": "json"
    }
  },
```



```
"facet_counts": {
  "facet_queries": {},
  "facet_fields": {
    "screen_name_s": [
      " QuoVadIs",
      1141,
      " alwaysafreind",
      48,
      " computerclub20",
      25,
      " April_Jeanette2",
      12,
      " MadlenneA",
      12,
      " natabm_16",
      8,
      " oullim02",
      7,
      " Kimberlyremmer1",
      6,
      " FlipHeck",
      5,
      " TeamHenryByalik",
      5,
      " Flashback_Adam",
      4,
```

Figure 74 An Example of Faceted Search

8.9 Further Discussion

8.9.1 Search

Solr provides multiple flexible functionalities for developers to design their own request handlers and search components. You can find more details about request handler in Section 8.1 or the following link:

https://cwiki.apache.org/confluence/display/solr/RequestHandlers+and+SearchComponents+in+SolrConfig

Also, more explanations about these plugins could be found in solrconfig.xml and it is easy to modify the default setting to build a new plugin.

For the custom weighting, the default "select" request handler can be modified. And we can set different weights on different fields so that while calculating the tf-idf value, Solr will select the customized formula instead of the default one.

For the specific results, a new request handler could be established but based on the "select" handler. The default value for the retrieved results is a star (*), which means all stored fields will be shown in the result list. It can be replaced by some specific fields with our needs. An example has been presented in the custom ranking part. After using our custom scoring function, we only show the default score and new score through the Solr Admin UI.

By leveraging the request handler, we are also able to deal with the profanity issue in the results. One possible way is to create one column (e.g., profanity_tag) to tag whether the record contains profanity terms or not. Then, we can add “profanity_tag = false” to the default search field to achieve our goal.

8.9.2 Custom Ranking

At present, we created a simple custom scoring function by adding the importance value provided by the CMT team into the original tf-idf score. The CMT team took multiple features (e.g., followers_count, list_count, friends_count) into consideration and calculated the value. The formula is shown below with the red color.

$$\boldsymbol{Score = Doc_{score,Solr} + Doc_{importance}}$$
$$+ W_{topic} \times Doc_{score,topic} + W_{cluster} \times Doc_{score,cluster}$$

For the future work, our team need to extract the values from the topic and cluster fields, then add the topic score and cluster score into the current formula. After updating or optimizing the custom scoring function, we are able to build a topic-based or cluster-based ranking function.

Moreover, we know that the custom ranking is based on the Solr search components, which is used to process the search results. It is possible for us to extend these plugins to achieve some sub-goals during the query search process, like counting or filtering.

8.9.3 Document Recommendation

Recommendation can be of various types.
For the textual similarity based recommendation, it has been implemented as a basic version this semester. Conceptual similarity based recommendation and collaborative filtering have not been implemented this semester but can be implemented in the future. Conceptual similarity can be based on the probability of the document belonging to a cluster label that is provided by

the CTA team. Research needs to be done on using dynamic custom ranking to search for items that can be recommended based on the cluster labels they are tagged to. The probability can be the boosting factor to a label. Thus, all items with a high probability of a cluster label will be grouped as similar.

For the collaborative filtering, it involves studying the behavior of a user and making use of the previous history of the same user or many other users to predict the results. This can be implemented using the data collected by Blacklight which records the search query fired, the result set returned, and the click history of the user that determines and provides us information on the similarity between the results obtained and the user's information need. The log file generated can be read by a program that grabs information required to compute query-user-click history similarity and display results based on the information.

8.9.4 Solr

For our previous plan, a two-node Solr server should be deployed into our Hadoop Cluster. The reason is the Hue is mainly designed for data analysis, but turns to be weak in searching. More Solr nodes might speed up the searching process. With the help of the FE team, our Solr in the Hadoop Cluster could be connected with the Blacklight, which is a well-designed user interface and has been tested with a standalone Solr. So now while sending a search query, users will get the search results in a short time.

Unfortunately, there was not enough time for the FE team to test the Blacklight with the big dataset, we still need to evaluate the time cost of the efficient tool. For the 1.2 billion tweets collection, it takes about 8 seconds for Solr to make a response. Though the current time cost seems tolerable, it is easy to see that with the continual increase of data, we have to face the time issue soon. Therefore, to figure out SolrCloud or multiple Solr nodes in Cloudera Search should be planned and we need to deploy a multiple-node Solr server in the Hadoop Cluster.

Additionally, though we tried to make our current user and developer manuals more clear and readable, those documents could be improved with more details and more functionalities in the future.

9 References

[1] Wikipedia, https://en.wikipedia.org/wiki/Twitter, 2016
[2] Manning, Christopher D et al. Introduction to Information Retrieval. Cambridge University Press, 2009.
[3] Solr 6.2.0, https://lucene.apache.org/Solr/, 2016.
[4] Grainger, Trey, Timothy Potter, and Yonik Seeley. Solr in Action. Manning, 2014.
[5] McCandless, Michael, Erik Hatcher, and Otis Gospodnetic. Lucene in Action: Covers Apache Lucene 3.0. Manning Publications Co., 2010.
[6] Apache Solr Reference Guide Covering Apache Solr 6.2. https://www.apache.org/dyn/closer.cgi/lucene/Solr/ref-guide/apache-Solr-ref-guide-6.2.pdf, 2016.
[7] Integrating Solr, Solr Wiki, Apache Wiki. https://wiki.apache.org/Solr/IntegratingSolr, 2016.
[8] Apache Lucene, Lucene Core, http://lucene.apache.org/core/, 2016.
[9] Cloudera, Inc. Cloudera QuickStart. http://www.Cloudera.com/content/Cloudera/en/-documentation/core/latest/topics/quickstart.html, 2016.
[10] Hadoop Essential: The HBase Data Model. https://www.safaribooksonline.com/library/view/Hadoop-essentials/9781784396688/ch05s04.html, 2016.
[11] Big Data: An Introduction to HBase. http://www.stratapps.net/intro-HBase.php, 2016.
[12] Getting Started with HBase. http://HBase.apache.org/book.html#_get_started_with_HBase, 2016
[13] ZooKeeper 3.4 documentation. https://zookeeper.apache.org/doc/trunk/, 2016
[14] ZooKeeper service architecture. https://zookeeper.apache.org/doc/trunk/zookeeperOver.html, 2016
[15] Cloudera Documentation: Using the Lily HBase Batch Indexer for Indexing http://www.Cloudera.com/documentation/archive/search/1-3-0/Cloudera-Search-User-Guide/csug_HBase_batch_indexer.html, 2016.
[16] Hue UI with IDEAL and GETAR collections, Virginia Tech, http://Hadoop.dlib.vt.edu:8888/accounts/login/?next=/, 2016.
[17] Apache Lucene similarity scoring API, https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/Similarity.html, 2016.
[18] Solr recommender, https://github.com/pferrel/Solr-recommender, 2016.
[19] Apache HBase, http://zhangjunhd.github.io/2013/02/25/apache-hbase.html, 2016.
[20] Solr Interaction, https://cwiki.apache.org/confluence/display/solr/A+Quick+Overview, 2016
[21] Lily HBase Indexer with Solr, http://www.slideshare.net/romannikitchenko/hbasecrazydances, 2016.
[22] Edward A Fox, Kristine Hanna, Andrea L Kavanaugh, Steven D Sheetz, Donald J Shoemaker, III: Small: Integrated Digital Event Archiving and Library (IDEAL), NSF grant IIS - 1319578, 2013-2016.
[23] Edward A Fox, Donald Shoemaker, Chandan Reddy, Andrea Kavanaugh, III: Small: Collaborative Research: Global Event and Trend Archive Research (GETAR), NSF grant IIS - 1619028, 2017-2019.

[24] Data flow diagram in IDEAL and GETAR,
https://canvas.vt.edu/courses/28455/files/folder/F2016/Tutorials?preview=1566514, 2016.
[25] Solr Faceting, https://cwiki.apache.org/confluence/display/solr/Faceting, 2016.
[26] MoreLikeThis in Solr, https://cwiki.apache.org/confluence/display/solr/MoreLikeThis
[27] How MoreLikeThis works? http://cephas.net/blog/2008/03/30/how-morelikethis-works-in-lucene/
[28] Java Code Geeks: Solr Tutorial for beginners,
https://examples.javacodegeeks.com/enterprise-java/apache-solr/apache-solr-tutorial-beginners/, 2016
[29] RequestHandler and SearchComponents in Solr,
https://cwiki.apache.org/confluence/display/solr/RequestHandlers+and+SearchComponents+in+SolrConfig, 2016

Appendix A HBase Structure and Description

The table below shows the structure and description of our tweets and webpages collections. Our current table name is ideal-cs5604f16. The column families include tweet, clean-tweet, webpage, doc-type, tweet-topic and tweet-cluster. The "tweet" family contains multiple columns that are extracted from the raw tweets. The "clean-tweet" family is to show more rich information about tweets. The features of the webpages are stored in the "webpage" family. With the help of the CTA team, tweet clusters and topic are created and saved in the relevant column families. The "description" column presents the meaning of each column in each column family. More details are shown as examples.

HBase KEY_ID: Tweet: collection_id + tweet_id Webpage: uuid

Table	Column Family	Column	Description	Example
ideal-cs5604f16	tweet	collection-id	number of the collection	651
		collection-name	name of the collection	electricity
		tweet-id	tweet's unique identifier	299755872668758016
		archive-source	twitter API's type	twitter-search, twitter-stream
		source	platform's type	Android, iPhone
		text	tweet's original text	I can't believe it was a Virginia Tech student that posted that yik yak today. Just so disappointing 😔
		screen-name	tweeter's username	FiremanDave32
		user-id	user's unique identifier	385665827
		created-timestamp	created-time (UNIX time)	1428951621
		created-time	created-time (readable)	Mon Apr 13 19:00:21 +0000 2015

		language	tweet’s main language	en
		geo-type	point / polygon	point
		geo-0	latitude	43.02099179
		geo-1	longitude	-80.44612986
		url	original URL in tweet	<a href=” http://twittercounter.com”>The Visitor Widget</a>
		to-user-id	unique identifier of the reply-to user	0
		profile-img-url	image URL from the user profile	http://a0.twimg.com/profile_images/3149217853/0026816c03013356b569a8775af351fb_normal.jpeg
	clean-tweet	clean-text-solr	1. no porngraphic URLs, hashtags. 2. inappropriate plaintext, e.g. fuck, redacted as f***	[clean text for Solr and FE]
		clean-text-cla	1. no porngraphic hashtags 2. regular hashtags 3. inappropriate plaintext, e.g. fuck, redacted as f*** 4. all URLs removed 5. stop words removed 6. text lemmatized 7. remove # or @ symbol from mentions or hashtags	[clean text for CLA]
		clean-text-cta	1. no porngraphic hashtags 2. regular hashtags 3. inappropriate plaintext, e.g. fuck, redacted as f*** 4. all URLs removed 5. stop words removed 6. text lemmatized 7. remove @ symbol only from	[clean text for CTA]

			mentions, but keep #	
		rt	tag for the retweets	0 / 1
		geo-location	readable location from Google API	Blacksburg, Virginia
		created-year	year extracted from the created-time	2015
		created-month	month extracted from the created-time	11
		hashtags	tweet's hashtags	#hurricane
		mentions	tweet's mentions	@VT
		long-url	extended URL	http://www.roanoke.com/news/arrest-made-in-threatening-virginia-tech-yik-yak-post/article_4743fe59-023c-5b26-bebd-662594f7d6ca.html (from http://t.co/KEe6gpOMoT)
		classification-label	label of each tweet	hurricane
		real-world-events	list of the real world events	Hurricane Sandy; Hurricane Arthur
		sner-people	extract names from each tweet	Obama; Jimmy
		sner-organizations	extract organizations from each tweet	Virginia Tech
		sner-locations	extract locations from each tweet	New York; London
		tweet_importance	The importance value for each tweet	0-1
	webpage	collection-id	number of the collection	651
		collection-name	name of the collection	electricity
		html	raw HTML of webpage	[raw HTML text]

		tweet-ids	unique identifiers of the tweets that contains the URL of this webpage	593392960886145024
		language	webpage’s main language	en
		url	full url of the webpage	http://www.roanoke.com/news
		title	extract title from the webpage	Student arrested after threatening Virginia Tech Yik Yak post
		author/publisher	extract author from the webpage	Tom LoBianco and Pamela Brown, CNN
		created-time	extract created-time from the webpage	Mon Apr 13 19:00:21 +0000 2015
		clean-text		
		clean-text-profanity	clean text with no profanity	[clean HTML text]
		sub-urls	sub urls in the webpage	
		domain-name	extract the domain name from the webpage	http://www.fs.fed.us/
		domain-location	extract the country name from the webpage	us
		organization-name	extract the organization name from the webpage with the help of ©	Cable News Network
		fetched-timestamp	fetched time (readable)	Mon Apr 13 19:00:21 +0000 2015
		event	a list of events in the webpage	Hurricane Matthew; Flood
		classification-tag	identify whether the webpage has been previously classified or not	0 / 1
		webpage_importance	The importance value of each webpage	[0 - 1]
	doc-type	doc-type	type of the document	tweet / webpage
	tweet-topic	label-list	1. labels generated by LDA model 2. extract the top two labels from each	Signed,students; event,excited; today,register; april,thanks;

			topic	community,little
		probability-list	each value presents the probability of the tweet belongs to a certain topic	0.29112; 0.01820; 0.12435; 0.02572; 0.54058
	tweet-cluster	cluster-label	label of the tweet's cluster	NAACP stories
		doc-probability	the probability of the doc in the cluster	0.55167194