Analysis Me

**Spring**

2015

Technical Report

Team 5 - Spartans

Enclosed in this document is the technical report of the OpenNEX sponsored by NASA.

# Table of Contents

1. Introduction
2. Motivation
3. Related work
4. System design
5. System implementation
6. Experiments and analysis
7. Conclusions and future work
8. REferences

# Introduction

OpenNEX is a platform for users to discover, share and utilize vast amounts of data, software and workflows in order to foster scientific collaboration. Thru OpenNEX, users will be able to access years of accumulated sensor data; analytical models, tools and software, provided by NASA, as well as contribute their own back to the community. With this model in mind, we expect accessible resources to grow significantly over time and prepare the system to scale alongside it.

In this report, we lightly touch upon approaches to Access Control for OpenNEX and how various services offered will conform to this model. A preliminary design is offered in Section 4. We also discuss in depth the necessity of Search and Recommendation to systems such as OpenNEX and how we can scale the functionality given the system’s expected growth. We will tackle several similar projects to OpenNEX and how they utilize Search and Recommendation as an initial point of study. We then define specifications for the feature and recommend an initial design towards its implementation.

In addition to the design, we have implemented a prototype demonstrating the functionality and viability of the recommendation on scientific workflows. The prototype is composed of a web service API written using the Java Play! Framework, backed by a MySQL database and a deployment of Apache Sqoop and Apache Hadoop for the data processing. Though the prototype is not as comprehensive as the design, it shows the workflow that will form the basis for building searchable and recommendable content.

# Motivation

With data sets and resources growing ever larger, Search is a common feature in every system nowadays. Enabling the users to quickly find articles, elements or even other users empowers them to be more productive for the task at hand. In the context of OpenNEX, users need to be able to find relevant data and resources to their experiment. With NASA opening its vast resources to the public, there must be a way to sort thru all these resources and focus on the most relevant ones.

Recommendations on the other hand are additional suggestions that the system can provide in order to further the goals of the user. By building content and contextual relationships, we can support user activities by recommending them the appropriate pieces of content relevant to their task. Plain search requires that the user has a pre-notion of what they are looking for. In order to perform queries, they would have some keyword or filter criteria in mind. However, not all resources are tagged and indexed the same way as a user might think. Recommendations can help surface these resources that in a way can be more relevant to the user than their pre-notion, based on previous user activity and the activities of users before him.

There are several challenges to building a Search and Recommendation engine on OpenNEX. First, is the sheer size of the expected resources being disclosed by NASA. In order to handle resources of this size, we would have to look into approaches to big data and distributed computing. Second, is the constant growth and evolution of resources. We expect the users to add their own data and tools to grow the system. Apart from resources, we also gain more insight based on user activity. These constant new data points means that the Search and Recommendation engine would need to process and adapt incrementally, employing a feedback loop taking into account both old and new data. Third, is generating this system from a cold start. Since this is a new system and there is no user analytics or data tagging done, we would have to build this system incrementally and step by step as well.

# Related Work

There are several existing services that offer social collaboration for scientific workflow. Although these options are not as vast as OpenNEX, we can model after them and improve upon their current offerings.

## crowdLabs.org [1]

CrowdLabs is a collaboration website where users can upload VisTrails workflows, software packages and datasets. It also enables organization by groups and projects. Each resource uploaded can be rated, discussed and reviewed by other users.

In terms of search, it offers two types of search methods. First is a simple textbox for search queries run against a specific resource type. Second is enabled by a set of filter criteria pre-determined by the system.

****

Figure : Simple Text Search

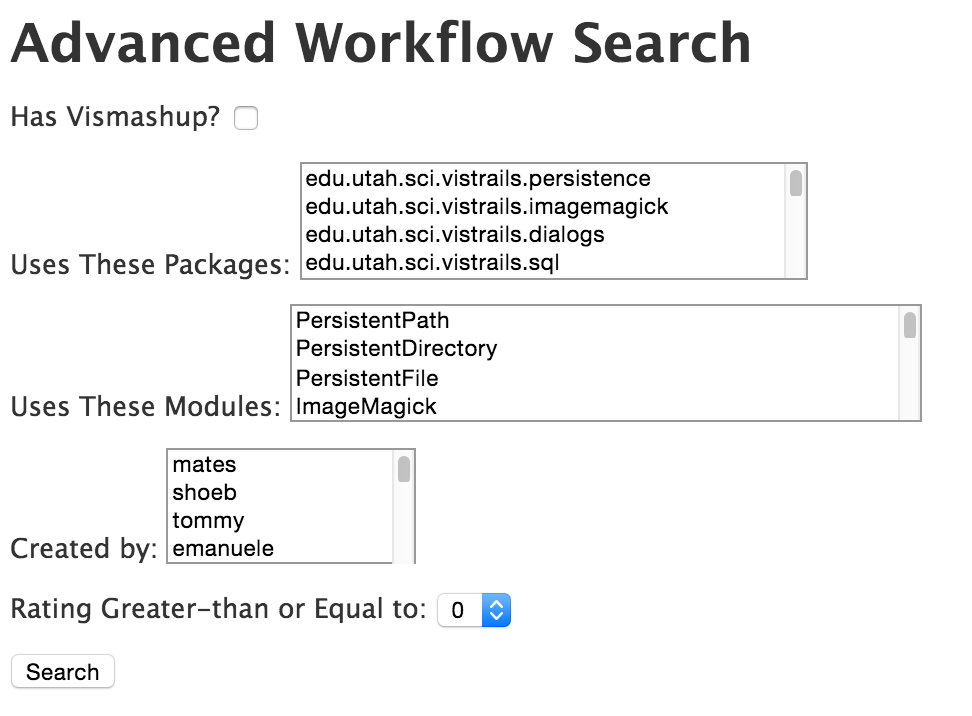
****

Figure : Advanced Search Filters

At the very minimum, OpenNEX should support a text based search that enables surfacing of user relevant content and not rely on keyword matching alone. Queries may also be made on larger resource registries and of a freeform nature. The filtered search offered by crowdLabs relies on specific metadata defined for each resource. Unfortunately, OpenNEX resources may not be as structured as resources uploaded to crowdLabs to generate similar metadata, however the option of a more guided and structured search is definitely a positive target for the system.

## myExperiment.org [2]

MyExperiment is a similar service to crowdLabs, though geared more towards Taverna workflows. Its holds more content that crowdLabs and has more recent activity. Shareable resources are limited to workflows, files and packs, while organization is limited to users and groups. They’re search functionalities however are far more flexible than crowdLabs.

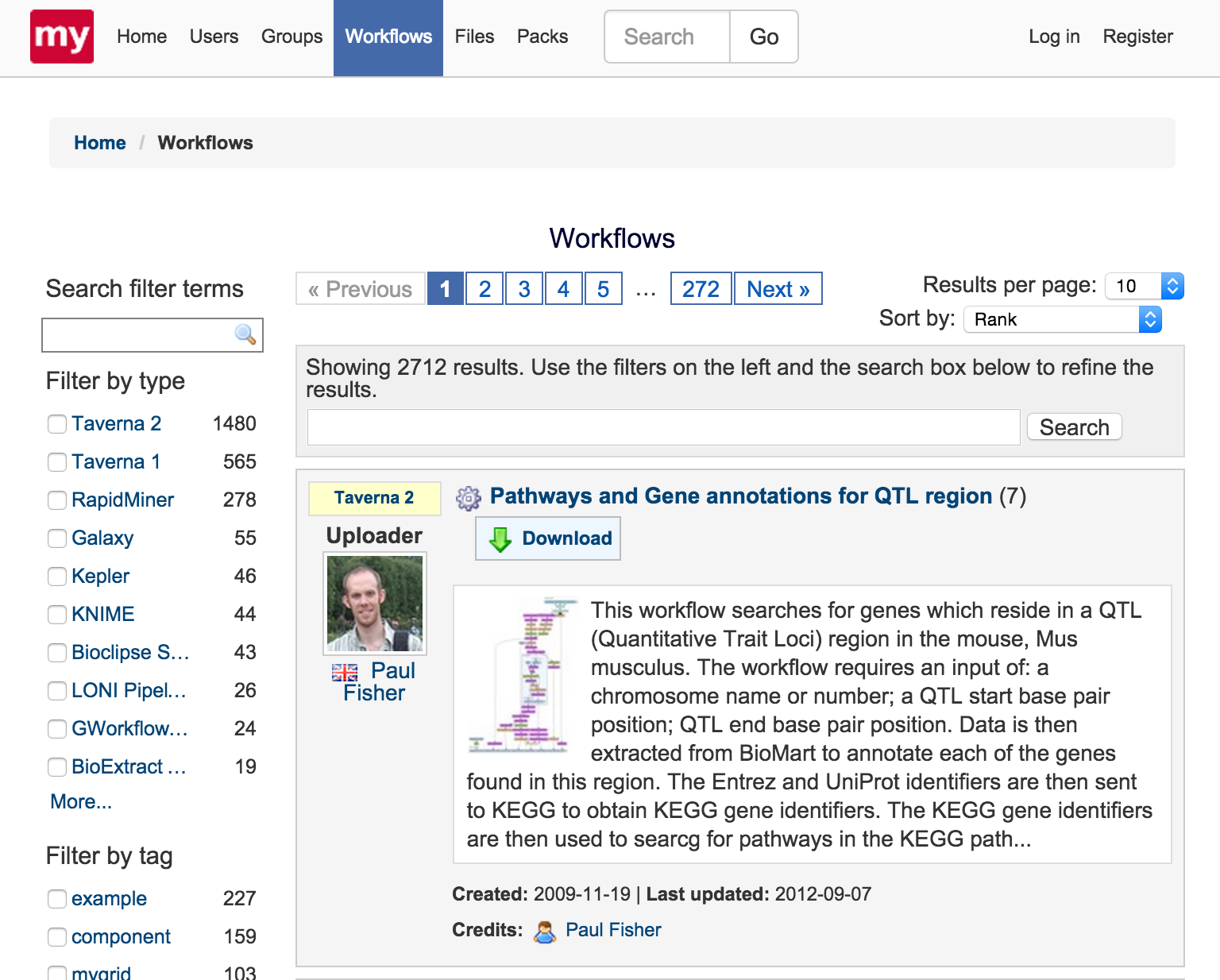
****

Figure : myExperiment Search Options

Searching in myExperiment can happen in two different contexts. The first is the searching from the main nav bar textbox, this allows searching thru all of myExperiments resources, users and organizations. The second context is content specific where it combines the search term textbox with several different filter suggestions. These filter suggestions are built dynamically and offer more insight as to how relevant each of them are. One drawback here is that the suggested terms can be too generic to the system that it offers much less insight that its current given weight (e.g. type = Taverna 2, tag = example).

While both crowdLabs and myExperiment offer great references for Search, they do not imply anything on Recommendations. In order to build Recommendations, we need to consider improving search relevance through relational models and factoring in user activity aside from resource metadata. Recommender systems typically employ heavy analytical algorithms to correlate resources and users [3]. However, in the scope of this report, we focus mainly on extracting relevant data from given resources and how to setup a system to handle this type of data flow.

# System design

## Access Control

### Assumption

The original OpenNEX project has basic access control service components like in user login and authentication parts, yet our design for access control mainly focuses on interaction between users of different groups and resources from different sources. In the OpenNEX that we try to design in this whole project, the users can be grouped according to different interests as well as levels of recognition, and the resources will have different accessibility level. Access control here aims at finding a way to grant users with different privilege of access, and verify the user privilege and resource accessibility.

### Functional Requirements

The access control service in this system should have 3 basic functionalities.

The first one is privilege granting mechanism. This at one hand includes how to measure credits of specific user, like users will earn credits as they use the system often, as their account are active enough, as the age of the accounts increases, or as they make significant or extra contribution to the community. The other side of this mechanism is to define criteria for granting privilege depending on credits. What privilege of access can one have if it has this many credits? It is obvious that more credits lead to higher access level, but in the real system the relationship doesn’t go in a linear way. There should be practical, automatic method to granting users with privilege they deserve.

The second one is verification for users and resources. This is to see if the privilege of users and the accessibility of resources can match when a specific user is asking for the access to certain resource or service. This can be done by looking up the ACLs (Access Control Lists).

The third one is to enable collaboration. Since the service we design may also want to help the crowdsourcing process, the information contained in this service will be useful for recommending collaborators. For instance if two users has similar privilege, the possibility that they may access the same resource is relatively high, which indicates there is a potential that they could work together on the same projects, or exchange ideas on related fields. Also, access control is a result of group management, and in turn an important tool for it.

### Access Control Design

In this part we present our design for the functionalities above that we’d like to achieve.

Figure 4 shows the process of verification for users and resources. When the user makes a request to access specific resource or service, the request would firstly go to Access Control Manager (ACM), the ACM then checks both of user’s privilege as well as the accessibility of the resource, and decide if this request could be fulfilled. If two parts can be matched, them ACM is also responsible for giving permission to user, and then user can go for what he wants.

## 

Figure : Access Verification

Another part is showing how resources are shared in the manner of group or community. The group is made up by users in a hierarchical level, and users may have different access rights related to their roles in the group. Like the system administrator may have highest access privilege in system level. And the team leader can manage the resources and users accessibility within team. And the ordinary users have basic access rights. The group also keeps what we called the Access Control Lists, which specifies what roles have what kind of access rights to what resource.

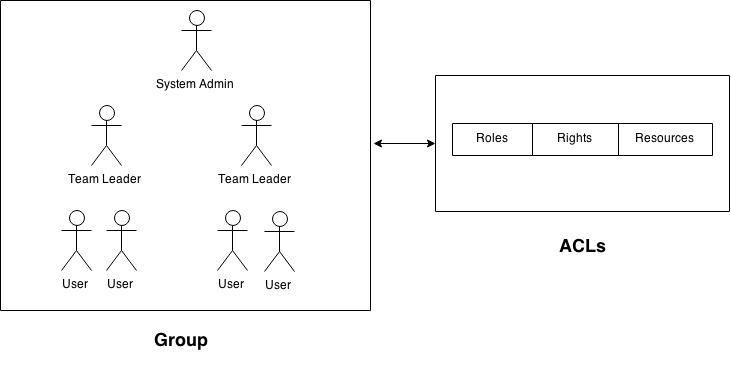


Figure : Hierarchical Access Rights

## Search and Recommendation

### Assumptions

In this iteration, we work with several assumptions on existing designs and parallel developments of the entire system. We will focus mainly on the Workflow data model and building an API to allow text based searching from a relatively large dataset.

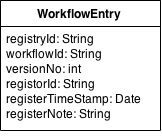
The figure aside shows the Workflow data model currently being used by NEX. Our assumption is that workflowId is the unique ID to identify a workflow and registerNote is an arbitrary text field to describe a workflow. Given this data model, we do not have much information to create indexable metadata without having to do a full text search thru registerNote. Our objective is to then extract valuable metadata from this model and construct a model which we can further breakdown and analyze for the purposes of search and recommendation.

Figure : Workflow Data Model

An additional assumption is that the service for creating and referencing these entries already exists. Therefore, we will focus on massaging metadata given a placeholder dataset instead of constructing a separate service for it.

### Functional Requirements

For a simple text based search, we need to be able to take an arbitrary search string and cross reference the search string with various workflow entry descriptions. With this criterion, we construct an API to for users to submit query text and return a list of relevant workflows containing that text.

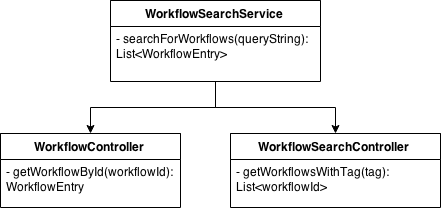


Figure : Workflow Search Service Components

For this exercise, we will handle that requirement using a WorkflowSearchService that takes an HTTP GET request containing the search query as parameters, and return a JSON array of matching workflows.

To componentize this further, we delegate responsibilities to a WorkflowController, which serves as the access point to the workflow entry database and assumed to be existing, and a WorkflowSearchController, which serves as the functional unit that correlates a tokenized search query with relevant workflowIds. The results are then combined together by the service and returned the workflow entries as results of the query.

### System Design

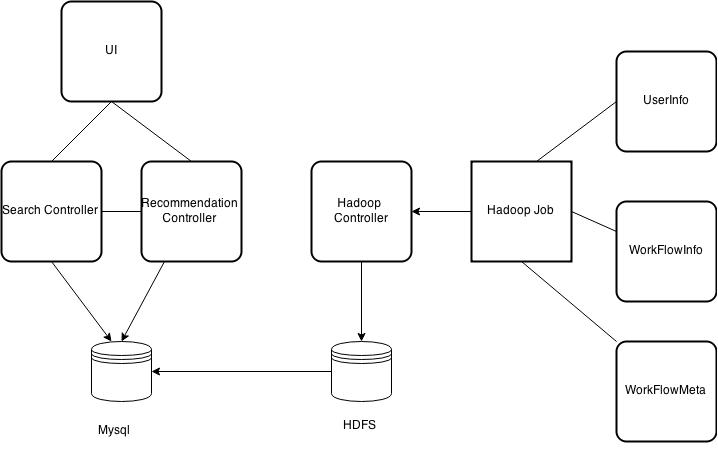


Figure : System Design

The design for backend system is to gather information for search and recommendation and do data processing to extract, transform, and load data to provide the frontend with data that are meaningful and can easily be used. The backend work consists of three parts.

#### Data Transfer

The data we are dealing with mainly come from 2 parts. One is the workflow entry table that we assume as already existed in OpenNEX when the workflows are being created. The other is the log files that are being generated constantly as users producing activities in the system. Needless to say, these two source of data are both massive and complicated. So the transfer of such big data flow matters significantly in the performance of the system. In our prototype, this mainly occurs when we have to transfer bulk data between Apache Hadoop and structured data stores such as relational databases (here we use MySQL) in the manner of real-time operations. In this case, we used Sqoop to help with the transferring of big data. It can transfer the table in MySQL directly to file in HDFS. Then we can straightly deal with HDFS in MapReduce work instead of connecting and doing querying to MySQL in mapreduce program, because the latter one can affect the performance of our system. Adopting Sqoop avoid from messing data up in MySQL for frontend and HDFS for backend. In this way, the data processing could be done in HDFS using MapReduce, and the search and recommending could be done in Play! framework querying MySQL. This design method makes it really clear and thus improves performance significantly.

#### Hadoop MapReduce

Our search engine is keyword-based, so we need to collect information about tags for user to search. The tags we come up with come from the registerNote field of workflow entry table. Assuming the registerNote are created along with the workflow, and they are a few sentences describing what the workflow is about. So the tags could be the keywords in those sentences. One problem here becomes how to extract keywords. A way to do that is to have pre-defined keywords that we use as tags in our system. Another way is to filter out the words we don’t want (Stopwords Method) instead of defining the words we want. We use MapReduce for this work. The input of MapReduce work is the file in HDFS that is already transferred from MySQL, the output file is the (tag, workflow-id list) key-value pair. Then we transfer this file back to workflow meta table in MySQL.

Our recommendation engine is count-based, so we need to gather information about usage counts and corresponding workflow. These information can be extracted from the usage logs generated along with the creation of workflow. So the usage count part is to analyze the usage log and calculate the count for each workflow. We also use MapReduce work to calculate the counts for our workflow. Likewise, we use Sqoop to transform the file we come up with in HDFS back to the same workflow meta table that we created in MySQL.

#### Real-time Processing

In the design of a search and recommendation system, one thing that must be taken into consideration is how to make the information real-time and dynamic so as to reflect users’ true preference and tendency for these workflows.

Since the system is built upon Unix-like operating systems, this can be done by using the built-in job schedule tool Cron. Then again we use Sqoop to aggregate the results in HDFS, and update the database every period.

# System implementation

## Our prototype

Our prototype contains mainly two parts, one part is the Hadoop job for the backend, and the other part is using Play! framework for actual search and recommend.

As explained in system architecture, we use the knowledge of workflow information in the workflow entry table and usage information in the log files to enable user search service and make recommendations. The first part in this design is to use Hadoop MapReduce work to ETL job. Extract useful information from the resources above, transform them into what we need and into formatted structure, and load them into database. After we transferred the data from HDFS into Mysql, we can now search and recommend data directly from Mysql. We define two controllers, one is search controller, the other one is recommendation controller. Search controller can be used to search workflow by tag, by name, by id, etc. Recommendation controller can be used to recommend workflow by usage counts. We assume that the higher usage counts, the better the workflow. RESTful APIs are exposed for user to use our application. A simple GUI is provided for users to test the application.

We use Play! framework, Hadoop, MySQL to implement our design.

For the MySQL database part, we have created two tables: one for workflow entries and one for workflow metadata. We referred to the data model that NEX uses for workflows. This will make it easier to integrate our prototype to OpenNEX. Granted, we could feed all information related with the workflow into one table. In that case, the data model of the table would have such fields:

String registryId

String workflowId

String versionNo

String registorId

String registerTimeStamp

String registerNote

String tags

int count

This is totally legal and valid but it might hurt performance and scalability of the site when it actually launches. Why we separate the information related to a workflow into two tables is mainly for performance concerns.

Firstly, separating workflow information into two different tables would make the database operations faster. Database operations is one big bottleneck for the whole system. This is mainly because the data we are dealing with is very huge. The data mainly comes from two sources: one is from NASA itself. Since this is a NASA supported platform and NASA will a large set of existing workflows and softwares to make it possible for other people to leverage existing work. The other source of data is from the public, since everyone can upload his own workflow at any time from anywhere in the world. Thus as time goes by there will also be a considerable amount of workflows contributed by the users. After all, this is the very reason NASA has created this collaborative platform: to form a good community where people can make their own contributions and also avoid reinventing the wheel. Making queries from such a big database will cause a big latency. And the dataset itself is ever growing as the platform gain popularity. To make it more challenging, queries could come from users from anywhere in the world at any time. It is definitely possible that the site could experience some sort of peak hours or days where the queries come in very high concurrency. If we deal with very high concurrent queries while interacting with such a big database, the site would become extremely slow, possibly causing it to crash.

By examining the columns of the table, we found that some of the fields are not frequently used in main logic of search and recommendation. When we search for and prioritise certain workflows, what we mainly care about is the tags and count of the workflow. We don’t really need all the metadata like version number and timestamp to finish the search or filtering. So we could put the data necessary for our search and recommendation feature into one table, and other metadata of the workflow into another table and only expose it when a specific workflow is requested. In this case, we could reduce the size of the table we are querying from, thus improving the throughput of the site.

Besides, separating workflow information into two different tables also facilitates updating the database and keep all data up to date. Once the workflow is updated, some metadata are relatively stable and rarely changed. While some other fields like the usage count of the workflow is always changing and should be constantly updated. The same goes field, where the user can append for the tags new tags to the workflow so it could be dynamically changed. Updating a smaller table would be faster than a big and comprehensive table.

Taking these factors into consideration, we have created tables as such:

The workflow entry table has such information:

String registryId

String workflowId

String versionNo

String registorId

String registerTimeStamp

String registerNote

The workflow meta table has such information: workflow\_id, tags, count.

String workflowId

String tags

int count

Workflow\_id serves as an unique identifier for each workflow. We map entries of the two table by workfow\_id.

As explained above, the first job for us is to generate information for workflow meta table from the initial workflow entry table we have and the log files generated while the system is being used by users.

## Workflow Information to HDFS

To deal with this big dataset of workflow information, we should use Hadoop MapReduce job to extract, transform and load data. Since the workflow information is in the big workflow entry table, and that constantly dealing with databases in MapReduce work is really time-consuming and inefficient, we should first of all get the data out of database. After some research and comparison. We decided to use Sqoop to transform the data from MYSQL to HDFS. In this way we now have a large file in HDFS in which each line represent each row in MYSQL.

## Tags Extraction, Transform, and Load

Since our search engine is keyword-based, we need to collect information about tags for user to search.

The tags we come up with come from the registerNote field of workflow entry table. Assuming the registerNote are created along with the workflow, and they are a few sentences describing what the workflow is about. So the tags could be the keywords in those sentences. One problem here becomes how to extract keywords. One way to do that is to have pre-defined keywords that we use as tags in our system. Here is a bunch of keywords that we use in one of our initial prototype (mostly are terms in the fields of geography and earth science related subjects).

*“animals”, “atmosphere”, “earthquakes”, “ecology”, “geography”, “geophysics”, “ice”, “soil”, “tectonics”, “temperature”, “volcanoes”, “water”, “weather”*

Restricted by the knowledge of the designers, the re-defined tags may not be so complete and scientific enough. So later on we come up with an idea of filter out the words we don’t want instead of defining the words we want. Inspired by this idea, we used the stop words that are mostly used in NLP (natural language processing) to filter out the commonly used words in daily life and keep the other words that are less used (mostly scientific terminology).

For all the workflows, we use Hadoop MapReduce job to extract the tags from registerNote. In the map job, for each workflow we get the registerNote and split it into different words, then write them into (workflow-id, tag) key-value pair. In the Reduce work, we filter the values (tags) to keep only the keywords we need, and invert the key-value pair to append the workflow-id for each tag, and write them out to HDFS. At last we use Sqoop again to transform the file in the HDFS back to workflow meta table in MySQL.

## Usage Count Analysis

Since our recommendation engine is count-based, we need to gather information about usage counts and corresponding workflow.

Unlike tags, which are generated along with the creation of workflow, the usage information is noted down constantly in logs. So the usage count part is to analyze the usage log and calculate the count for each workflow. Assuming our usage log is a bunch of real-time time-stamped logs, we also use MapReduce work to calculate the counts for our workflow. In the map job, for each line in the log, we get the workflow-id that is used at that time, and set the corresponding count to 1. In the reduce job, we sum up the counts for each workflow and write them out to HDFS. Likewise, we use Sqoop to transform the file we come up with in HDFS back to the same workflow meta table that we created in MySQL.

## Real-time Analytics – Cron and Sqoop

Another challenge in recommendation is to make the information real-time and dynamic. Since the usage logs are constantly generated from users’ activities, it is important for the recommendation to be real-time enough to reflect users’ true preference and tendency for these workflows.

Since the system is built upon Unix-like operating systems, this can be done by using the built-in job schedule tool Cron. We configure using Cron to run Hadoop MapReduce work every minute automatically over the new log files generated during the past 1 minute, and then use Sqoop to aggregate the results in HDFS, and update the database every period.

## Web Service

Play! framework adopts MVC design pattern, which makes the whole application easy to maintain and reuse. We created four controllers under controller folder, which are RecommendController, SearchController, WorkflowController, WorkflowMetaController. RecommendController is responsible for recommending suitable workflows, currently; we recommend top ten usage count workflow for a specific tag. SearchController can search workflow by tags. WorkflowController can be used to search a workflow by title. WorkflowMetaController can be used to search a workflow meta by id.

### Retrieve all workflows

One use case is simply getting all workflows from the system. We configured the routes file in the conf folder, so that Play! framework will help us do the routing and call the corresponding handlers. When the user makes a get request from the client side to the “/workflows” endpoint, the Play! framework will call the getWorkflows() method of the WorkflowController class. The controller will utilize the data model class to get information from the workflow entry table.

### Search workflow

Another use case is to search for certain workflows by tag. The user makes a get request to the endpoint “/workflows/search/:tag”, where the tag is the specific tag the user is interested, like “weather”, then the getWorkflowsByTag(String tag) method of the SearchController class will be called. Here we query the workflow meta database for all ids of the workflows that has the tag the user is requesting. Here we could return the list of ids, and if the user is interested in a specific workflow we return, he can query for the complete information of this specific workflow. When querying by tag, we order the result by count and only return top 10 workflow ids for the endpoint “/workflows/tag/:tag”. The “/workflows/search/:tag” endpoint will return a complete list of all workflows and the complete information of that workflow. The count information represents the historical usage of this workflow. The more times the workflow has been used, the higher the chance it would be more reliable. So we recommend the user the top workflows of the field they search for based on the usage history.

# Experiments and analysis

In this part we will present the functionality and system output of our design.

These are the data schemas that we used in MySQL.

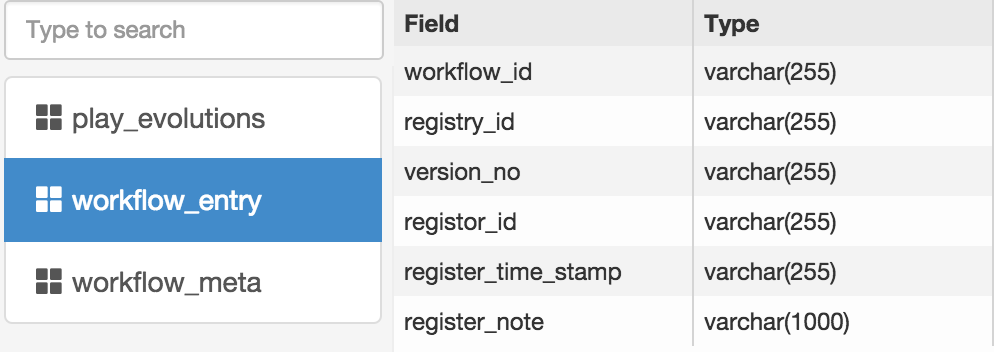


Figure : Workflow Entry Table

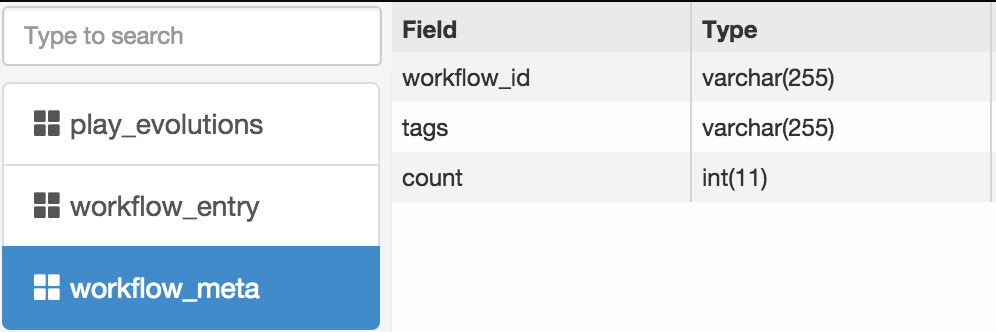


Figure : Workflow Meta Table

This is the file in HDFS generated from­ workflow entry table using Sqoop.

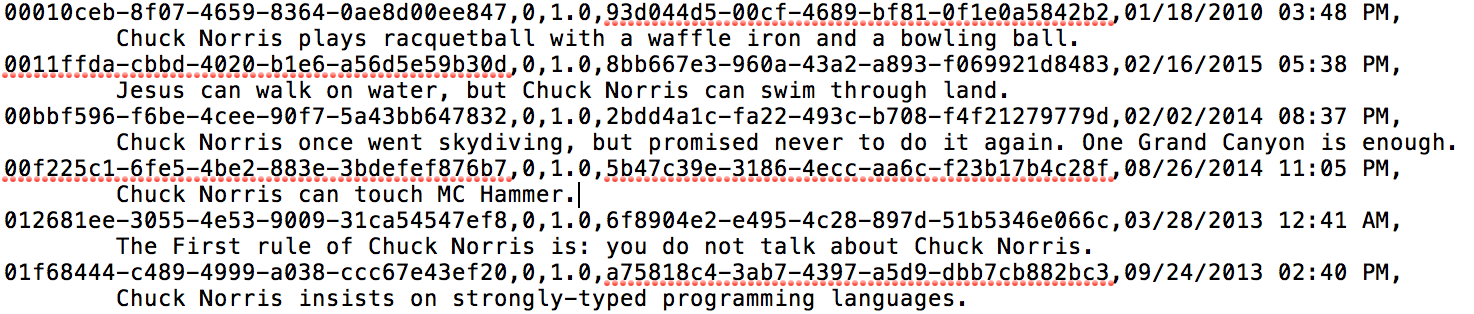


Figure : HDFS Input File

This is the dummy file of usage logs.



Figure : Usage Log File

These are the results that are located in HDFS.

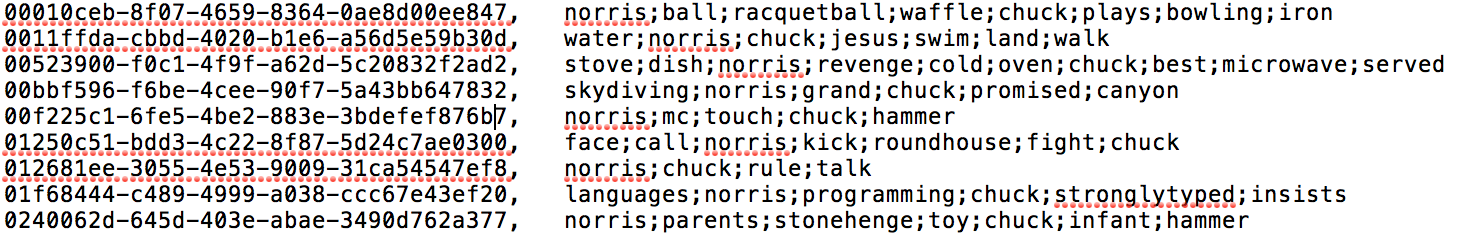


Figure : Hadoop Output in HDFS

This is the results that are transferred back to MySQL using Sqoop.

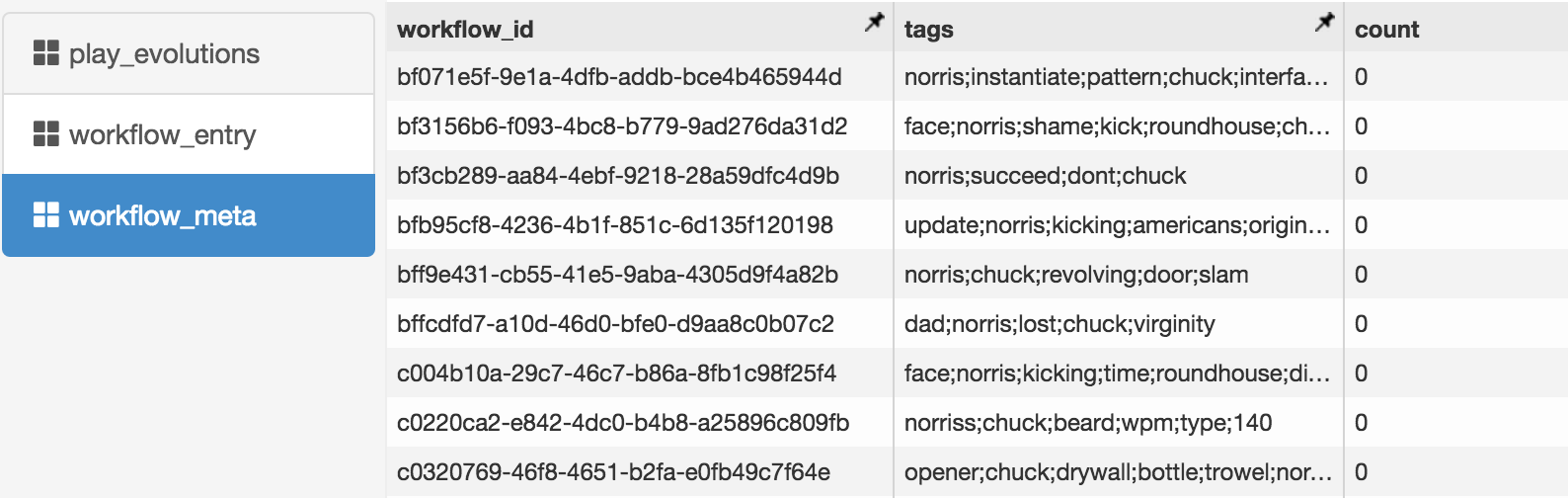


Figure : Hadoop Output Transferred to MySQL

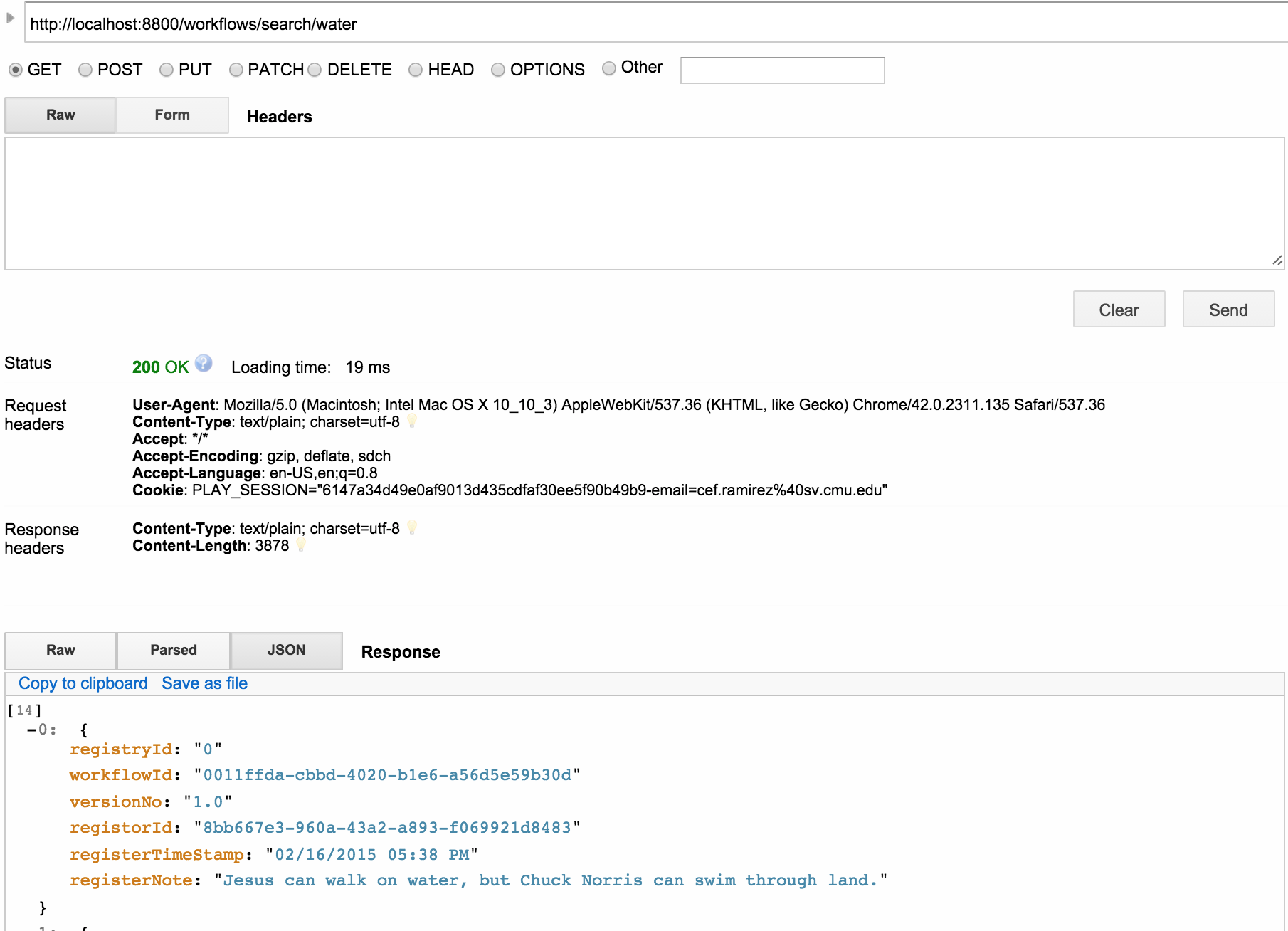
This is to show the search functionality in Play! framework.

Figure : Querying the Search API

# Conclusion

In this project, we mainly focus on design and implementation for search and recommendation service for OpenNEX, containing using Play! Framework for frontend that deals with the search queries and recommendations, as well as using Hadoop MapReduce, MySQL and Scoop for the backend to extract, transform and load the useful information in workflow entry table and system log files into workflow metadata table. We also developed a prototype to show how our design and implementation can work out as predicted, and how they can be reused and enhanced by later design.

Since there are many restrictions and security access issues in NASA, we mainly focus on implementing the functionalities we mostly care about in this project, that is the design of search and recommendation as a prototype. Yet there are many aspects in the system that we could dive deep into.

For search part

* Adopt more Natural Language Processing (NLP) knowledge to better capturing tags for workflows
* Change the basic search method from keyword based to like syntax based or semantics based
* Provide more searching options rather than tags. Like search for workflows used by specific user or group.

For recommendation part

* Besides from count-based recommendation, we can have relationship based workflow recommendation, which means providing recommendations in more general ways. The users needn’t type specific tags he wants, we could recommend workflows based on user profile, on the age of the accounts, on the credentials, qualifications and records of the user.
* In another scenario, if we find out the user has used or searched for some workflow, we could recommend workflows that coherent to the previous one. Also, depending on the coherence of related scientific subjects or communities, the users could be recommended to other workflows that other scientists or groups are using.
* Enable the co-recommendation between users. They could provide recommendation for each other on purpose.

# References

(1). Retrieved May 6, 2015, from crowdLabs.org: http://www.crowdlabs.org

(2). Retrieved May 6, 2015, from myExperiment: http://www.myexperiment.org

(3). Recommender Systems (Machine Learning Summer School 2014 @ CMU). Retrieved May 6, 2015, from http://www.slideshare.net/xamat/recommender -systems-machine-learning-summer-school-2014-cmu