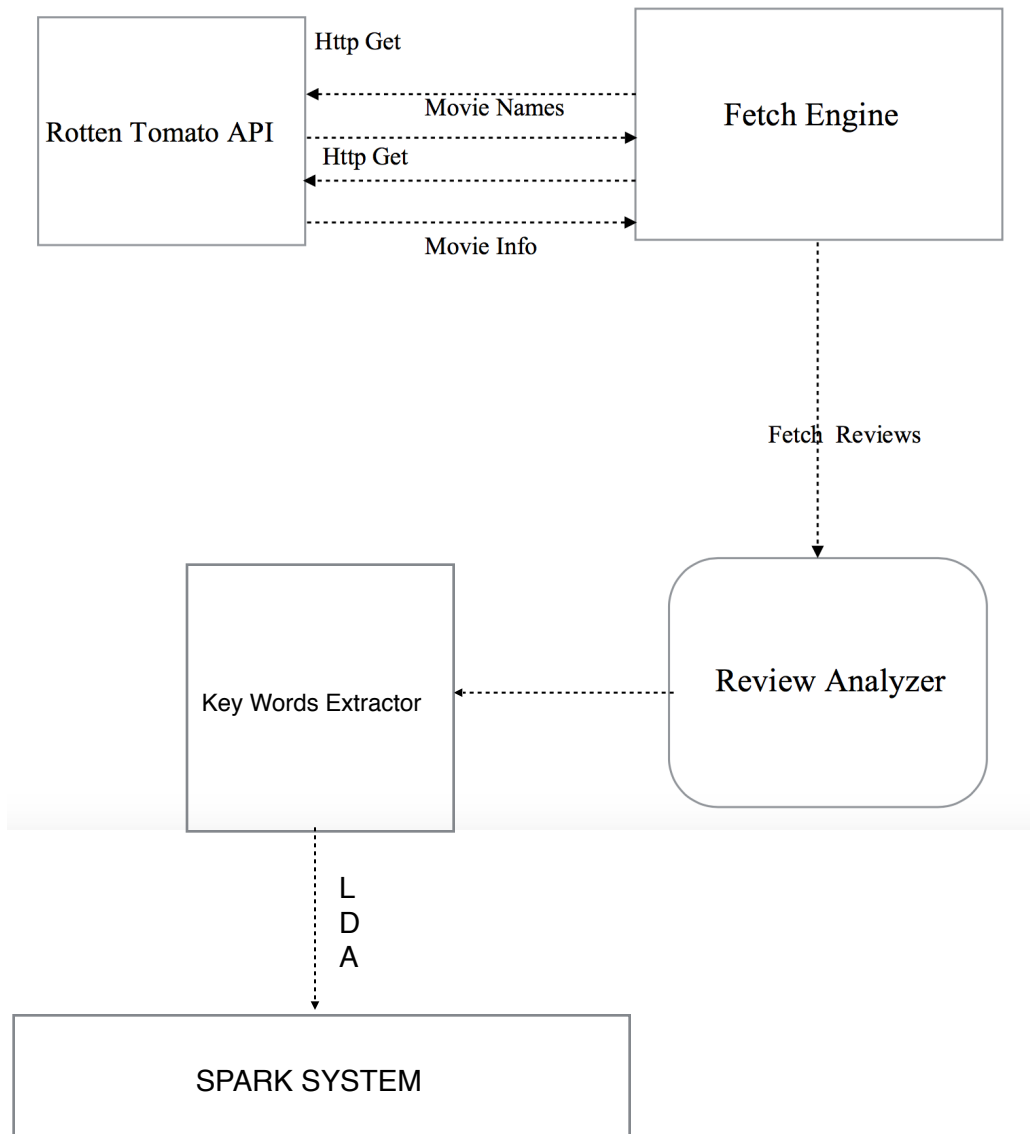


# CS560 KDM Project Increment 2 By

**Cuong Cu**  
**Rishabh Bhojak**  
**Vishnu Chelle**  
**Ran Chen**

- **System Architecture Diagram**



This is the general architecture of our application for Phase 2. The core elements are, Fetch Engine, Rotten Tomato & Analyzer.

1. Fetch Engine: Fetch engine is the program which makes the Http get request to get movie names from Rotten Tomato using their API, Once the fetch engine gets the movie names, using their movie ID it makes another Http request to get info of that movie. That's how we are getting our data.
2. Review Analyzer: This is our main Component, It fetches the data from fetch engine and process the data(movie reviews) to analyze it. It mainly consist of the following sub-components,
  1. Text Extractor: Once the review is fetched, text extractor runs on top of the review, to divide the sentences. Sentence with before any full stop(dot) is considered as a single sentence.
  2. NLP, Runner: The NLP, Runner looks for the key words after parses each sentence indepen-

dently. We have been able to achieve the negation sentences and reference words.

3. Coreference: We are using Stanford Coreference method for our application, which makes it smart to understand words like “it”, “they”, etc. This helps to understand the long reviews.
4. Sentiment Analysis: Once the extractor extracts the review into sentences and NLP, Runner and Coreference works on these sentences the Sentiment Analysis do the classification, here we are

We are using the following functions in CoreNLP: tokenize, s-split, pos, lemma, ner, parse, dcoref, sentiment Annotators

using 5 classes. (Very Negative, Negative, Neutral, Positive, Very Positive).

3. Scoring: Review Analysis does the Sentiment Analysis based on score, this scoring is done based on the key words for reviews and gives each sentence a score, from 0 to 4, where 0 is very negative and 4 is very positive.

4. TF/IDF: Once all the above things are done, we store the snapshot of the output as XML document. This document is then given to TF/IDF to get the important/key terms in the document after CoreNLP techniques. TF/IDF gives all these term a score, we set a threshold to non-negative. So all the terms from TF/IDF output which has negative score are discarded. We only take non negative terms. Then a new document is revised which consists of these non-negative key terms.

5. LDA: We give the non-negative key terms-revised-output as input to LDA which helps us to get the relationship between all the terms, For LDA we pushed this data to Spark System instead of Hadoop because of fast processing. We set the value of no. of topics as 10. So it creates a matrix of 10 topics with 10 subtopics associated to 10 topics each. As, one of the drawback, you have to manually define these topics, we stopped here for this increment.

## • Domain Model

**Data Sources:** We are using Rotten Tomato, as our data source for phase 2, Using Rotten Tomato API we are fetching the reviews based on the name, the user selects and later, these reviews are analyzed. We are collecting maximum 50 reviews for each movie.

Here is the basic example to collect data using Rotten Tomato API: Review Sample URL:

[http://api.rottentomatoes.com/api/public/v1.0/movies/24220/reviews.json?review\\_type=top\\_critic&page\\_limit=50&page=1&country=us&apikey=zsnu54m8bqfzbct8ju4jz88g](http://api.rottentomatoes.com/api/public/v1.0/movies/24220/reviews.json?review_type=top_critic&page_limit=50&page=1&country=us&apikey=zsnu54m8bqfzbct8ju4jz88g)

Movie Search Sample URL:

[http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=king&page\\_limit=50&page=1&apikey=zsnu54m8bqfzbct8ju4jz88g](http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=king&page_limit=50&page=1&apikey=zsnu54m8bqfzbct8ju4jz88g)

## **Algorithms used:**

We have an algorithm similar to map reduce Base forms of words

Sentence Extractor

Parts of Speech

Scoring

Noun for headlining every sentence Sentiment Analysis

Coreference

Tokenization

s-split

Lemmatization

Dcoref

Sentiment Annotators

Here is the pseudocode which does the mapper & reducer job in our application

```
mapping (Coref coref): topic := getTopic (coref)
```

```
for sentence in getSentences (coref): emit (topic, sentence)
```

```
reducing (String topic, Iterator sentences): S := []
```

```
for sentence in sentences: append (S, sentence)
```

```
emit (topic, S)
```

```
reviewAnalyze (String review):
```

```
C := StanfordNLP.getCorefs (review)
```

```
M := {}
```

```
for coref in C:
```

```
put (M, mapping (coref))
```

```
R := reducing (M)
```

```
for topic in R:  
    emit (topic, StanfordNLP.sentimentAnalyze (R[topic]))
```

## **Glue Logic(Mashup)**

Using Rotten Tomato to get movie reviews, using CoreNLP extractor from Stanford to use the algorithms and using coreference with Sentiment Analysis to understand the reviews with words like, “it”, “this”, etc for actor, movie names, etc.

### **Features:**

1. Understanding Context/Topic of every sentence
2. Calculating score for every sentence based on words & meaning.
3. Coreference
4. Context based Sentiment/Opinion analysis for every sentence.
5. Create a snapshot of this output and store it in XML document for every review.
6. Give this XML document to TF/IDF to get key terms in every document
7. Generate a revised document with these key terms after removing the terms below the threshold(non-negative in our case)

8. Give this document to LDA to create a matrix for topic generation.

## **Expected Inputs/Outputs**

Input: Movie Reviews (Upto 50)

Output: Context Based Sentiment/Opinion Analysis, for every sentence, XML Document, TF/IDF Terms, LDA Matrix

Scoring (0 - 4)

0: Very Negative 1: Negative

2: Neutral

3: Positive

4: Very Positive

Example:

Input:

Tim Burton's Batman wasn't a bad movie. It was entertaining and had a nice comic-book feel to it, but it struck me as being quite formulaic. Some scenes worked for me, but others felt forced and oddly out of place. I imagine Tim Burton had some of the acto

Output: "{

"Tim Burton": [ {

"score": 1,



"opinion": "Tim Burton's Batman wasn't a bad movie." },

{

"score": 0,

"opinion": "I imagine Tim Burton had some of the actors, like Jack Palance, overact on purpose, to give the movie a similar feel to the Batman TV show from the 1960s; however, the result felt more like a high-school play than a movie."

}],

"Jack Palance": [ {

"score": 0,

"opinion": "I imagine Tim Burton had some of the actors, like Jack Palance, overact on

purpose, to give the movie a similar feel to the Batman TV show from the 1960s; however, the result felt more like a high-school play than a movie."

}],

"the movie": [ {

"score": 2,

"opinion": "It was entertaining and had a nice comic-book feel to it, but it struck me as being quite formulaic."

}, {

"score": 1,

"opinion": "Tim Burton's Batman wasn't a bad movie." },

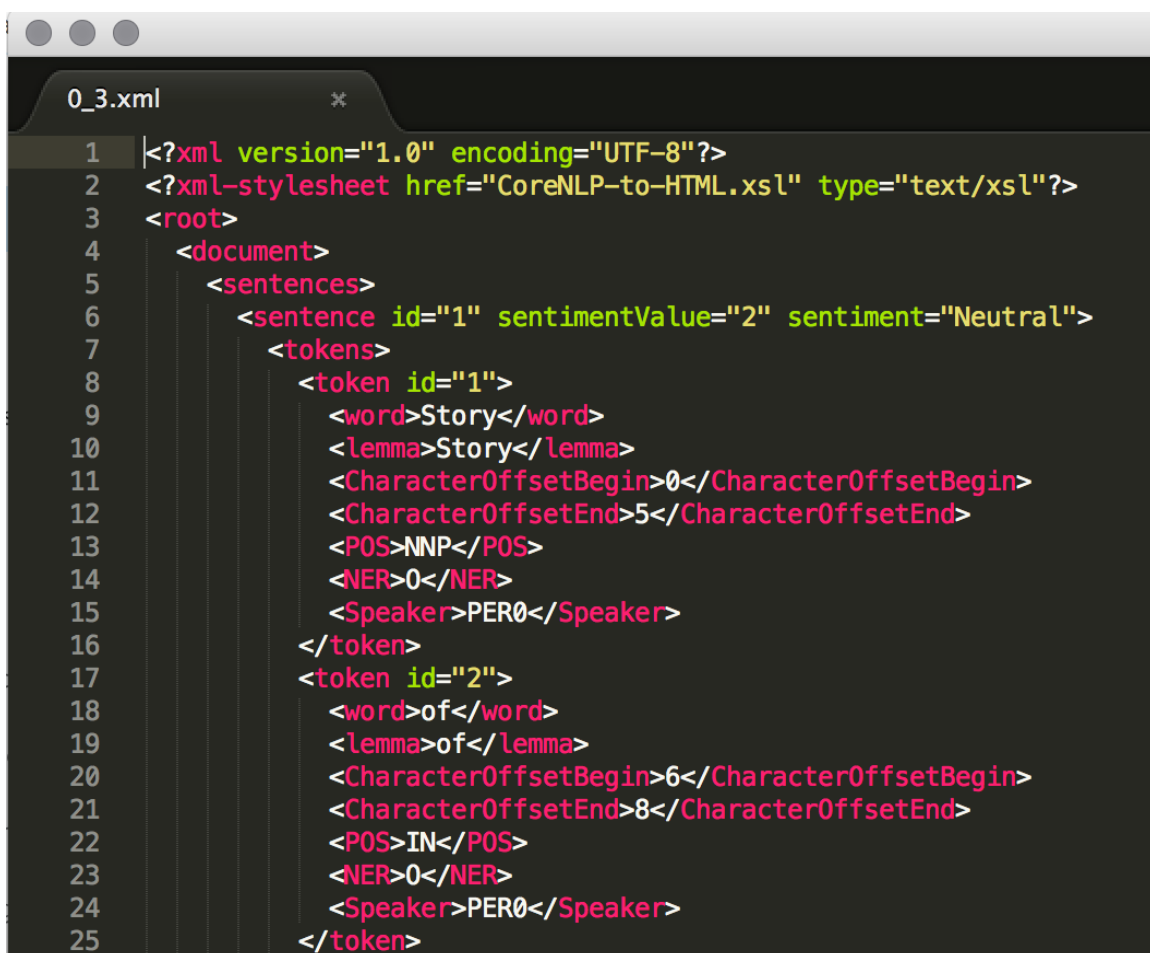
```
{  
  "score": 0,
```

"opinion": "I imagine Tim Burton had some of the actors, like Jack Palance, overact on purpose, to give the movie a similar feel to the Batman TV show from the 1960s; however, the result felt more like a high-school play than a movie."

```
  ]  
}
```

```
}"
```

Here is the screenshot from our XML document



```
0_3.xml  
1 |<?xml version="1.0" encoding="UTF-8"?>  
2 |<?xml-stylesheet href="CoreNLP-to-HTML.xsl" type="text/xsl"?>  
3 |<root>  
4 |   <document>  
5 |     <sentences>  
6 |       <sentence id="1" sentimentValue="2" sentiment="Neutral">  
7 |         <tokens>  
8 |           <token id="1">  
9 |             <word>Story</word>  
10 |            <lemma>Story</lemma>  
11 |            <CharacterOffsetBegin>0</CharacterOffsetBegin>  
12 |            <CharacterOffsetEnd>5</CharacterOffsetEnd>  
13 |            <POS>NNP</POS>  
14 |            <NER>0</NER>  
15 |            <Speaker>PER0</Speaker>  
16 |          </token>  
17 |          <token id="2">  
18 |            <word>of</word>  
19 |            <lemma>of</lemma>  
20 |            <CharacterOffsetBegin>6</CharacterOffsetBegin>  
21 |            <CharacterOffsetEnd>8</CharacterOffsetEnd>  
22 |            <POS>IN</POS>  
23 |            <NER>0</NER>  
24 |            <Speaker>PER0</Speaker>  
25 |          </token>
```

```

1      <CharacterOffsetEnd>52</CharacterOffsetEnd>
2      <POS>.,</POS>
3      <NER>O</NER>
4      <Speaker>PER0</Speaker>
5      </token>
6    </tokens>
7    <parse>(ROOT (NP (NP (NNP Story)) (PP (IN of) (NP (NP (DT a) (NN man)) (SBAR (WHNP (WP who))
8      unnatural) (NNS feelings)) (PP (IN for) (NP (DT a) (NN pig)))))) ( . .)) </parse>
9    <dependencies type="basic-dependencies">
10      <dep type="root">
11        <governor idx="0">ROOT</governor>
12        <dependent idx="1">Story</dependent>
13      </dep>
14      <dep type="prep">
15        <governor idx="1">Story</governor>
16        <dependent idx="2">of</dependent>
17      </dep>
18      <dep type="det">
19        <governor idx="4">man</governor>
20        <dependent idx="3">a</dependent>

```

## • Application Specification System Specification:

IntelliJ IDEA IDE Java 1.6 or higher CoreNLP libraries  
 Play framework Languages used: Scala

Java

General flow of our application

User provide a movie name -> list all possible movies  
 on web page -> user chooses a movie from suggested  
 list of movie -> display 50 first re- views of the select-  
 ed movie -> get XML document -> get Key terms us-  
 ing TF/IDF -> give this to LDA to generate the matrix

**Implementation:**

Algorithms used,

We have an algorithm similar to map reduce Base forms of words

Sentence Extractor

Parts of Speech

Scoring

Noun for headlining every sentence Sentiment Analysis

Coreference

TF/IDF

LDA

We have not deployed this project anywhere as web application or anything, Application works fine in Localhost.

## • Documentation

The screenshot shows an IDE window titled 'FetchReviews.java - [playjournal] - PlayJournal - [~/IdeaProjects/PlayJournal]'. The code in the editor is as follows:

```
17 public List<Review> getReviews(String message) throws IOException
18 {
19     final String reviewList = apiReviews(message);
20
21     List<Review> reviewContainer = new ArrayList<>();
22
23     JsonParser jsonParser = new JsonParser();
24
25     //fetch the movie id, name and release year for all the movies listed for the search
26     JsonObject jsonObject = (JsonObject) jsonParser.parse(reviewList);
27 }
```

The project structure on the left shows a package hierarchy: `app > com > api > rottentomatoes > FetchReviews`. Below the code, a list of 45 reviews is displayed, each preceded by a number and a dash. The reviews are about the movie King Kong. The status bar at the bottom indicates 'Compilation completed successfully in 50 sec (28 minutes ago)'.

As seen in the screenshot, it shows for 45 reviews collected from Rotten Tomato for the movie King Kong. Here you can see, all the reviews are extracted and sentence by sentence parsing is done. Lets take one example here, 45<sup>th</sup> review. This is referred to the set. And the review about this reference is very positive. (4 score)

We follow Batch-Sequential architecture: Cause of its nature: simplicity

Design Flow

Review From Rotten Tomato using their API -> Mapper -> Reducer ->

Analyzer -> Results XML -> KeyTerms -> Topic Matrix from LDA

## **Project Management:**

Implementation status report Work: Rotten Tomato API

To get movie names, movie info and movie reviews based on the names selected by user. We used java code to implement the same.

Time Taken: 4 Hours

Contributions: Vishnu Chelle(50), Ran Chen(50)

Work: Coreference

To use Standard Coreference in our application to understand words like “this”, “that”, etc. We used Scala to implement the same.

Time Taken: 10 Hours

Contributions: Cuong Cu(60), Rishabh Bhojak(40)

Work: CoreNLP

To use CoreNLP in our application using Stanford libraries, which laid the core work of the application. We used scala to implement the same.

Time Taken: 8 Hours

Contributions: Cuong Cu(30), Ran Chen(20), Vishnu Chelle(25), Rishabh Bhojak(25)

Work: Integration of different algorithms used & General Testing

Integrating different algorithms and data sources combined to make the application as a whole. Also, we did a general testing of our application.

Contributions: Ran Chen(30), Vishnu Chelle(40), Rishabh Bhojak(15), Cuong Cu(15)

## **For this increment**

Work: CoreNLP with other functions

Doing all the functions explained earlier and generating the XML file as snapshot output.

Cuong Cu(70) & Rishabh Bhojak (30)

Work: Key Term identification

Understanding XML and using this as a text for TF/IDF to get important terms in the document and getting it in XML format. Using this matrix to generate the re-

vised document contains these key terms once they get filtered by threshold value.

Ran Chen(70) Vishnu (30) Rishabh(10)

Work: LDA with Spark

Push the data to Spark system and use LDA libraries to generate the matrix.

Rishabh(60) Cuong (40)

Work: Report Writing, miscellaneous work & issues fix

Writing the Report & miscellaneous work like GitHub, research, ideas, etc.

Vishnu Chelle (60) Ran Chen (40) Cuong Cu(5)  
Rishabh Bhojak(5)

Work: Issues/Bugs fixes

General issues faced by different members and fixed

Cuong Cu (60)



Vishnu Chelle(10)

Rishabh Bhojak (15)

Ran Chen (15)

Deployment:

Code: <https://github.com/cnsgcu/Stage03>

Documentation: <https://github.com/rbx44/CS560>

References:

Increment report of Stage 2 report, can be very similar.

<https://github.com/rbx44/CS560>