CS 297 Report

SCALABLE SEARCH ENGINE AGGREGATOR

By
Pooja Mishra
SJSU ID: 009295000
Email: pooja192009@gmail.com

Project Advisor : Dr. Chris Pollett

Department of Computer Science
San José State University
One Washington Square
San José, CA 95112

Contents

Introduction

Yioop is a PHP search engine developed by Dr. Pollett. The Yioop search engine is designed to allow users to produce indexes of a web-site or a collection of web-sites. The number of pages a Yioop index can handle range from small site to those containing tens or hundreds of millions of pages. In contrast, a search-engine like Google maintains an index of tens of billions of pages. Nevertheless, since you, the user, have control over the exact sites which are being indexed with Yioop, you have much better control over the kinds of results that a search will return. Yioop provides a traditional web interface to do queries, an rss api, and a function api. It also supports many common features of a search portal such as user discussion group, blogs, wikis, and a news aggregator. Certain search engine tasks such as updating news feeds, rss feeds, sending out notifications are done periodically in bulk. All these different functions are part of the media updates any search engine usually does. Currently, these tasks are not done periodically in Yioop. Yioop has a news updater process that can be used to re-index RSS and Atom feeds on an hourly basis. This more timely information can then be incorporated into Yioop search results.

The list of video and news sites can be configured through the GUI. Yioop has a news_updater process which can be used to automatically update news feeds hourly.

I have worked on building aggregator to periodically update news feeds. This aggregator will be distributed over all machines in the Yioop instance. I have worked on modifying the different features of Yioop by first doing some experimentation with its current load capacity and architecture.

Yioop has following media related facilities that it provides-

News updaters

Google news and Yahoo news are news updaters. Google News is a computer-generated news site that aggregates headlines from news sources worldwide, groups similar stories together and displays them according to each reader's personalized interests.

Traditionally, news readers first pick a publication and then look for headlines that interest them. Yioop has a news updater process that can be used to re-index RSS and Atom feeds on an hourly basis. This more timely information can then be incorporated into Yioop search results.

Video Uploaders

Just like Youtube , user can upload videos in Yioop and once it gets uploaded it can be viewed anytime later. One can also configure the video sources in order to fetch videos from these sources periodically.

Social Groups, Blogs, and Wikis.

Yioop can be configured to allow users to create discussion groups, blogs, and wikis. If Yioop is configured to allow multiple users, then users can share mixes of crawls they create. Blogs and discussion group can be made public or private. Public ones have public RSS feeds and the better amongst these can be chosen for incorporation in what Yioop's news service indexes.

Deliverable 1:

Get familiar with the current feature of news updater and experiment with the current news feed features in the search engine.

I followed following steps in order to get Yioop installed and get the news updater working.

- I downloaded the search engine and installed on the local machine. I did a fresh clone of git and created the working directory.
- I downloaded and installed Xampp server and placed downloaded yioop directory inside htdocs folder.
- I changed the config.php and set WORK_DIRECTORY to the path of yioop_data directory.
- In Manage Machines I added a single machine under Add Machine as local.
- I then restarted my computer to get the yioop working on the localhost.
- I went to search sources and added few newssources in the media section.
- Then I went to manage machines and started the news_updater which is currently working on the name server and let the news_updater crawl and fetch feeds from the news sources.

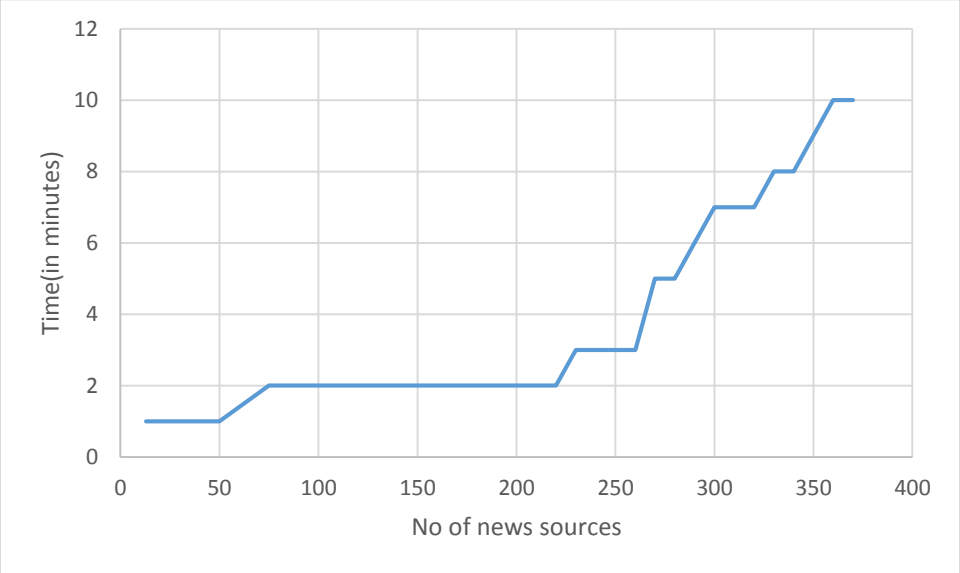
Experiments with news feed capacity

I added few news sources (rss and html type) to the media sources and ran the news_updater by reducing its update time to 60 seconds. If after running the news_updater, the index file was being created under feeds then the test was pass otherwise fail.

No of news feed sources	Time	Index Created?
13	1 min	Y
17	1 min	Y
17	5 mins	Y
17	10 mins	Y
50	1 min	Y
75	1 min	Y
100	1 min	Y
125	1 min	Y
170	1 min	Y
180	1 min	Y
200	1 min	Y
210	1 min	Y
220	1 min	Y
230	1 min	Y
240	1 min	Y

250	1 min	Y
260	1 min	Y
270	1 min	Y
280	1 min	Y
290	1 min	Y
300	1 min	Y
310	1 min	Y
320	1 min	Y
330	1 min	Y
340	1 min	Y
350	1 min	Y
360	1 min	Y
370	1 min	Y

The graph can be plotted as follows :



X-axis – No of news sources

Y-Axis- Time taken in minutes to generate the index file.

After adding few news sources, the news_updater was increasingly taking more time to update.

After almost 275 news sources, it took more than 5-6 minutes to build the index file and after 350 news sources it took almost 10 minutes to generate the index file. We can see that , As the no of news sources goes on increasing, there has been increase in the time it takes to build the index shard. Referring to this chart, we can roughly make an estimate that in an hour single machine news updater can handle around 3-4 k news sources.

Deliverable 2

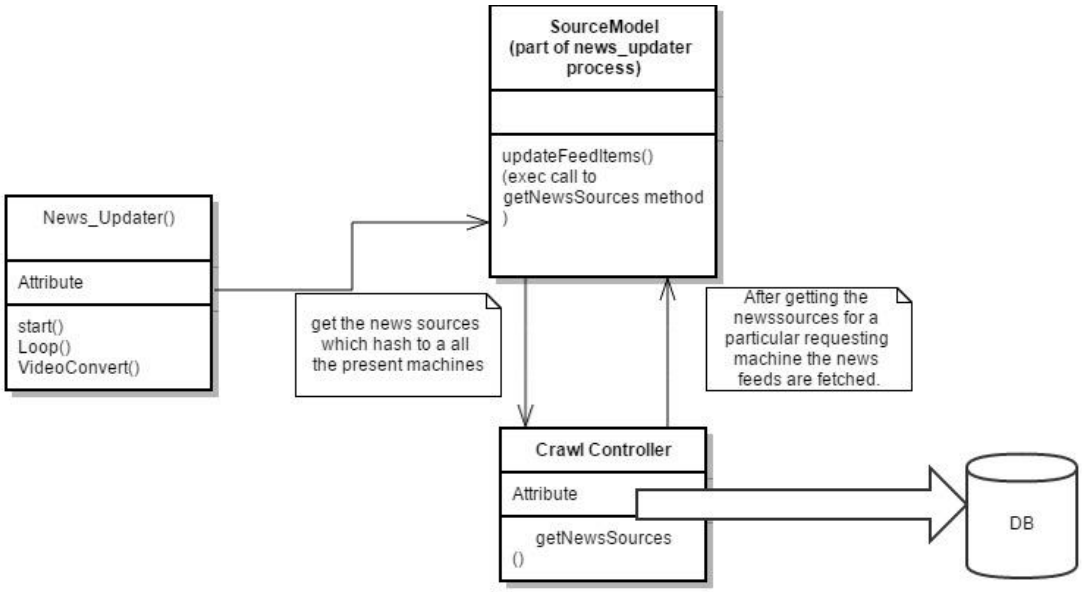
Rewrite and modify existing news updater to work on the distributed system.

Currently, the news updater feature runs only on the name_server machine. It follows the following steps in the existing process.

- News_updater will run after an hour periodically and check if there are any new updates needed.
- If not, it will go to sleep; otherwise,
- it will call the updatefeeditems function from source_model.
- The source_model function will fetch the news_sources (both rss and html) and do the crawl to fetch new news items.

This way the number of news sources and news feeds that could be fetched will be limited after a certain point. In order to speed up the process on the different machines, I modified and rewrote the code to distribute over the no of machines.

Architecture



Process

I modified the news_updater functionality to distribute it over multiple machines, so that each machine can independently run its own news_updater and make a request to the name server and get the news sources from where it can fetch the news.

The flow of the process can be described as below.

News_updater will be running on all the machines except name_server at periodic intervals.

- A variable MULTIPLE_NEWS_UPDATER has been defined in config.php file. If the value of this

Variable is set to true then the news_updater will work in the distributed fashion and if it is set to false then it will run in single machine mode. One can also create local_config.php and set the variable from there instead of setting it from config.php file.

- If the variable MULTIPLE_NEWS_UPDATER is set to true then following events will occur.
- The machine will call the updatefeeditems method from source_model from its news_updater.
- The machine which makes the request to name_server for news updater will be identified by reading in the text file 'current_machine_info.txt' by updatefeeditems method of source_model.
- Updatefeeditems makes en exec call to getNewsSources function in Crawl_controller.
    - The exec call is made through parallel_model and current_machine hash string is appended to the request url.
- The getNewsSources function gets all the news_sources from database and calculates their md5 hash and puts in an feeds array which is having same length as no of machines.
- After the hash values array for all the feeds are calculated then the feeds values corresponding to the requesting machine are returned to the calling function.
- Once the machine receives the feeds_array it will resume its process and fetch the articles from these news_sources.

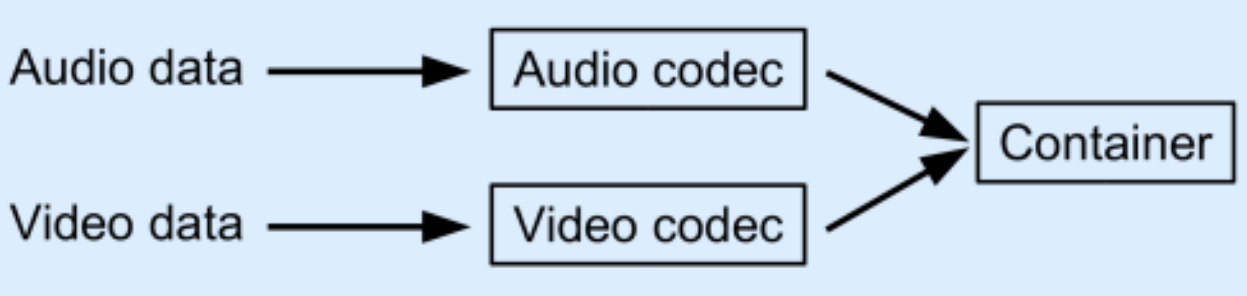
Deliverable 3:

Learn FFMpeg recoding and design the architecture for the video uploader to be incorporated in the system.

Traditional, lossless compression algorithms such as ZIP, gz and bzip2 don't get anywhere near reducing the size of the data enough, so we need to look into lossy compression, ie: compression that is far more efficient but with a trade-off in that the picture quality suffers. The same applies to audio with a degradation of the sound quality. The object of much research over the past few decades has been the elaboration of new algorithms that manage to reduce the quantity of data needed for the audio and video while also reducing the damage done to the picture and sound as a result of the compression. These algorithms that allow us to encode the data in order to transport it, and to decode the data the other end, are called codecs. Several codecs are contained in the libavcodec library supplied with ffmpeg – others are provided by LAME, libgsm, XviD, AMR and libvorbis.

Once the audio and video streams have been encoded by their respective codecs, this encoded data needs to be encapsulated into a single file. This file is called the container. One particular type of container is AVI (Audio-Video Interleaved). AVI is only a method of mixing the encoded audio and video together in a single file. It contains data informing the media player trying to play the file back what audio and video codecs were used, and there are many different possible combinations of codecs that can be used within each type of container, which explains why a system may play back some AVI files and not others. The system can extract the encoded audio and video data from the AVI file no problem, but can it decode that encoded data once it has done so? Only if the required codecs are present.

FFmpeg is a free software project that produces libraries and programs for handling multimedia data. FFmpeg includes libavcodec, an audio/video codec library used by several other projects, libavformat, an audio/video container mux and demux library, and the ffmpeg command line program for transcoding multimedia files.



Playback of a multimedia file operates in the exact opposite direction.. Once the container used is identified, it tells us which codecs are needed to decode the data. The audio and video

streams are then extracted from the container and fed through the appropriate codecs, and out the other end we get raw audio and video data that can be fed to the audio and display subsystems of the computer.

ffmpeg is a tool that, in its simplest form, implements a decoder and then an encoder, thus enabling the user to convert files from one container/codec combo to another, for example a VOB file from a DVD containing MPEG2 video and AC3 audio to an AVI file containing MPEG4 video and MP3 audio, or a QuickTime file containing SVQ3 video and MP3 audio to a 3GP file containing H263 video and AMR wideband audio. The original container is examined and the encoded data extracted and fed through the codecs. The newly-decoded data is then fed through the "target" codecs into the new container.

ffmpeg can also do a few basic manipulations on the audio and video data just before it is re-encoded by the target codecs. These manipulations include changing the sample rate of the audio and advancing or delaying it with respect to the video.

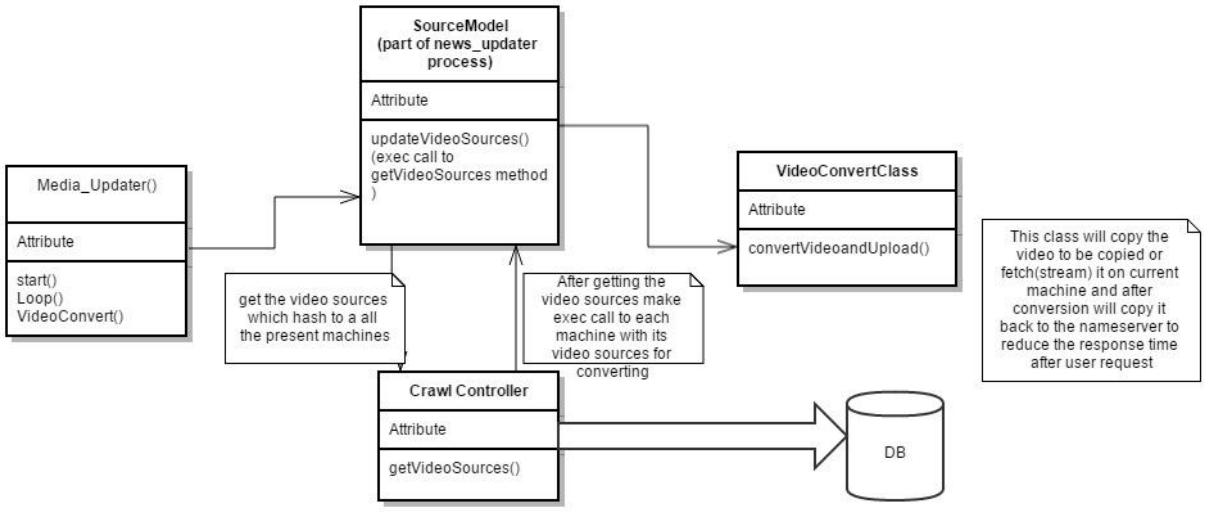
Existing Functionality

In the media_updater, there is a provision for uploading a videos and different video_sources. Certain video formats such as mp4 or webm does not work in browsers such as Mozilla.

In order to convert such video formats to different format, we can use a tool ffmpeg which is described above. Using this linux utility , a script can be used to convert the uploaded video formats in a particular format such as mov and then uploaded back on the server for the users to watch.

As part of this deliverable I have designed the architecture for the video uploader which will be part of Media uploader only.

Architecture



Process

I have designed the architecture for video_uploader which will be part of media_updater and will again run in the distributed process. Below process explains the step by step flow which will take place according to the above architecture.

The sequence of actions can be described as follows:

- The video uploader will be part of media_updater and will be running on all the machines periodically.
- Every machine can invoke its video_uploader(same as news_updater) and check if there are any new videos uploaded or new video_sources updated.
- In case it finds new videos uploaded, it will call updateVideoSource function from source_model.
- The updateVideoSource will then will make an exec call to getVideoSources function in crawl_controller same as news_updater.
- The getVideoSources function in crawl_controller will get the video_sources from the database and calculate its hash and store it in an array and will return the corresponding video sources to requesting machine.
- Once the requesting machine receives the video sources, it will fetch the video on itself and run the videoConverter script on it and after conversion it will upload back on the server.
- Usually converting a small size video will take upto 15-20 minutes. So in this way multiple videos can be converted simultaneously on no of machines.
- After any user has uploaded a video, he/she can be notified that he/she will be able to view the video after a certain period of time.

Conclusion

In this way, I have understood , learnt and experimented with the news feed feature of Yioop and rewrote the code in such a way that it can now work on the distributed system as well as on name server too. I also learnt the ffmpeg tool and its purpose which I can use for video uploader which will be part of media updater in Yioop.