Developing Fully Text Searchable Video

2013

The purpose of this project is to investigate a system which can turn a video lecture recorded by a professor into a complete text transcript automatically. This transcript would be edited by the video viewers to ensure accuracy. The transcript could then be marked up into a book format thereby creating a book format of an entire course with only audio recordings of lectures. This paper looks at the storage technologies necessary to provide this service to a large number of concurrent users (as IIT online lectures are consumed greatly in other parts of the world).

An investigation of the required storage technologies

Christopher King
ITM 497-170
Summer 2013

Project Overview

Generating text transcripts of video lectures allows for numerous improvements to the overall online lecture system. In one use case students who are new to the English language will be able to easily obtain translations of all recorded classes as today's technology provides numerous methods of relatively accurate translation of text documents. In other cases it allows students to obtain a "hard-copy" of their lectures to study from, follow along in review sessions, and even search for keywords in order to find a specific moment in a semester's worth of lectures.

Generating these transcripts for the hundreds of lectures occurring at once on a college campus, while also displaying already rendered transcripts alongside the video, and allowing for each user to edit the displayed transcript in order to easily correct errors requires a fairly complex storage system. This system needs to be able to handle many concurrent users who will be not only reading data from the database but also providing updates to the data in the database. In addition to storing the transcripts the storage system will need to be able to store metadata about each lecture (class, department, professor, date, etc.) and track the location of the actual video and audio recordings.

When a professor first accesses this website they will upload a video recording of their lecture along with a few fields of metadata, including class number, department, date of lecture, etc. This metadata will then be stored in a database while the video lecture is uploaded into an elastic cloud storage system similar to Amazon's S3 service. From there the video will be analyzed by the CMU Sphinx speech to text library and a rough transcript will be created and

stored in another document storage database. This document storage database will hold only the transcript itself (and translations or conversions of the transcript into both other languages and other formats) as the database which holds the metadata will store all other information next to the ID of the transcript in the document database. After the rough transcript is first created subsequent views of the video lecture will display the transcript while allowing editing by the viewer. These edits will be saved back into the document database presenting one of the first backend issues in this process. As multiple users edit the same transcript at the same time some system of merging edits must be put in place. After a sufficient number of users have edited the transcript and accepted it as complete the text transcript will be available for conversion into the DocBook XML format. Another edit view will allow users to drag and drop XML tags representing chapters, paragraphs, and sections into the transcript in order to create a textbook formatted lecture transcript. This book format would allow for easy studying of material as well as an entirely different way to consume a professor's course. This XML formatted document is also stored in the document database so that it can be fully text searchable and easily accessed by numerous students at one time.

Metadata Storage Database

The metadata which describes which class a lecture is from, who the professor giving the lecture is, where the lecture took place, the data of the lecture, etc. must be stored in some kind of database in order to be retrieved later. In addition this same database must also be able to relate the documents in the document database to actual lectures. Because this data will be accessed relatively infrequently (a single original write as the data is not likely to be updated

after upload and only one read per video view instead of constant reading and writing over the course of the video) and the data contains many relationships (this professor teaches this class on this day for this department, etc.) the best solution is an SQL database. One of the most widely used SQL databases is MySQL. MySQL is relatively easy to setup and maintain, provides reasonable scaling, and has numerous APIs to connect our application with the database. While MySQL may not be the most efficient SQL database system it will provide good enough access times for its one read per video view operation. In addition, it is relatively easy to switch between SQL servers if it becomes necessary in the future of this project.

Document Storage Database

The generated transcript should contain sentence delimited lines with timestamps for each word or timestamps for sentences as a whole. The generated transcript will then be fed into a document storage database. Storing a document of unknown length with accompanying timestamps is inherently a hard job for a traditional SQL database. Many NoSQL databases, on the other hand, are document centric and do not require formal schemas meaning data can be stored in the database with any number of fields and each document essentially defines its own schema. In addition NoSQL databases are also very good at horizontal scalability, meaning they are easily able to expand and utilize numerous relatively low powered machines to meet the demand of their users. However, NoSQL databases inherently place much more of the burden of validating, manipulating, and generally working with the data onto the application itself instead of the database system. In order to link the metadata about a specific lecture with its transcript the id of the document in the NoSQL database will be equivalent to the ID of the

lecture in the SQL database. This allows for the appropriate storage of the document while still maintaining the relational powers of an SQL database. Figure 1 shows a potential schema for the SQL database and how it would relate to a document in the document database.

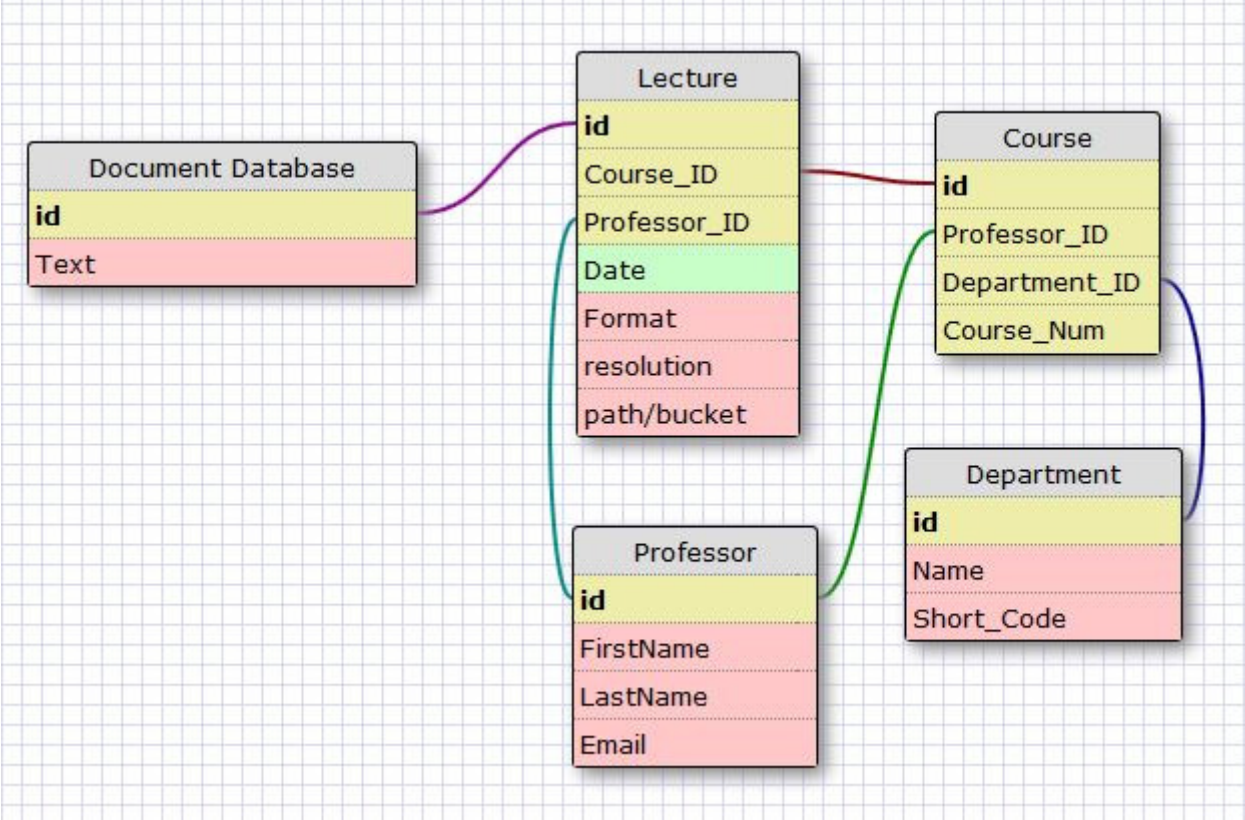

Figure 1

After the document is stored initially into the database it will then be served to a user who will watch the lecture while also correcting the transcription. This is a tricky part of the process as the document will need to be updated appropriately and multiple people may be watching and editing at the same time. Because of this it is important that our document database can store multiple versions of each document. If changes are submitted while another

user has the document open for editing a new version of the document should be created to be merged in some way with the other currently open documents. This could be implemented on a sentence/section level so that on each “page load” of the transcript on the edit page any changes that are made are synced to the database (potentially going through a merge algorithm if there are conflicts). It could also be implemented over the document as a whole. When the video is loaded the webpage would grab the latest copy of the document storing changes temporarily until the video is finished or the user choses to save manually. Again when the changes are pushed to the database a merge algorithm would need to be applied (if another user has submitted changes already) to hopefully combine the versions. If there are unresolvable conflicts after the merge multiple versions of the document would be stored into the database for later human intervention perhaps through some administrator interface. Another simple versioning system could be created similar to Wikipedia’s edit page displaying a list of versions of the documents. In this system if conflicts cannot be resolved through a merge algorithm the most recently submitted version could be accepted and the user would be able to see where his version is in the list allowing them to resubmit their changes.

The final step in the process is the creation of DocBook formatted XML from the edited transcript. The application will retrieve the most current version of the document from the database, display it to the user, and allow them to place DocBook XML tags in appropriate locations throughout the document. This part of the application is significantly less likely to receive multiple concurrent users on the same document but a similar (if not the same) versioning system will need to be implemented here as well. However, it may be possible to craft the available XML tags in such a way that most users place them in the same locations. For

example, a user would be unlikely to misplace a section/chapter tag when the professor says "Now we are going to talk about X." In order to store the rendered XML another "version" of the document will be stored in the document database. Depending on the type of database ultimately used the XML document may be stored as a blob within a JSON document, translated to JSON format, or stored natively as XML in the database.

NoSQL Database Options

There are numerous NoSQL database options available which all provide their own benefits to our project. The two main categories which suit this project are native XML databases and (mostly) JSON based document stores. Native XML databases have the advantage of storing their documents in XML, a format that suits transcripts very well. These databases are built from the ground up to provide fast and intuitive access to XML files. JSON based document stores can do all the same things as XML databases except their data is stored in a JSON format. The lightweight JSON format allows for unparalleled speed and scalability with most databases providing access to any piece of data in under a millisecond.

eXist-db is an XML document database which provides many of the necessary requirements for this project. It has built in minor versioning support giving it the capability to store only the differences between versions of a document instead of storing another copy of the entire document. In addition the versioning support also includes rudimentary support for detecting conflicting writes. However, much of the system will still need to be implemented by our program instead of allowing the database to handle it by itself. eXist-db also includes integrated full-text search based on the popular Lucene search engine library. It also utilizes the

XQuery query method for access to documents which is an XML based query language completely different from SQL. One of the biggest disadvantages of this database is its relative lack of scalability when compared to some of the more popular NoSQL databases. However eXist-db still has horizontal scaling out of the box allowing multiple servers to process information requests in order to improve access time. eXist-db is definitely the top candidate for XML document storage databases.

The top JSON document storage system is likely Couchbase server. Couchbase is based on CouchDB, one of the original NoSQL database systems, giving it maturity, scalability, and speed unmatched by most other document storage systems. Couchbase is extremely scalable with out of the box support for clusters and enormous horizontal scaling. With only a few clicks in the Couchbase server console a cluster can grow from a single server to hundreds of servers providing unparalleled scalability nearly instantly with no downtime. In addition Couchbase is fully Memcached compatible which allows for submillisecond response times. Couchbase also automatically spreads the workload of an application across all servers available to it allowing for even better scalability and performance. One of the biggest disadvantages of Couchbase for our application is its inability to retrieve or update only a part of a document at a time. Whenever a document is operated on the entire document must be retrieved. This could present a problem when retrieving transcripts which are many pages long and only need a single word changed. Another issue with all JSON document stores is the storage of the DocBook XML. There is no efficient way to store XML in a JSON document store, although it is possible to store it as a blob within a JSON document. However, storing the XML as a blob leads to other problems with accessing and modifying the XML document and needlessly complicates

the process of parsing, editing, and displaying the XML by adding an extra layer to the process. Overall it may simply be better to opt for a slightly less efficient and scalable database which stores documents natively in XML format.

Resources

1. Apache Lucene - Welcome to Apache Lucene. (n.d.). *Apache Lucene - Welcome to Apache Lucene*. Retrieved July 21, 2013, from http://lucene.apache.org
2. CMU Sphinx - Speech Recognition Toolkit . (n.d.). *CMU Sphinx - Speech Recognition Toolkit* . Retrieved July 21, 2013, from http://cmusphinx.sourceforge.net
3. Couchbase | Document-Oriented NoSQL Database. (n.d.). *Couchbase | Document-Oriented NoSQL Database*. Retrieved July 21, 2013, from http://www.couchbase.com/
4. MySQL :: The world's most popular open source database. (n.d.). *MySQL :: The world's most popular open source database*. Retrieved July 21, 2013, from http://mysql.com
5. NOSQL Databases. (n.d.). *NOSQL Databases*. Retrieved July 21, 2013, from http://nosql-database.org/
6. NoSQL. (n.d.). *Wikipedia, the free encyclopedia*. Retrieved July 21, 2013, from http://en.wikipedia.org/wiki/NoSQL
7. eXist-db Open Source Native XML Database. (n.d.). *eXist-db Open Source Native XML Database*. Retrieved July 21, 2013, from http://exist-db.org