DETAiLS FUSE System Architecture
Columbia University
Contract Number: D11PC20153

PI: Kathleen McKeown
Technical Lead: Suvarna Bothe
Software Engineer: Arfath Pasha

1. Architecture Motivation

The DETAiLS architecture design was motivated by two concerns: efficiency and modularity. This project involves processing very large corpora of very long full text documents, and investigating the use of potentially computationally intensive natural language, network and time series features over that corpora. As such, it was necessary to develop a robust architecture capable of handling both the large scale of the input corpora as well as the complexity and diversity of techniques to be applied to that corpora. Equally important to these efficiency goals was designing a system flexible enough to support the ongoing research of the team. Since the FUSE problem invites techniques from a variety of domains, it was necessary to make a system that allowed researchers to independently develop and test components to compute new features as well as allowing them to enable or disable other components on the fly.

While we designed the architecture with all collections in mind, the midterm evaluation system was configured to run with Elsevier and WOS documents only.

2. Input Data

Our input data were a set of corpora of scientific documents provided by IARPA; Web of Science, Elsevier, Open Access, and Nature. Web of Science (Reuters, 2013) is the largest collection, containing 48 million records. It is a collection of metadata records with abstracts culled from full text documents by Thompson Reuters. Elsevier is a full text corpus containing 3.8 million records provided by Elsevier Science B.V. (B.V., 2013) Open Access is a collection of 220,000 full-text records obtained by crawling the web, and is mostly drawn from PubMed articles. Nature is a collection of 200,000 full text records from the Nature Publishing Group (Group, 2013)(Group, 2013).

Since our input data is drawn from such a large collection of different sources with varying formats and conventions, we developed a central point at which we process documents and transform them into a single, uniform internal representation. We also handle text that is badly formatted or contains OCR errors in this step, taking advantage of the Parscit tool (Councill, Giles, & Kan, 2008).

3. Architecture Design

3.1 Pipeline

Our pipeline is broken down into three general steps: Shard Generation, Document Level Processing, and Shard Level Processing. In the first step, we iteratively select a collection of documents, or shard, that is relevant to the current entity. This selection is driven by the needs of the shard level processing components. In the second step, each document in each shard is processed individually using our core NLP pipeline. The third and final step involves the processing of the documents at the shard level. The Spring Framework is used to manage the modular enabling or disabling of components on the fly by researchers (Johnson et al.,)

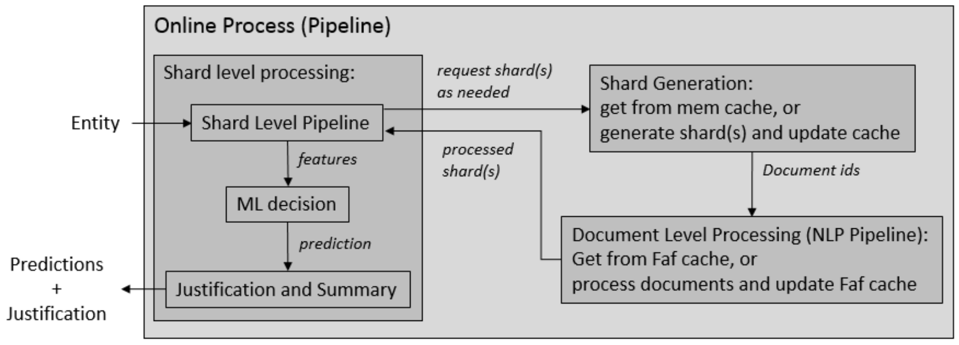
Online Process (Pipeline)
Shard level processing:
Entity
Shard Level Pipeline
features
ML decision
prediction
Justification and Summary
Predictions + Justification
request shard(s) as needed
processed shard(s)
Shard Generation: get from mem cache, or generate shard(s) and update cache
Document ids
Document Level Processing (NLP Pipeline): Get from Faf cache, or process documents and update Faf cache

Figure 1: The main pipeline used for online processing. It contains three high level stages - shard generation, document level processing and shard level processing. The system accepts entities as input and provides a prediction and justification as its output.

3.2 Shard Generation

Motivated by the idea that science is done in individual articles, the shard generation process is designed to take an input entity, either an author, document or term, and compute a set of shards; each shard is a set of documents related to that entity. In a nod to the modular nature of the system, the set of shards computed on each run of the system is determined by the shard level processing components being invoked. Shards are cached in memory and do not need to be recomputed if required again in future iterations or in the processing of another entity.

The goal of the shard generation step is to winnow down the entire ˜55 million document collection to a smaller set of documents that are actually relevant to the current term being requested. In some sense, each shard is designed to provide the components requesting it the answer to some question, i.e. "Can I have all documents with this term in the abstract?" or "Please give me all documents published by this author and all documents cited by those documents."

The shards are generated using one of two methods - Exact Match and Expand One. Exact Match generates a single exact document that matches the entity name when the entity is a document. When the entity is a term, a set of documents that contain the term in the abstract or title is generated. Expand One generates an initial shard utilizing the Exact Match method and then expands the matched nodes in the shard by one in its citation graph.

For the midterm evaluation system, we restricted generation of shards to be the same for all componnets, using Exact Match for term entities and Expand One for document entities.

The shard generation step relies heavily on our back-end database and index, described in 4.1 and 4.3, respectively, to perform searches over the full corpora of documents. This allows us to scan the entire corpora and pick our subset efficiently using both full-text search, with the index, and relational queries, with the database.

3.3 Document Level Processing

With a goal towards more offline processing, we separate out all of the processing that can be done on a single document without reference to other documents into a separate step. This is primarily our NLP pipeline. Once a document has been processed, it is stored in persistent memory. Since this processed data does not reference any of the other documents in the shard, we can also do some of the processing in an offline preprocessing step before computing entities. This allows us to use the available computational resources more efficiently during the online process.

To perform the document level processing, we use the UIMA framework developed at IBM (Ferrucci & Lally, 2004). After processing in UIMA, we transfer the results into our own internal format, known as File Annotation Framework (Faf), and store them in a database. The processing occurs as a pipeline, with the order and specific list of components to run determined by the researcher.

During document level processing, indicator values are produced by the following components: concept detection, relation extraction, argumentative zoning and citation sentiment. These are all run by the core NLP system.

Both concept detection and relation extraction comprise the information extraction component. Concept detection identifies concepts such as *genes*, *algorithms*, or *problems*, while relation extraction detects more complex text structures, such as *novelty claims*, *funding information*, and the *purpose of a dataset* in a paper.

Argumentative zoning is a technique for automatically annotating the rhetorical structure of a scientific paper (Teufel, 2010). Our system component is a maximum entropy-based sequence classifier that labels each sentence in a document with one of six labels: *Own*, *Background*, *Contrast*, *Basis*, *Aim* and *Other*.

The citation sentiment component is a Support Vector Machine classifier that labels each citation-containing sentence in the document as bearing *Positive*, *Negative* or *Neutral* sentiment towards the cited paper(s).

3.4 Shard level processing

After shard generation has returned a set of shards, and all of the documents in those shards have been processed by document level processing, the collection is passed to the shard level processing. This is the level at which our network analysis, time series analysis, and aggregate analysis of document level features occurs. Each component here is designed to take the set of computed shards and, potentially, the results of previous components, and compute features that will support a decision about the prominence of the given entity.

Aggregate features corresponding to indicators are produced for each shard. Some of them correspond to features produced at the document level.

For instance, one of the document level indicators that the system produces in the relation extraction component informs whether the document includes a claim of novelty (e.g., a new approach to solve a problem, a new family of genes or proteins, among others). Another set of indicators come from the concepts that have been detected in text. Specifically, we produce indicators for each concept type indicating the proportion of mentions of a type in the shard. For instance, if a shard includes documents with 2 mentions of algorithms and 1 mention of genes, then the indicators will report that 66% of the concepts correspond to algorithms, while the remaining 33% corresponds to genes. The argumentative zoning and citation sentiment produce both total counts and average proportions for each label they define, aggregated over all sentences in the shard that mention the entity. For example, one AZ feature for a document entity is the proportion of sentences assigned the label *Background* out of all sentences containing a citation to the entity,

Some of these correspond to features produced only at the shard level. For example, the author collaboration network and the citation network for the shard are produced along with network metrics that represent the importance of these two indicators. We cache the global citation network for the entire document collection in advance and during run-time, only read the projection of the citation network on the shard documents from this cache. The author collaboration network is generated on the fly. We compute several metrics on these network. For example, we compute the Watts Strogatz clustering coefficients for both networks, which is a measure of tendency of the nodes in these networks to form clusters.

The time-series component is applied to all aggregated features collected by other components. It is responsible for tracking the aggregated features over time and computing new temporal features that can show the impact of underlying indicators over time. These features are generated on the fly and they are fed to the emergence component. A simple example of a time-series feature is the slope of the best-fitted linear function. With this feature we can detect whether the underlying feature tracked over time has an upwards or downwards trend.

There are two components that do not fit this paradigm, EmergenceDetection and Justification. EmergenceDetection takes all of the previously computed features and makes a decision about the prominence of the input entity. Justification takes the input features and the EmergenceDetection decision and produces a human readable justification of our system's result.

To support Justification in its goal, we built a system whereby the shard-level components talk to each other in arbitrarily-typed features; moreover, each of those features must contain reference to the feature or data that generated it. This produces clear trails of evidence for Justification to follow in its work. The data referenced is either a paper, or in the case of the NLP features, a specific sentence from that paper.

4. Caching and offline processing

Due to the large size of the corpora this system needs to process, it was necessary to build a system that was designed to cache as efficiently as possible. Our main unit of storage is a single document. This allows us the flexibility to see the same document in two different sets of documents but only process it once, regardless of context. We can also process documents before running the system in an offline step by iterating through documents in the collection. Since the document level processing is designed to process anything not in cache on the fly, offline processing can be done whenever experimentation is not occurring. The system stores the results of its analysis on documents into a SQL database.

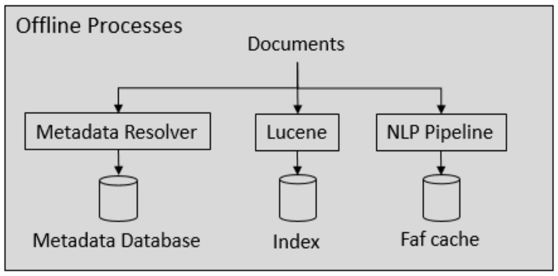

Figure 2: Offline processing of data to save processing time in the online process.

4.1 Database

Much of the metadata in a scientific article is highly structured information, and is well suited to storage in a relational database. We use MySQL as our database of choice (MySQL, 2006). There are two databases in our infrastructure, one of which contains only metadata, and the other contains the results of caching document level processed data. Our metadata database is designed to the specific metadata we find in the corpora we process. The database to store our document level processing is dynamically generated based on the components that have been run on a given document.

4.2 Entity Resolution

One of the biggest challenges facing the use of our metadata database is the resolution of specific named entities within our corpora. For the resolution of author entities within Web Of Science, we utilize a dataset computed based on the methods in Wick et. al.(Wick, Singh, & McCallum, 2012) For resolving articles, institutions, venues, grants and authors outside of Web of Science we use an implementation of the Duke deduplication system (discriminative hierarchical model for fast coreference at large scale, 2012).

4.3 Index

In addition to the highly structured information contained in the metadata, each article also contains a large amount of unstructured information in it's main text and abstract. A full-text index was necessary to allow our system to search this information efficiently, and we chose to use Lucene to implement this index (Lucene, 2012). The index allows keyword search over the full text, abstract, title, and several other metadata fields, as well as more complicated queries combining these fields.

5. Parallelization and System Performance

The system's online process is parallelized with the use of a load balancer that converts incoming requests into batched jobs that can be evenly distributed to pipelines running on multiple machines on the network. The results from the pipelines are collected and merged by the load balancer before being sent to the requesting entity. Each pipeline also takes advantage of Java's support for multi-core optimization to further enhance processing speed.

The system has been tested on a distributed system containing eight Linux Virtual Machines (VM). Each VM was configured to have four Intel Xeon processors rated at 2.66GHz. The pipelines and load balancer were set up to run on six of the eight VMs and the databases were set up to run on the remaining two VMs. The system took 2-5 minutes to process one entity per pipeline. The pipelines ran faster when they were able to benefit from cache lookups during the shard generation and document processing stages. They ran slower when some or all of the shards had to be generated and/or some or all of the documents in these shards had to be processed and the respective stores updated.

The system requires about a terabyte of storage space for the data processed from 55 million documents.

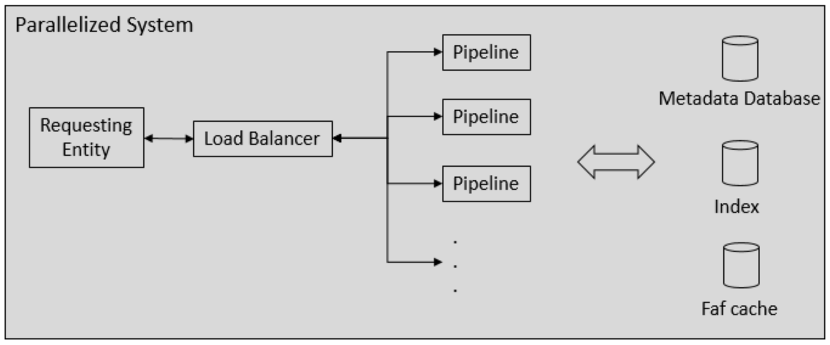

Figure 3: The parallelized system made up of a load balancer and a number of pipelines.

6. Appendix - Software dependencies

Apache Commons	http://www.apache.org/licenses/LICENSE-2.0 (Apache 2.0)
Berkeley Parser	http://code.google.com/p/berkeleyparser/ (GPL V 2.0)
ClearTK	http://code.google.com/p/cleartk/ (BSD)
Duke	http://code.google.com/p/duke/ (Apache 2.0)
JUNG	http://jung.sourceforge.net/license.txt
LibLinear	https://github.com/bwaldvogel/liblinear-java/blob/master/pom.xml (BSD)
Lucene	http://www.apache.org/licenses/LICENSE-2.0.html (Apache 2.0)
MySQL	http://www.mysql.com/about/legal/licensing/index.html
OpenNLP	http://www.apache.org/licenses/LICENSE-2.0 (Apache 2.0)
Parallel Colt	https://sites.google.com/site/piotrwendykier/software/parallelcolt (LGPL V2.1)
Parscit	http://www.gnu.org/copyleft/lesser.html
R	http://www.r-project.org/Licenses/AGPL-3 (GPL 3.0)
Scala	http://www.scala-lang.org/node/146
Second String	http://secondstring.sourceforge.net/ (University of Illinois/NCSA Open Source L
Semafor	http://code.google.com/p/semafor-semantic-parser/(GPL v3)
Simmetrics	http://www.aktors.org/technologies/simmetrics/ (GPL)
Spring Framework	http://www.apache.org/licenses/LICENSE-2.0 (Apache 2.0)
Stanford Chinese Word Segmenter	http://nlp.stanford.edu/software/index.shtml (GPL V 2.0)
Stanford dependency parsing	http://nlp.stanford.edu/software/index.shtml (GPL V 2.0)
Stanford Part-of-Speech Tagger	http://nlp.stanford.edu/software/index.shtml (GPL V 2.0)
UIMA	http://uima.apache.org/license.html
uimafit	http://www.apache.org/licenses/LICENSE-2.0 (Apache 2.0)
Weka	http://www.cs.waikato.ac.nz/ml/weka/index.html (GPL V3)

References

(2012). Duke..

B.V., E. S. (2013). Elsevier...

Councill, I. G., Giles, C. L., & Kan, M.-Y. (2008). Parscit: an open-source crf reference string parsing package.. In *LREC*.

Ferrucci, D., & Lally, A. (2004). Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, *10*(3-4), 327–348.

Group, N. P. (2013). Nature publishing group...

Johnson, R., et al. Spring framework reference documentation 3.0. 2004-2010. *See http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html*.

Lucene, A. (2012). Apache lucene..

MySQL, A. (2006). Mysql 5.0 reference manual. *Bestandteil des Programnmpaketes*.

Reuters, T. (2013). Web of science. 2012..

Teufel, S. (2010). *The Structure of Scientific Articles: Applications to Citation Indexing and Summarization*. CSLI Publications, Stanford, CA.

Wick, M., Singh, S., & McCallum, A. (2012). A discriminative hierarchical model for fast coreference at large scale. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pp. 379–388. Association for Computational Linguistics.