GatorDSR: University of Florida Knowledge Base Acceleration Notebook

Morteza Shahriari Nia
University of Florida
Department of Electrical & Computer Engineering
Gainesville, Florida, USA
m.shahriarinia@ufl.edu

Christan Grant, Yang Peng, Daisy Zhe Wang
University of Florida
Department of Computer & Information Science & Engineering
Gainesville, Florida, USA
{cgrant,ypeng,daisyw}@cise.ufl.edu

Milenko Petrovic
Institute for Human and Machine Cognition
Ocala, Florida, USA
mpetrovic@ihmc.us

September 17, 2013

Abstract

In this paper we will present the system design and algorithm adopted by the GatorDSR team, University of Florida to efficiently process TREC KBA 2013 — SSF track. Here we will describe the system evolution as well as the details of our algorithm to extract slot values for the given slot name. Scalability and efficiency in precision and recall are the major goals of this work, given the overly limited time limitation and available computational resources.

1 Introduction

In this article we will describe the approach the GatorDSR team from the University of Florida takes at addressing the challenge at NISTs Text REtrieval Conference (TREC) - Knowledge Base Acceleration track 2013 - Stream Slot Filling (SSF) section. In the first section of this article we describe the system design and articulate its evolution during the whole process, challenges and needs that we have overcome. In the second section we will go through the

details of the NLP techniques we took at streaming slot filling (SSF): given a slot name (entity + relation type) we fill in the slot value (relation value) on structured text documents.

The task is that given certain Wikipedia or Twitter entities (wiki/twitter URLs) and certain relations of interest given in Table 1, extract as many binary relations as possible (hence, slot filling). The corpus is in English and is a snapshot of part of the web containing arXiv, social, news, linking, weblog, forum, classifieds, reviews, and memetracker documents. Each document is annotated using lingpipe and is called StreamItem, a bundle of StreamItems are put together and serialized as Apache Thrift objects, then compressed using xz compression with LempelZivMarkov chain algorithm (LZMA2) and finally encrypted using GNU Privacy Guard (GPG) with RSA asymmetric keys. To have a sense of the scale of objects and compression is e.g. 6mb gpg.xz files 45 mb thrift objects. Some of the documents have null values for their annotation fields.

Since this is *streaming* slot filling, we are only interested in new slot values that were not substantiated earlier in the stream corpus, which ranges from October 2011 to February 2013. Below you can view the slot values and their types, the comprehensive description of which can be found at http://www.nist.gov/tac/2012/KBP/task_guidelines/TAC_KBP_Slots_V2.4.pdf and http://projects.ldc.upenn.edu/ace/docs/English-Events-Guidelines_v5.4.3.pdf

Example 1: "Matthew DeLorenzo and Josiah Vega, both 14 years old and students at Elysian Charter School, were honored Friday morning by C-SPAN and received $1,500 as well as an iPod Touch after winning a nationwide video contest."

target_id: http://en.wikipedia.org/wiki/Elysian_Charter_School

Affiliate: "Matthew DeLorenzo" Affiliate: "Josiah Vega"

NOT an affiliate: "C-SPAN" NOT an affiliate: "iPod Touch"

Example 2: "Veteran songwriters and performers Ben Mason, Jeff Severson and Jeff Smith will perform on Saturday, April 14 at 7:30 pm at Creative Cauldron at ArtSpace, 410 S. Maple Avenue."

target_id: http://en.wikipedia.org/wiki/Jeff_Severson

Table 1: Ontology of Slot Name Categories

PER	FAC	ORG
Affiliate		
AssociateOf		
Contact_Meet_PlaceTime		
AwardsWon	Affiliate	Affiliate
DateOfDeath	Contact_Meet_Entity	TopMembers
CauseOfDeath		FoundedBy
Titles		
FounderOf		
EmployeeOf		

Affiliate: "Ben Mason" Affiliate: "Jeff Severson" Affiliate: "Jeff Smith"

NOT an Affiliate: "Creative Caldron" NOT an Affiliate: "Art Space"

Example 3: "Lt. Gov. Drew Wrigley and Robert Wefald, a retired North Dakota district judge and former state attorney general, unveiled the crest Friday during a ceremony at the North Dakota Capitol."

target_id: http://en.wikipedia.org/wiki/Hjemkomst_Center

Contact_Meet_PlaceTime: "Friday during a ceremony at the North Dakota Capitol"

2 System

In this section we describe the system design for big data analysis regarding TREC KBA the Stream Slot Filling (SSF) task. Due to our limited time, human resources and the computational capabilities, we started off by trying to use the off-the-shelf tools for this task. Mainly we started by using Scala/Spark. But Spark was not as efficient as expected, then we got rid of it and tried to use Scala parallelization itself to do the job, and it didnt satisfy our needs, there was excessive memory overhead. We migrated the core of the system to Java Parallelism APIs and that was not good enough either, we still needed better. So we built our own parallel system which in actual performance had the least of overhead, the least memory consumption and the most robust memory model which avoided unpredictable CPU stalls. In what follows we will describe each step of this evolution in details. The system that we run our pipeline over was a 32 core server having 64GB of memory.

2.1 Spark

Spark is an open source cluster computing system that aims to make data analytics fast both fast to run and fast to write. To run programs faster, Spark provides primitives for in-memory cluster computing: your job can load data into memory and query it repeatedly much more quickly than with disk-based systems like Hadoop MapReduce.To make programming faster, Spark provides clean, concise APIs in Scala, Java and Python. You can also use Spark interactively from the Scala and Python shells to rapidly query big datasets [1].

Spark is supposed to be an in memory map-reduce versus Hadoop which accesses disk in each iteration. Spark can also run on the multi-cores of a single processors by replicating the binary code of the system. For example by replicating Jar files of the system code into each process local memory and running each in parallel while having an abstract layers to merge/match results and recover from failures. We experimented with various parameters trying to maximize the performance of Spark in processing the corpus and extracting entity StreamItems.

The initial system was coded in Scala (which has goals for scalability) with double systems being coded along side with each other to explore the capabilities of Scala/Spark and pick the best out of each world. One approach was trying

to be as smart as possible trying to use all the cool features of Scala/Spark with the hope that all these optimizations will help us in the long run; another approach was looking into KISS metrics (Keep It Stupid Simple) the ideology to make sure we do as little processing per data unit as possible, keep as little data references as possible and taking care of everything ourselves instead of letting Spark decide for us.

Unfortunately for our scenario it did not perform as expected. It seems to be targeted at large scale cluster systems whereas for small scale systems it was not as performant. JVM was too slow Garbage Collection caused large pauses because of memory fragmentation. We really needed more control over the memory usage. Spark suffered from huge memory dependencies in multi-stage map-reduce tasks which would be used in failure recovery in big data cluster processing. This is an essential need for such a system as otherwise there is no point to migrate from a generic Hadoop system to Spark. Reliability overhead was a very big disadvantage of Spark which sacrificed performance. The CPU usage was at 100Spark RDD does not perform better that parallel scala. The intermediate collections they create also cause GC problems and there is also a non-trivial startup time for each job.

On huge memory consumption of spark, one of the features of spark we ignored is that in spark if a system fails while the map-reduce is still in progress it can specifically rebuild that section and doesn't need to do the whole job from scratch. It makes sense if they keep references of the upstream objects even if a piece of a multi-stage map-reduce is done for reliability and not release its memory.

Given that the already generated lingpipe is not very accurate, our initial ambition was at using StanforNLP. Using Spark and StanfordNLP our statistics were at around 500 documents processing in an overnight execution which was unacceptably slow given our data scale challenge.

2.2 Scala Parallelism

From Spark to scala parallelization: In Scala, using 'map' or 'filter' to iterate over large collections creates intermediate collections. These variable may hold reference to large objects and delay their disposal. Instead using a mix between 'withFilter' and 'foreach' let's you iterate without creating another intermediate collection. Conceptually a reduce job is an aggregate over the set of tasks that have been mapped, in our scenario we could bypass this step and process the corpus regardless of different sections, hence avoiding map-reduce. Onthe other hand the the method scala offers to download/decrypt the files (ProcessBuilder class) leaks resources, which was a bug that was later on fixed in Scala 2.10 form version 2.92 that we were using on our server.

The take-away was that Scala is a very powerful language and can help in concise code for map-reduce type of jobs that can be coded extremely fast. and Spark is a powerful tool but neither could prove their performance to us even after plenty experiments and parameter adjustments. In the long run, scala could not get rid of memory references and after a while we were clogged up in

Table 2: Ontology of Slot Name Categories

1	2	3
4	5	6
7	8	9

Table 3: Ontology of Slot Name Categories

1	2	3
4	5	6
7	8	9

heap space.

Below you can view the profiling of the system in scala using Java VisualVM. As you can see the CPU utilization is very unpredictable and we lose CPU utilization after a while.

Thus ended up being saturated as below - zero CPU utilization, Heap overloaded, almost all threads sleeping:

After fixing memory issues we could optimize to:

The jump in the number of the classes was due to the java profiling tool we use JVisualVM.

2.3 Java Parallelism API

After porting the project to Java and migrating scala sbt to java's Maven build tool and going through some major optimizations we could achieve several major gains:

- Reduce the memory usage by 73
- We can use the newly available memory for fast clustering, classification, etc. for slot filling.
- Reduce unwanted run-time class generation by 80classes to 1,500)
- Reduce # of threads by 47

  ...

cons:

Table 4: Ontology of Slot Name Categories

1	2	3
4	5	6
7	8	9

Table 5: Ontology of Slot Name Categories

1	2	3
4	5	6
7	8	9

Table 6: Ontology of Slot Name Categories

1	2	3
4	5	6
7	8	9

- Increased run-time by 3than we just manually trying to adjust the parameters in java.

  ...

An interesting phenomona that happened was that JVM appeared to be adapting to the behavior of the system, after around 20-30 minutes of execution JVM adapted heap size by 50

We tried various parameters (# of threads 31,32, 64, 128, 256) and via extensive simulations 31 was the optimal one but the margin was very small from 32 (one thread per core), this could be due the race conditions at the server for other services running. The performance was linear over time. The code base is as simple as possible and as smart as possible to ensure scalability demands.

The code is as simple as possible to ensure there's no reference kept any where and there is no extra processing. CPU utilization is at 98

To put things into perspective and talk in numbers:

- org.apache.thrift is the dominant CPU time user
- Then is the decompression of thrift objects and decrypt the RSA encrypted data (note that asymetric cryptography is very expensive and we are looking at some thing like 2 seconds per file)
- Then is streamcorpus (the TREC KBA library to deserialize from their StreamItems objects).
- Our code takes less than 3% of the CPU time.

Table 7: Ontology of Slot Name Categories

1	2	3
4	5	6
7	8	9

2.4 Bare Java Thread Pool

Following our efforts to speed up the system we even went down more low level and created our own thread pool instead of using Java's ExecutorService Library. Still according to java visualvm more than 40% of CPU time is taken by sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(). In ExecutorService this number was even higher at 60%-80%. We were expecting ExecutorService to be implicitly taking up TCP connections for inter-thread communications, but this experiment proves that this is not correct and this method call is related to even lower java issues which is out of our reach.

At this point we are running at 20GB/hour on 64 threads with 98% CPU utilization. A multi-processing idea that we had did not work and java was causing troubles at memory management. These changes increased processing speed by 65% from 12-13 GB/hours that we were getting before. Not bad! Some more micro optimizations took us even farther to 22GB per hours of encrypted and compressed thrift objects.

2.5 Back to C++

At the end we developed a C++ version of the system to go through the corpus in batch process, as we didnt have any more time on processing the data, but kept the entity matcher and pipeline in Java for ease of pipeline development as time was tight and we were short on human resources. With C++ it was going at about 74-100 GB/hour on bare string matching but this was not an option for the development of the pipeline and using NLP tools and libraries.

2.6 Other notes

Javas library for decompressing the .xz files performed faster than using the Linux file systems. We believe this is because keeping a 45mb file in memory and moving it is slower than decompressing it in java and streaming into the output to the thrift builder.

2.7 Cumulative Citation Recommentation

2.8 Streaming Slot Filling

2.9 Post Processing Algorithm

3 Implementation

We extract aliases for entities from wikipedia automatically both using API and using the actual page content, then apply pattern matching rules for slot value extraction.

3.1 Alias Generation

We use Wikipedia API to get some aliases automatically. This is done by retrieving backlink references (redirects of a wiki entity). Unfortunately this is not good enough and to enhance recall we need more aliases. To have better use of a wiki page we parse HTML DOM of the page, then use regular expressions to extract the bold phrases of the first paragraph as alias of the actual entity. Based on our observation this is a very accurate heuristic and provides us with lots of famous aliases of the entities. To consider other typical cases we consider some generic first name last name order swapping conventions such as Bill Gates -¿ Gates, Bill.

3.2 Pattern Matching

In this system, we use pattern matching methods to find corresponding slot values for entities. The procedure is: 1) Find stream items that contain the entities by using alias names of the entities. 2) Inside these stream items, fetch the sentences which contain entities by using alias names and the coreference information provided by Lingpipe tags. 3) Use these sentences to match existing patterns. 4) When patterns matched, generate SSF results.

The format of the patterns is consist of five fields: 1) the type of the entity. 2) the slot name. 3) the pattern content. 4) the direction of the slot valule to the entity. 5) the type of the slot values. When we match one pattern, we match all the fields except the third field. The type of slot values could be the entity type tagged by Linpipe, noun phrase tagged by OpenNLP and the time phrases we hard-code. For these three kinds of patterns, we implement them in different ways accordingly.

Next we will explain the patterns with more details. Letss first look at one example of the first kind of patterns. For slot FounderOf, we have this pattern: ¡PER, FounderOf, founder, right, ORG¿. PER means that the entity we are finding slot values for is a PER entity; FounderOf means this is a pattern for FounderOf slot. founder is the ankor word we are trying to match in the sentence text; right means that we are going to the right part of the sentence to match the pattern and find the slot value; ORG means the slot value should be a ORG entity.

Here is an example for the second type: ¡PER, AwardsWon, awarded, right, NP¿, NP meaning noun phrase. This pattern can be interpreted as that we are looking for a noun phrase after the awarded since that noun phrase could possibly represent an award. Since titles and awards are usually not the Lingpipe entities, we use the OpenNLP noun phrase chunker to fetch the noun phrases.

Another example for the third type: ¡PER, DateOfDeath, died, right, last night¿. In this pattern, last night means we are looking for exactly the phrase last night to the right of died. This pattern is inspired by the intuition that in news articles, people often mention that somebody died last night instead of mentioning the accurate date information and Lingpipe tends not to tag phrases like last night as a DATE entity.

To improve recall, we introduce generic patterns that can generate many results. But these patterns generate many errors, too. Thus there is a trade-off between the accuracy and recall. This is one drawback of using pattern matching method, meaning it is really hard to find good patterns with both high accuracy and recall.

For accuracy, we have found that there are three major kinds of errors: 1) wrong entities found; 2) wrong tags by the Lingpipe; 3) wrong results matched by the patterns. We solve the first problem by using the wikipedia or twitter information of the entities to get better alias names. And we use post-processing to reduce the second and third types of errors.

3.3 Post Processing

The SSF output of many extractions is noisy. The data contains duplicates and incorrect extractions. We can define rules to sanitize the output only using the information present in the SSF file. The file is processed in time order, in a tuple-at-a-time fashion to minimize the impact on accuracy. . We define two classes of rules deduplication rules and inference rules.

The output contains many duplicate entries. As we read the list of extrated slots we create rules to define "duplicate". Duplicates can be present in a window of rows; we use a window size of 2 meaning we only be adjacent rows. Two rows are duplicates if 1) they have the same exact extraction, 2) if the rows have the same slot name and a similar slot value and 3) if the extracted sentence for a particular slot types come from the same sentence.

New slots can be deduced from existing slots by defineing inference rules. For example, two slots for the task are "FounderOf" and "FoundedBy". A safe assumtion is these slot names are biconditional logical connectives with the entities and slot values. Therefore, we can express a rule "X FounderOf Y" equals "Y FoundedBy X" where X and Y are single unique entities. Additionally, we found that the slot names "Contact_Meet_PlaceTime" could be infered as "Contact_Meet_Entity" if the Entity was a FAC and the extracted sentence contained an additional ORG/FAC tag.

We also remove erronious slots that have extractions that are several pages in length or tool small. Errors of extracting long sentences can typically be attributed to poor sentence parsing of web documents. We have some valid "small" extractions. For example a comma may separate a name and a title (e.g. "John, Professor at MIT"). But such extraction rules can be particularly noisy, so we check to see if the extracted values have good entity values.

The aim in post processing was to sanitize the results increasing the precision of the results by eliminating spurrious results.

3.4 Shortcomings

The missing twitter entities are more than the 50% of the twitter entities we need to work on and this is a very very bad performance. One reason could be that those entities are not very famous. For example Brenda Weiler google

search result is 860,000 documents our of billions of web documents. For our small portion of the web it might make sense. If you pay attention to the histogram of the entities found you'll note that more than half of the entities have appeared in less than 10 StreamItems. A good portion have appeared only once. Which the document does not necessarily have good information for us.

A theory is that we are falling behind because we are using the cleansed version of the corpus. We believe there has been a reason that TREC has included the actual HTML document as well as the cleansed version. This definitely will convey some information to us. If it was as easy as reading some clean text they wouldn't bother including so much data for teams to be useless. So we guess is that we are missing some information from not using the actual document. And, we are looking for tokens with entity value set which will depend us dramatically on the accuracy of lingpipe, which is a fast algorithm but is not as good as other NLP tools can be e.g. Stanford NLP.

4 Results

Document extraction using query entity matching with aliases, sentence extraction using alias matching and co-reference. Slot extraction using patterns, NER tags and NP tags. 158,052 documents with query entities, 17885 unique extracted slot values for 8 slots and 139 entities, 4 slots and 31 entities missing.

On the performance of the first submission run we performed some random sampling via two processes, the results of which are as follows:

5 Discussion

As mentioned before, when we evaluate the results of slot extraction, we find three kinds of problems for accuracy: 1) wrong entities found; 2) wrong tags by the Lingpipe; 3) wrong results matched by the patterns. We also have recall problems: 1) not enough good alias names to find all the entities. 2) not enough and powerful patterns to capture all the slot values.

We will use entity resolution methods and other advanced methods to improve the accuracy and recall of entity extraction part.

For slot extraction part, in order to improve the performance, what we need to are: 1) Using multi-class classifiers instead of pattern matching method to extract slot values in order to increase both recall and accuracy for slots Affiliate, AssociateOf, FounderOf, EmployeeOf, FoundedBy TopMembers, Contact_Meet_Entity and so on. 2) For special slots, like Titles, DateOfDeath,

Table 8: SSF Performance Measure on Initial Submission

	Correct	Incorrect Entity name	Incorrect Value
Sampling #1	55%	17%	27%
Sampling #2	54%	15%	31%

CauseOfDeath, AwardsWon, using different kind of advanced methods, e.g. classifiers, matching methods. 3) Using other NLP tools or using classifiers to overcome the drawbacks of the LingPipes inaccurate tags. The first and second tasks are the most important tasks we need to do.

6 Conclusions

Acknowledgements

Christan Grant is funded by a National Science Foundation Graduate Research Fellowship under Grant No. DGE-0802270.

References

[1] Apache spark. http://spark.incubator.apache.org/.

Table 9: SSF Performance Measure on Submission Infer

	Correct	Incorrect Entity name	Incorrect Value
Affiliate	%	%	%
AssociateOf	%	%	%
AwardsWon	%	%	%
CauseOfDeath	%	%	%
Contact_Meet_Entity	%	%	%
Contact_Meet_PlaceTime	%	%	%
DateOfDeath	%	%	%
EmployeeOf	%	%	%
FoundedBy	%	%	%
FounderOf	%	%	%
Titles	%	%	%
TopMembers	%	%	%