Using LAPPS Grid Services in Hadoop

Master's Thesis

Presented to

The Faculty of the Graduate School of Arts and Sciences
Brandeis University
Graduate Program in Computational Linguistics
James Pustejovsky, Advisor

In Partial Fulfillment
of the Requirements for the Degree

Master of Arts
in
Computational Linguistics

by
Kelley Lynch

August 2017

ABSTRACT

Using LAPPS Grid Services in Hadoop

A thesis presented to the Graduate Program in Computational Linguistics

Graduate School of Arts and Sciences
Brandeis University
Waltham, Massachusetts

By Kelley Lynch

The Language Application (LAPPS) Grid project has among its goals promoting reusability and interoperability of Natural Language Processing (NLP) tools. LAPPS Grid processing services are NLP tools such as tokenizers and part of speech taggers, that are packaged as web services to be used on the LAPPS Grid. The processing services can be used to develop and evaluate different sequences of tools. Interoperability of the tools simplifies development and makes experimentation with a large variety of tools more accessible.

According to the International Data Corporation's 2014 report, the size of the digital universe is doubling every two years. As the quantity of available data grows, the use of distributed tools for big data processing is becoming more of a necessity. Hadoop has proven to be a useful tool for large-scale processing in NLP. This paper describes the development of a system for using LAPPS Grid services with Hadoop. This system offers some of the benefits of the LAPPS Grid, interoperability and reusability, in an environment that can be scaled for processing of large data sets.

Table of Contents

Introduction **1**

Rationale **3**

Description of the LAPPS Grid **3**

Hadoop for NLP **4**

Description of LAPPS + Hadoop **6**

UI and Pipeline Validation **9**

Testing **10**

Proposed Improvements **14**

Extending Input and Output Format Capabilities 14

Machine Learning on the Cluster 16

UI Additions 16

Conclusion **18**

Bibliography **19**

Introduction

The Language Application (LAPPS) Grid is a tool for development and testing of Natural Language Processing (NLP) pipelines. It allows users to easily test different tools without needing to be concerned with the interoperability of the tools (Ide, 2014). The LAPPS Grid can be used with a "Galaxy" frontend to easily develop and visualize pipelines and process small collections of documents. In order to make the LAPPS framework and tools useful for larger corpora or collections of documents, it is necessary to make the processing services work in a distributed environment. The qualities of the LAPPS Grid that allow interoperability between services, allow the services to be easily modified to be used within Hadoop.

Using the LAPPS Grid services within Hadoop allows a user to process much larger corpora or collections of documents by processing documents in parallel. This is well suited to the LAPPS Grid services because the services are applied independently on each document. In Hadoop, map tasks receive a document or data point as input, process the input, and construct an output without needing to access any of the other documents being processed. Using the LAPPS Grid processing services within Hadoop also introduces the possibility of adding services in addition to the LAPPS services that allow analysis over LAPPS annotated data. One goal of the LAPPS Grid project is to promote service composition and reuse (Ide, 2014). As data becomes more easily accessible and the size of corpora grow, it is necessary that the tools used to process the data are able to be used in a

distributed environment. This project seeks to promote the goals of composition and reuse for the LAPPS Grid project by proposing a system through which LAPPS grid services can be used on a Hadoop cluster.

In order to make the LAPPS/Hadoop system easy to use, this project included developing a web interface for choosing datasets and designing pipelines. The web interface files, data sources, LAPPS processing services, and necessary Hadoop files can be stored within Apache Tomcat, which allows for simple setup and distribution. The interface simplifies pipeline development by providing assistance to the user during the process of choosing processing services.

The use of Hadoop for NLP tasks is not new. However, within the framework of the LAPPS Grid, having the ability to incorporate Hadoop and reuse services in a distributed environment is important for promoting development and use of LAPPS Grid resources. Testing various pipelines with different data sets on different cluster configurations has shown promising results.

Rationale

Description of the LAPPS Grid

Among the goals of the LAPPS Grid project are achieving interoperability between components and encouraging community involvement in the development of services (Ide, 2014). The project makes available various data sets and NLP tools packaged as web services. The tools can be used to experiment with different pipelines on the data sets. Pipelines consist of a series of services, each adding annotations to the text or to annotations that have been added by previous services in the pipeline.

LAPPS grid services take as input either raw text or a data structure called a "LIF" (LAPPS Interchange Format). LIF's are JSON objects containing fields designating the type of the input (discriminator), the raw text of the input (payload), metadata about the text, and a list of objects containing annotations on the raw data. Each service reads either the raw text or the annotated data and applies an "execute" method to the data. If the input to the service is raw data, the service will create a LIF object containing the data and add the annotations resulting from the processing to the LIF object. If the input to the service is already in the LIF format, the annotation resulting from applying the service will be added to the LIF.

```
"@context" : "http://vocab.lappsgrid.org/",
"metadata" : { },
"text" : {
    "@value" : "Some of the strongest critics of our welfare system..." }
"steps" : [ {
    "metadata" : {
          "contains" : {
          "Token" : {
          "producer" : "org.anc.lapps.stanford.SATokenizer:1.4.0",
          "type" : "stanford"
        }
      }
    },
    "annotations" : [ {
      "@type" : "Token",
      "id" : "tok0",
      "start" : 18,
      "end" : 22,
      "features" : {
        string" : "Some" }
    },
```

Figure 1: Sample LIF[1]

Services in a LAPPS grid pipeline add annotations that services later in the pipeline can use to add their own annotations. There are various types of annotations that are standardized across services and defined by the LAPPS Web Service Exchange Vocabulary. For example, both the Stanford Tokenizer and OpenNLP tokenizer services add "token" annotations to the LIF. The object added to the list of annotations contains metadata specifying the service that produced the annotation. To services later in the pipeline that require "token" annotations, it is only necessary that the annotation have the correct type and it does not matter which service provided the annotation. This standardization, along with the LIF format, allows the tools to be interoperable.

Hadoop for NLP

Hadoop is a framework based on Google's MapReduce Programming Model. Many companies such as Facebook and Yahoo have used Hadoop for various Big Data tasks

including those involving Natural Language Processing. Hadoop jobs typically consist of a map task and a reduce task. The map task processes a specified portion of the data, such as lines of a document, and produces an intermediate result with a key and value. The reduce task combines intermediate results sharing the same key. For purposes of this project, only "map" tasks are used.

A cluster running Hadoop typically consists of multiple nodes. A master node allocates work and monitors multiple worker nodes that execute the map and reduce operations. Testing Hadoop with GATE NLP tools processing web data has shown nearly linear scale-up in relation to the number of worker nodes (Nesi, 2015). The pipeline tested is comparable to a typical LAPPS pipeline and consisted of sentence splitting and tokenization, POS-tagging, and JAPE annotation rules.

Description of LAPPS + Hadoop

Machine learning tasks in NLP require very large datasets and the functionally of the LAPPS grid alone does not make the creation of those datasets feasible. The LAPPS grid allows users to test and evaluate pipelines, but when the optimal pipeline has been found, it is still necessary to pre-process and post-process data to allow the tools in the pipeline to be used consecutively on larger data sets. By integrating the LAPPS grid services with Hadoop, it is possible to generate large annotated data sets with the LAPPS grid tools. This also increases value of developing LAPPS grid components, because they can be used for testing within the LAPPS grid and for large scale automated annotation on Hadoop.

In order to allow the processing services to be used in Hadoop, it was necessary to define the prerequisite java classes to be used in the framework. The Whole File Input Format (White, 2011) uses an entire file as the input to each map task. This format was well suited for this project, because many corpora are structured as text files within a directory structure. Each file in a dataset is treated as document and will be converted to one LIF. After the completion of each map task, each LIF is stored as a document within the Hadoop Distributed File System (HDFS). From there it can be used as input to the next map task in the pipeline.

In the LAPPS grid, each service is packaged as a web service and registered within a service manager. In order to be used within Hadoop, the services needed to be packaged within jar files as opposed to the war web services files. This process required removing

web service dependencies from the project's pom file and changing the package attribute of the pom file from "war" to "jar". This process often introduced problems and some of the services being used within the LAPPS grid could not be converted to jar files that could be used on Hadoop without making multiple, often unintuitive changes to the pom.xml file. As more services were converted, it was necessary to maintain a list of incompatible plugins and dependencies that needed to be removed before the jar file created could be used by Hadoop. In the future, in order to minimize storage requirements, it will be beneficial to remove all plugins and dependencies that are only present in the pom file for web application purposes and not for the processing functionality of the service.

In order to use the processing services classes within the jar files, I wrote a Hadoop class to wrap the processing service as a Hadoop map job. The class takes the desired services java class name as a command line input then constructs an object of that class. This was made possible by the interoperability of the LAPPS components. All LAPPS components implement a Processing Service class that requires an “execute” method. The class passed as a command line argument must implement the LAPPS Processing Service class. The wrapper applies the class’s “execute” method to the input and writes the result as a file in the HDFS.

Each consecutive task in a LAPPS/Hadoop pipeline starts a Hadoop job. When the system is started, a script runs that moves all of the data sources to HDFS. When a pipeline runs, the first job in the pipeline reads the specified input from the collection of data sources. The job writes its output to HDFS where it can be used as input for the next job in the pipeline. After the last job is completed, the file is moved from HDFS to the Tomcat

server and compressed as a tar.gz file. The compressed file can then be downloaded by the user.

UI and Pipeline Validation

The user interface (Figure 2) for the LAPPS/Hadoop system was designed using Angular JS. When initially accessing the UI, it displays a box with a single tab. This box contains a drop-down selector for choosing a data source. At the bottom of the web page is a selector that displays available Processing Services. When the user selects and adds a Processing Service, the service will be added as a tab on the box. The contents of the box will display the annotation and format requirements for the Processing Service and the format that the service produces and the annotations that the service adds. If the service is not compatible with the current pipeline, an alert is displayed that explains the reason for the incompatibility and lists Processing Services that would resolve the incompatibility

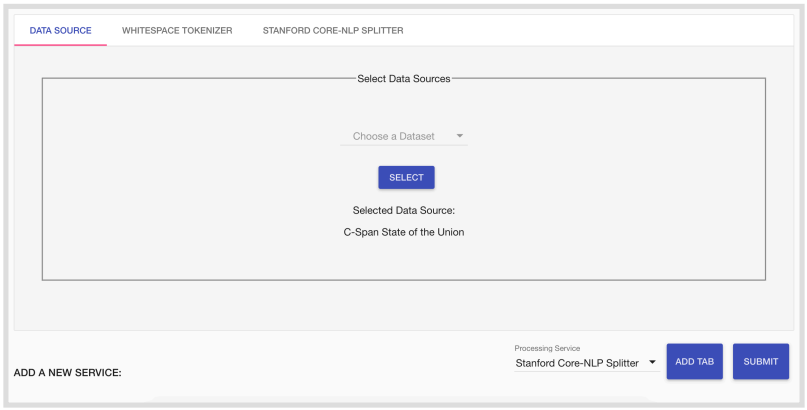

Figure 2: User Interface

The user interface for constructing a LAPPS/Hadoop pipeline was an important component of this project because it allows a user to avoid the complexity of constructing complex Hadoop commands and makes it possible to restrict pipelines to those only consisting of compatible services. Each processing service in a LAPPS pipeline requires the input to be in a particular format and may require specific annotations to be present. For example, the OpenNLP Parser requires a LIF input with sentence annotations. The UI uses the metadata about each service to confirm that the input passed to each service in the pipeline will contain the annotations that that service requires. If a user attempts to select a processing service, but adding that service would result in an invalid pipeline, an alert is displayed explaining which annotation is missing and listing services that provide that annotation. The user can add any of the services providing the necessary annotation then add the originally requested service.

In addition to requiring particular annotations, services also require that the input be in a particular format. Typically, services require the input to be raw text or in the LIF format. The services contain metadata specifying which input types they accept and what output type they produce. Each of the data sources contains metadata specifying these types. This metadata is used to verify that each service receives input in an accepted format. If the user attempts to construct a pipeline that is invalid due to input format requirements, a message is displayed describing the error and suggesting alternative compatible services.

Testing

I tested the system using Amazon Web Services (AWS). Data sources and processing services were stored on Amazon's s3 cloud storage service. The Tomcat server hosting the

UI ran on an EC2 instance. During development, I had tested the system with a local installation of Hadoop, so the scripts used to start Hadoop jobs and interact with the filesystem had to be modified to use commands from the AWS command line interface. I tested various pipelines on three datasets. The first dataset, input_test, was used to validate that the scripts were working correctly. This dataset consisted of 4 identical copies of a 900-byte news article. The C-Span Corpus contained transcripts of 67 state of the union addresses totaling 2.2 MB. The largest dataset tested was the "Trip Advisor" dataset which contained 1760 reviews from the Trip Advisor website, totaling 258.6 MB.

Various pipelines were tested throughout the development process. Not all pipelines tested successfully completed on the cluster. For jobs on small datasets, no improvement was seen by varying the cluster size. This is due to the fact that a certain amount of time will always be necessary to initiate a Hadoop job. For larger jobs, the amount of time for each step in the pipeline scales as expected with the number of nodes being used.

Tables 1 - 4 show the results of tests on the input_test, C-Span, and Trip Advisor datasets. The pipeline tested consisted of three steps. The first step, "Whitespace Tokenizer", converted the raw text to the LIF format and added token annotations where the tokens were found by splitting the text on whitespace. The remaining two steps used tools from the Stanford CoreNLP library. These steps were part-of-speech tagging and dependency parsing. The configurations all used clusters of Amazon's m4.large instances. I performed tests with 3, 10, and 19 nodes. With each configuration, 1 node was the Master Node and the remaining nodes were worker nodes. The tables show the total amount of time required for running the job on the cluster. The results show for the 10 and 19 node

cluster configurations show nearly linear scaling. Unfortunately, only the Whitespace Tokenizer completed processing on the Trip Advisor dataset.

	3 nodes	10 nodes	19 nodes
input_test	130 seconds	130 seconds	130 seconds
C-Span	FAILED	24 minutes	12 minutes
Trip Advisor	FAILED	FAILED	FAILED

Table 1: Total Processing Time

	Whitespace Tokenizer	Core NLP POS Tagger	Core NLP Dependency Parser
input_test	38 seconds	46 seconds	46 seconds
C-Span	3 minutes	7 minutes	FAILED
Trip Advisor	20 minutes	FAILED	----------

Table 2: 3-Node Cluster Processing Time

	Whitespace Tokenizer	Core NLP POS Tagger	Core NLP Dependency Parser
input_test	38 seconds	46 seconds	46 seconds
C-Span	1 minute	2 minutes	21 minutes
Trip Advisor	5 minutes	FAILED	----------

Table 3: 10-Node Cluster Processing Time

	Whitespace Tokenizer	Core NLP POS Tagger	Core NLP Dependency Parser
input_test	38 seconds	46 seconds	46 seconds
C-Span	56 seconds	1 minute	10 minutes
Trip Advisor	2 minutes	FAILED	FAILED

Table 4: 19-Node Cluster Processing Time

Proposed Improvements

Currently, adding data sources or processing services to the LAPPs/Hadoop project requires manually adding the data source files or processing service jar file to the server then adding the necessary metadata to the xml files that manage data source and processing service information. In order to simplify the process of adding these components, I would like to add a page to the project that would allow a user to register new components, this would reduce the possibility of user error in adding new services. In addition, the registration process could simplify adding new services by automatically extracting metadata about the processes or data being added. Processing Services contain metadata specifying the format and annotation requirements and the format and annotations that the service produces. This metadata is necessary in order for the UI to validate pipelines. Having the registration service automatically extract this metadata would simplify the process and ensure that pipeline validation functions correctly.

Extending Input and Output Format Capabilities

In order to allow the system to work with a larger variety of corpora, I would like to define additional input formats that would allow data other than raw text files to be used. For example, a custom input class could read data structured in the CONLL format and convert it to the LIF format. Map tasks could also be used to convert LIF's containing annotations to other formats necessary for machine learning tasks. For example, the Mallet

machine learning library requires that input data be structured so that each line of a file contains an individual token and its annotations. This preprocessing of data for machine learning tasks could easily be done within a Hadoop job. Being able to specify a particular output format would streamline the process of using the LAPPS/Hadoop system for creating annotated data for machine learning.

Testing and Evaluation

Throughout this project, the focus has been on transforming a dataset and increasing its utility by providing annotations using the "Map" step of MapReduce. Using this framework is effectively applying complex pre-processing to the data. The large scale capabilities of the system make it valuable to provide testing and evaluation services within the framework. In the future, it would be useful to add additional services that analyze the dataset that results from processing a dataset through a LAPPS pipeline. Adding these types of services will benefit from the interoperability of the system. Evaluation services are already available through the LAPPS Grid, however they cannot be directly used with Hadoop in the same way as the other processing services. One type of service that would be particularly valuable is comparing the result of a pipeline to a gold standard annotated dataset. This type of service would allow a user to evaluate the performance of various tools on a dataset. This type of evaluation is well suited to the machine learning and analysis capabilities of Mahout.

Machine Learning on the Cluster

Annotated data is used for machine learning tasks. Because the annotated data generated by the LAPPS/Hadoop system resides on the HDFS, it is possible to use the machine learning tools included in Mahout on the data. Mahout is an open source project designed to be used with Hadoop and to scale in a distributed environment. Amazon Elastic MapReduce clusters come with Mahout installed by default. Mahout is used by various companies such as Facebook and Yahoo for large scale machine learning tasks.

The primary hindrance to implementing processing services that use Mahout is that many machine learning tasks are less straightforward than automated annotation. It may be too complex to provide services for supervised machine learning tasks, however it may be possible to make use of some of the unsupervised machine learning libraries in Mahout. In addition, it could be possible to allow users to upload their own custom machine learning tools that use Mahout.

UI Additions

In order to make the LAPPS/Hadoop user interface more useful. It would be helpful to include monitoring of the currently running jobs. Currently, when a job is running, the web page appears to be loading the next page. Because jobs take at least a few minutes, it would be helpful to be able to monitor the progress of a job as it is running. The user interface for job monitoring should display each task in the pipeline and indicate when the task has been completed. This type of monitoring is already being done in the script being used to run jobs on the AWS EMR cluster. In order for the script to "wait" before sending consecutive commands, the script requests the status of the running job every second. When the job is no longer in the "pending" state or "running" state, the next command is

issued. With an updated user interface, the script could also inform the user when each task in the pipeline has been completed. If a task has failed, a status request to the cluster contains information about the reason for the failure. An updated user interface could display information about the error and potentially recommend a solution if possible.

An issue with the current user interface is that it requires the user to leave the web page open until the job is completed. If the user closes the web page, the job will complete and the data will be available on the server, but the user will be unable to retrieve the data without accessing the file system on the server. It is not always feasible for a user to keep the web page available and users should not be limited to downloading the result of a job on the device from which the job was issued. In order to make it possible to download the results of a job after the web page has been closed or on a device other than the device a job was issued from, it is necessary to add an additional web page that displays information about the results currently available on the server and allows those results to be downloaded.

Conclusion

The results of testing show that using the LAPPS/Hadoop system scales effectively for large scale data-processing. There are still many improvements and enhancements to be made to the system. The results of experimentation and development show that it is possible to use LAPPS services within Hadoop by only modifying the Processing Service project's pom.xml file. Given the consistency of these files, it is likely possible to programmatically modify the files, so that no additional development work is needed to use LAPPS services in Hadoop.

The LAPPS/Hadoop system builds on the framework developed by the LAPPS Grid project, and expands the utility of services in that project. The near linear scale up of the system with larger cluster sizes is consistent with prior research on NLP with Hadoop. Further testing with larger cluster sizes is necessary in order to demonstrate the value of the system for very large datasets.

Bibliography

Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." *Communications of the ACM*. ACM, Jan. 2008. Web. 18 May 2017.

Ide, N., Pustejovsky, J., Cieri, C., Nyberg, E., Wang, D., Suderman, K., Verhagen, M., Wright, J.: The language application grid. In: Proceedings of the Ninth International Conference on Language Resources and Evaluation LREC 2014. European Language Resources Association (ELRA), Reykjavik, Iceland, May 2014.

Ide, N., Suderman, K., Pustejovsky, J., Verhagen, M. Cieri, C.: "The Language Application Grid and Galaxy." LREC 2016, Portoroz, Slovenia. May 23-28, 2016.

Nesi, Paolo, Gianni Pantaleo, and Gianmarco Sanesi. "A Hadoop Based Platform for Natural Language Processing of Web Pages and Documents." *Journal of Visual Languages & Computing* 31 (2015): 130-38. Web.

Ide, Nancy, Keith Suderman, Marc Verhagen, and James Pustejovsky. "The Language Application Grid Web Service Exchange Vocabulary." *Worldwide Language Service Infrastructure Lecture Notes in Computer Science* (2016): 18-32. Web.

Nesi, Paolo, Gianni Pantaleo, and Gianmarco Sanesi. "A Hadoop Based Platform for Natural Language Processing of Web Pages and Documents." *Journal of Visual Languages & Computing* 31 (2015): 130-38. Web.

Turner, V., Gantz, J.F., Reinsel, D., Minton, S., The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, IDC White Paper, 2014.

Verhagen, M. , Suderman, K., Wang, D., Ide, N., Shi, C., Wright, J., and Pustejovsky, J. 2016. The LAPPS Interchange Format. In Revised Selected Papers of the Second International Workshop on Worldwide Language Service Infrastructure - Volume 9442, WLSI 2015, pages 33–47, New York, NY, USA. Springer-Verlag New York, Inc.

White, Tom. *Hadoop: The Definitive Guide*. 2nd Edition: O'Reilly Media, 2011. Print.