ABSTRACT

Many companies keep large amounts of text data inside of relational databases. Several challenges exist in using state-of-the-art systems to perform analysis on such datasets. First, expensive big data transfer cost must be paid up front to move data between databases and analytics systems. Second, many popular text analytics packages do not scale up to production sized datasets. In this paper, we introduce GPText, Greenplum parallel statistical text analysis framework that addresses the above problems by supporting statistical inference and learning algorithms natively in a massively parallel processing database system. GPText seamlessly integrates the Solr search engine and applies statistical algorithms such as k-means and LDA using MADLib, an open source library for scalable in-database analytics which can be installed on PostgreSQL and Greenplum. In addition, GPText also developed and contributed a linear-chain conditional random field (CRF) module to MADLib to enable information extraction tasks such as part-of-speech tagging and named entity recognition. We show the performance and scalability of the parallel CRF implementation. Finally, we describe an eDiscovery application built on the GPText framework.

Categories and Subject Descriptors
H.2.4 [Database Management]: [Systems, parallel databases]; I.2.7 [Artificial Intelligence]: [Natural Language Processing, text analysis]

General Terms
Design, Performance, Management

Keywords
RDBMS, Massive parallel processing, Text analytics

1. INTRODUCTION

Text analytics has gained much attention in the big data research community due to the large amounts of text data generated in organizations such as companies, government and hospitals everyday in the form of emails, electronic notes and internal documents. Many companies store this text data in relational databases because they rely on databases for their daily business needs. A good understanding of this unstructured text data is crucial for companies to make business decisions, for doctors to assess their patients, and for lawyers to accelerate document review processes.

Traditional business intelligence pulls content from databases into other massive data warehouses to analyze the data. The typical “data movement process” involves moving information from the database for analysis using external tools and storing the final product back into the database. This movement process is time consuming and prohibitive for interactive analytics. Minimizing the movement of data is a huge incentive for businesses and researchers. One way to achieve this is for the datastore to be in the same location as the analytic engine.

While Hadoop has become a popular platform to perform large-scale data analytics, newer parallel processing relational databases can also leverage more nodes and cores to handle large-scale datasets. Greenplum database, built upon the open source database PostgreSQL, is a parallel database that adopt a shared nothing massively parallel processing (MPP) architecture. Database researchers and vendors are capitalizing on the increase in database cores and nodes and investing in open source data analytics ventures such as the MADLib project [3, 7]. MADLib is an open source library for scalable in-database analytics on Greenplum and PostgreSQL. It provides parallel implementation of many machine learning algorithms.

In this paper, we motivate in-database text analytics by showing the GPText, a powerful and scalable text analysis framework developed on Greenplum MPP database. The GPText inherits scalable indexing, keyword search, and faceted search functionalities from an effective integration of the Solr [6] search engine. The GPText uses the CRF package, which was contributed to the MADLib open-source library. We show that we can use SQL and user defined aggregates to implement conditional random fields (CRFs) meth-
ods for information extraction in parallel. The experiment shows sublinear improvement in runtime for both CRF learning and inference with linear increase in the number of cores. As far as we know, GPText is the first toolkit for statistical text analysis in relational database management systems. Finally, we describe the need and requirement for eDiscovery applications and show that GPText is an effective platform to develop such sophisticated text analysis applications.

2. RELATED WORK

Researchers have created systems for large scale text analytics including GATE, PurpleSox and SystemT [4, 2, 8]. While both GATE and SystemT uses rule-based approach for information extraction (IE), PurpleSox uses statistical IE models. However, none of the above system are native built for a MPP framework.

The parallel CRF in-database implementation follows the MAD methodology [7]. In a similar vein, researchers show that most of the machine learning algorithms can be expressed through a unified RDBMS architectures [5].

There are several implementations conditional random fields and but only a few large scale implementations for NLP tasks. One example is the CRFs [11] that are implemented over massively parallel processing systems supporting Message Passing Interface (MPI) such as Cray XT3, SGI Altix, and IBM SP. However, this is not implemented over RDBMs.

3. GREENPLUM TEXT ANALYTICS

GPText runs on Greenplum database (GP), which is a shared nothing massive parallel processing (MPP) database. The Greenplum database adopts the most widely used master-slave parallel computing model where one master node orchestrates multiple slaves to work until the task is completed and there is no direct communication between slaves. As shown in Figure 1, it is a collection of PostgreSQL instances including one master instance and multiple slave instances (segments). The master node accepts SQL queries from clients, then divide the workloads and send sub-tasks to the segments. Besides harnessing the power of a collection of computer nodes, each node can also be configured with multiple segments to fully utilize the multicore processor. To provide high availability, GP provides the options to deploy redundant standby master and mirroring segments in case of master segments and primary segments failure.

The embarrassing processing capability powered by the Greenplum MPP framework lays the cornerstone to enable GPText to process the production sized text data. Besides the underlying MPP framework, GPText also inherited the features that a traditional relational database management system brings to GPText for example, online expansion, data recovery and performance monitoring to name a few. On top of the underlying MPP framework, there are two building blocks, MADLib and Solr as illustrated in Figure 1 which distinguish GPText from many of the existing text analysis tools. MADLib makes GPText capable of doing sophisticated text data analysis tasks, such as part-of-speech tagging, named entity recognition, document classification and topic modeling with a vast amount of parallelism. Solr is reliable and scalable text search platform from Apache Lucene project and it has been widely deployed in web servers. The major features include powerful full-text search, faceted search, near real time indexing. As shown in Figure 1, GPText uses Solr to create distributed indexing. Each primary segment is associated with one and only Solr instance where the index of the data in the primary segment is stored for the purpose of load balancing. GPText has all the features that Solr has since Solr is integrated into GPText seamlessly. The authors submitted the CRF statistical package for text analysis to MADLib (more details on section 4).

3.1 In-database document representation

In GPText, a document can represented as a vector of counts against a token dictionary, which is a vector of unique tokens in the dataset. For efficient storage and memory, GPText uses a sparse vector representation for each document instead of a naive vector representation. The following are an example with two different vector representations of a document. The dictionary contains all the unique terms (i.e., 1-grams) exist in the corpus.

**Dictionary:**
{am, before, being, bothered, corpus, document, in, is, me, never, now, one, really, second, the, third, this, until}.

**Document:**
{1, am, second, document, in, the, corpus}.

**Naive vector representation:**
{1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1}.

**Sparse vector representation:**
{{1, 3, 1, 1, 1, 6, 1, 1}: {1, 0, 1, 1, 1, 1, 0, 1, 1}}.

GPText adopts the run-length encoding to compress the naive vector representation using pairs of count-value vectors. The first array in the vector representation contains count values for the corresponding values in the second array. Although not apparent in this example, the advantage of the sparse vector representation is dramatic in real world documents, where most of the elements in the vector are 0.

3.2 ML-based Advanced Text Analysis

GPText relies on multiple machine learning modules in MADLib to perform statistical text analysis. Three of the commonly used modules are k-means for document clustering, multinomial naive bayes (multinomialNB) for document classification, and latent dirichlet allocation (LDA) for topic modeling for dimensionality reduc-
4. CRF FOR IE OVER MPP DATABASES

Conditional random field (CRF) is a type of discriminative undirected probabilistic graphical model. Linear-chain CRFs are special CRFs, which assume that the next state depends only on the current state. Linear-chain CRFs achieve start-of-the-art accuracy in many real-world natural language processing (NLP) tasks such as part of speech tagging (POS) and named entity recognition (NER).

4.1 Implementation Overview

In Figure 2 illustrate the detailed implementation of the CRF module developed for IE tasks in GPText. The top box shows the pipeline of the training phase. The bottom box shows the pipeline for the inference phase. We use declarative SQL statements to extract all features from text. Any features in the state of art packages can be extracted using one single SQL clause. All of the common features described in literature can be extracted with one SQL statement. The extracted features are stored in a relation for each document. For feature extraction, we use user-defined functions to calculate the maximum-a-priori (MAP) configuration and probability for each document. Inference over all documents depends with other documents. For learning, the computation of gradient and log-likelihood over all training documents dominates the overall training time. Fortunately, it can run parallel since the overall gradient and log-likelihood is simply the summation of the gradient and log-likelihood for each document. This parallelization can be expressed with the parallel programming paradigm, user-defined aggregates which is supported in parallel databases.

4.2 Feature Extraction Using SQLs

Text feature extraction is a step in most statistical text analysis methods. We are able to implement all of the seven types of features used in POS and NER using exactly seven SQL statements. These features include:

- **Dictionary**: Does this token exist in a dictionary?
- **Regex**: Does this token match a regular expression?
- **Edge**: Is the label of a token correlated with the label of a previous token?
- **Word**: Does this token appear in the training data?
- **Unknown**: Does this token appear in the training data below certain threshold?
- **Start/End**: Is this token first/last in the token sequence?

There are many advantages for extracting features using SQLs. The SQL statements hide a lot of the complexity present in the actual operation. It turns out that each type of feature can be extracted out using exactly one SQL statement, making the feature extraction code extremely succinct. Secondly, SQL statements are naively parallel due to the set semantics supported by relational DBMS’s. For example, we compute features for each distinct token and avoid re-computing the features for repeated tokens.

In Figure 3 and Figure 4 we show how to extract edge and regex features, respectively. Figure 3 extracts adjacent labels from sentences and stores them in an array. Figure 4 shows a query that selects all the sentences that satisfies any of the regular expressions present in the table ‘regextext’.

4.3 Parallel Linear-chain CRF Training

```plaintext
Algorithm 1 CRF training(z_{1:M})

Input: z_{1:M}, \triangleright Document set
Convergence(),
Initialization(),
Transition(),
Finalization()

Output: Coefficients w \in \mathbb{R}^N

Initialization/Precondition: iteration = 0
1: Initialization(state)
2: repeat
3: state \leftarrow LoadState()
4: for all m \in 1..M do
5: state \leftarrow Transition(state, z_m)
6: state \leftarrow Finalization(state)
7: WriteState(state)
8: until Convergence(state, iteration)
9: return state.w
```

Figure 2: The MADLib CRF overall system architecture.

Figure 3: Query for extracting edge features

```sql
SELECT doc2.pos, doc2.doc_id, 'E.', ARRAY[doc1.label, doc2.label]
FROM segmenttbl doc1, segmenttbl doc2
WHERE doc1.doc_id = doc2.doc_id AND
    doc1.pos + 1 = doc2.pos
```

Figure 4: Query for extracting regex features

```sql
SELECT start_pos, doc_id, 'R_' || r.name, ARRAY[-1, label]
FROM regextbl r, segmenttbl s
WHERE s.seg_text \sim r.pattern
```

Figure 3: Query for extracting edge features

```sql
SELECT start_pos, doc_id, 'R_' || r.name, ARRAY[-1, label]
FROM regextbl r, segmenttbl s
WHERE s.seg_text \sim r.pattern
```
Programming Model.

In Algorithm 1 we show the parallel CRF training strategy. The algorithm is expressed as an user-defined aggregate. User-defined aggregates are composed of three parts: a transition function (Algorithms 2), a merge function1, and a finalization function (Algorithm 3). Following we describe these functions.

In line 1 of Algorithm 1 the Initialization function creates a ‘state’ object in the database. This object contains a coefficient \(w\), gradient \(\nabla\) and log-likelihood \(L\) variables. This state is loaded (line 3) and saved (line 7) between iterations. We compute the gradient and log-likelihood of each segment in parallel (line 4) much like a Map function. Then line 6 computes the new coefficients much like a reduce function.

Transition strategies.

Algorithm 2 contains the logic of computing the gradient and log-likelihood for each tuple using the forward-backward algorithm. The overall gradient and log-likelihood is simply the summation of gradient and log-likelihood of each tuple. This algorithm is invoked in parallel over many segments and the result of these functions are combined using the merge function.

Algorithm 2 transition-lbfgs(state, \(z_m\))

<table>
<thead>
<tr>
<th>Input:</th>
<th>state, (z_m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>state</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{Input:} & \quad \text{state, } z_m, \quad \text{A Document} \\
\text{Gradient}() & \quad \text{state} \\
\text{Output:} & \quad \text{state} \\
1: & \quad \{\text{state}, \nabla, \text{state}, L\} \leftarrow \text{Gradient}(\text{state}, z_m) \\
2: & \quad \text{state.num_rows} \leftarrow \text{state.num_rows} + 1 \\
3: & \quad \text{return state}
\end{align*}
\]

Finalization strategy.

The finalization function invokes the L-BFGS convex solver to get a new coefficient vector. To avoid overfitting, we choose to penalize the log-likelihood with a spherical Gaussian weight prior.

Limited-memory BFGS (L-BFGS), a variation of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is a leading method for large scale non-constraint convex optimization method. We translate an in-memory Java implementation [10] to a C++ in-database implementation. Before each iteration of L-BFGS optimization, we need to initialize the L-BFGS with the current state object. At the end of each iteration, we need to dump the updated variables to the database state for next iteration.

4.4 Parallel Linear-chain CRF Inference

The Viterbi algorithm is applied to find the top-k most likely labelings of a document for CRF models. The CRF inference is naively parallel since the inference over one document is independent of other documents. We chose to implement a SQL clause to drive the Viterbi inference over all documents and each function call will finish labeling of one document. In parallel databases, e.g., Greenplum, Viterbi can be run in parallel over different subsets of the documents on a single-node or multi-node cluster and each node can have multiple cores.

5. EXPERIMENTS AND RESULTS

In order to evaluate the performance and scalability of the linear-chain CRF learning and inference on Greenplum, we conduct experiments on various data sizes over on a 32-core machine with 2T hard drive and 64GB memory. We use CoNLL2000 dataset containing 8936 tagged sentences for learning. This dataset is labeled with 45 POS tags. To evaluate the inference performance, we extracted 1.2 million sentences from the New York Times dataset. In Figure 5 and Figure 6 we show the algorithm is sublinear and improves with an increase in the number of segments. Our POS implementation achieves .9715 accuracy, this is consistent with the state of the art [9].

6. GPTEXT APPLICATION

With the support of Greenplum MPP data processing framework, efficient Solr indexing/search engine, and parallelized statistical text analysis modules, GPText positions itself to be a great platform for many applications that need to apply scalable text analytics methods with varying sophistications over unstructured data in databases. One of such application is eDiscovery.

eDiscovery has become increasingly important in legal processes. Large enterprises keep terabytes of text data such as emails and internal documents in their databases. Traditional civil litigation often involves reviews of large amounts of documents both in the plaintiffs side and defendants side. eDiscovery provides tools to
pre-filter documents for review to provide a more speedy, inexpensive and accurate solution. Traditionally, simple keyword search are used for such document retrieval tasks in eDiscovery. Recently, predictive coding based on more sophisticated statistical text analysis methods are gaining more and more attention in civil litigation since it provides higher precision and recall in retrieval. In 2012, judges in several cases [1] approves the use of the predictive coding based on the indication of Rules of the Federal Rules of Civil Procedure. GPText developed a prototype of eDiscovery tool using the Enron dataset. Figure 7 is a snapshot of the eDiscovery application. In the top pane, keyword ‘illegal’ is specified and Solr is used to retrieve all relevant emails that contains the keyword displayed in the bottom pane. As shown on the middle pane on the right, it also supports topic discovery in the email corpus using LDA and classification using k-means. K-means use LDA and CRF to reduce the dimensionalities of features to speed up the k-means classification. CRF is also used to extracted named entities. As you can see the topics are displayed in the Results panel. The tool also provides visualization of aggregate information and faceted search over the email corpus.

7. CONCLUSION
We introduce GPText, a parallel statistical text analysis framework over MPP database. With the seamless integration with Solr and MADLib, GPText is a framework with powerful search engine and advanced statistical text analysis capabilities. We implemented a parallel CRF inference and training module for IE. The functionalities and scalability provided by GPText positions itself to be a great tools for sophisticated text analytics applications such as eDiscovery. As future work, we plan to support learning to rank and active learning algorithms in GPText.

Acknowledgments
Christian Grant is funded by NSF under Grant No. DGE-0802270. This work was supported by a gift from EMC Greenplum.

8. REFERENCES

Figure 7: GPText application