MapDupReducer: Detecting Near Duplicates over Massive Datasets *

Chaokun Wang[†] Jianmin Wang[†] Xuemin Lin[‡] Wei Wang[‡] Haixun Wang[♯]
Hongsong Li[♯] Wanpeng Tian[†] Jun Xu[$†] Rui Li[†]
[†]School of Software, Tsinghua University
Key Laboratory for Information System Security, Ministry of Education
Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
[‡]School of Computer Science and Engineering, University of New South Wales & NICTA, Sydney, Australia
[♯]Microsoft Research Asia, Beijing, China
[$]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[†]{chaokun, jimwang}@tsinghua.edu.cn, [‡]{lxue, weiw}@cse.unsw.edu.au
[♯]{haixunw, songhli}@microsoft.com, [†]{twp07, [$]xujun05, r-li09}@mails.tsinghua.edu.cn

ABSTRACT

Near duplicate detection benefits many applications, e.g., on-line news selection over the Web by keyword search. The purpose of this demo is to show the design and implementation of MapDupReducer, a MapReduce based system capable of detecting near duplicates over massive datasets efficiently.

Categories and Subject Descriptors

H.3.4 [**Information Storage And Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Design, Experimentation

Keywords

MapReduce, near duplicate detection

1. INTRODUCTION

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be a document set where each x_i is a document. The task of **Near Duplicate Detection** (NDD) is to find all of the pairs (x_i, x_j) such that the similarity between x_i and x_j is no smaller than a given threshold, which is typically close to 1.

There are many causes for the existence of near duplicate data: typographical errors, versioned, mirrored, or plagiarized documents, emails generated from the same template, etc. Identifying near duplicate objects benefits many applications. One example is on-line news browsing through keyword search. It is a waste of time, patience, and energy to read similar news. Another example is on-line citation tracking, where citations to the same paper may be treated as citations to different papers due to formatting and/or typographical errors.

With the development of multimedia, storage, and network technology, more and more massive data sets are emerging. It is of great importance to be able to detect near duplicates in massive datasets effectively and efficiently. For instance, Microsoft's BING search engine indexes over 20 billion Web documents; the web documents and their related indices are dispersed over more than 10,000 machines around the globe; and the size of the whole dataset is well over 300 TB. Managing and processing such large amount of data is extremely challenging. But web data typically contains many near duplicates due to a myriad of reasons. The cost of a search engine is closely related to the size of the document set to be indexed. If the size of the Web document set to be indexed can be reduced by detecting near duplicate copies, the indexing costs — the main part of the cost of the search engine — goes down as well.

Clearly, for the above application, the ability to efficiently detecting near duplicates is critical. For example, in modern Web search engines, periodic NDD may greatly reduce the indexing cost if NDD can be conducted efficiently enough. Furthermore, in a hotspot dataset, where a set of documents are updated frequently (e.g., a set of news Web pages), the freshest document set Δ_i to be inserted is replaced rapidly. If near duplicates Δ_i could be detected efficiently each time, the cost of indexing on the whole hotspot dataset can be substantially reduced.

Although many efficient techniques for NDD have been proposed (See [2] for a recent survey), it is clear that a single processor does not have enough computation power to support NDD over a massive dataset efficiently due to physical limits on the processing speed and memory capac-

*Chaokun Wang is supported by the National Natural Science Foundation of China (No. 60803016), the National Basic Research Program of China (No. 2007CB310802) and the National High Technology Research and Development Program of China (No. 2008AA042301). Xuemin Lin is supported by ARC Grants (DP0987557 and DP0881035) and Google Research Award. Wei Wang is supported by ARC Discovery Grants DP0987273 and DP0881779.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

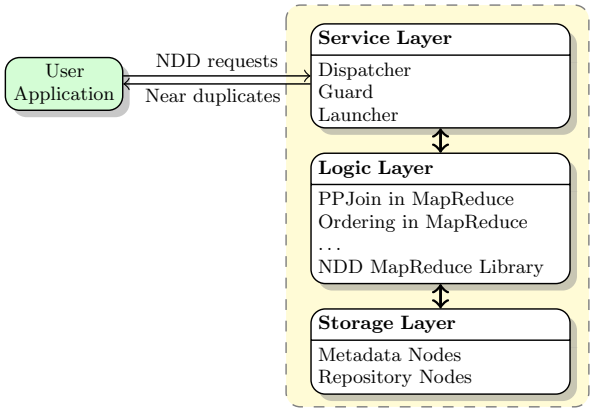

Figure 1: The architecture of MapDupReducer

ity. Therefore, NDD algorithms that can leverage multiprocessor systems are highly desirable. In this paper, we present a MapReduce-based system, MapDupReducer, developed for supporting NDD efficiently over massive datasets. We chose the MapReduce framework as it is a mature platform with distinctive features such as ease of use and high fault-tolerance [1], and it has been widely adopted in the industry. The technical contribution in this demo is the non-trivial generalization of our PPJoin algorithm [5] into the MapReduce framework.

2. SYSTEM OVERVIEW

In this section, we describe the proposed system MapDupReducer. First, we give the architecture of MapDupReducer. Then, we present the technical support, especially the PPJoin paradigm in MapReduce.

2.1 System Architecture

MapReduce is a programming model that enables easy development of scalable parallel applications to process vast amounts of data on large clusters of commodity machines [1]. The system MapDupReducer is implemented on top of the open-source implementation of the MapReduce framework in Hadoop[1]. MapDupReducer consists of two parts: the user application on the client side and the server engine on the server side. The interface of the user application will be presented in the next section with demonstration descriptions. As shown in Figure 1, the server engine of MapDupReducer consists of three layers, i.e., the storage, logic, and service layers. They are described in detail below.

In the service layer, the functionality of a dispatcher includes network monitoring and load distribution. The dispatcher listens on a certain port as a usual Web server does. When a request arrives, the dispatcher establishes proper connections between the server engine and the user application. Also, the dispatcher can perform intelligent load balancing when the number of incoming requests is high. When receiving the account and password of a user, the guard is in charge of enforcing access control policies. Charging policies, e.g., pay as you go, can be implemented in this component. After the verification phase, the user's request is transferred to the launcher, and then is transformed into a program in MapReduce. The program will be triggered by the launcher and executed by the logic layer. Finally, the service layer obtains NDD results stored in XML format from the logic layer and forwards them to the user application.

The second layer is the kernel of MapDupReducer, which implements the core business logic of the system. Much functionality of MapDupReducer, e.g., the total ordering of words within the current document collection, is implemented in the logic layer. The heart of the system, the near duplicate detection by the PPJoin paradigm in MapReduce, is also developed and will be described in the next subsection. To facilitate writing codes in MapDupReducer, a library called "NDD MapReduce library" is provided, which includes many useful routines, such as test data generation, and global ordering map construction.

The third layer, called the storage layer, is responsible for the secure and effective storage of massive datasets. The storage layer has two kinds of nodes. Repository nodes store all the documents; each document has several redundancy copies to account for node failures. Metadata nodes store all necessary meta-information, such as locations where a document and its copies are stored.

2.2 Near Duplicate Detection in MapReduce

In this subsection, near duplicate detection by the PPJoin paradigm adapted to the MapReduce framework is presented. Compared to our previous serial PPJoin algorithm [5], we have significantly extended the PPJoin paradigm by (1) redesigning the position and prefix filtering specially for the MapReduce framework, and (2) introducing document signature filtering to further reduce the candidate size. These optimizations substantially reduce both the computational and network workload. As shown in Figure 2, the processing framework mainly consists of four MapReduce jobs, and is presented as follows.

Firstly, the pre-processor parses, cleans, and transforms each document into a multiset of tokens, i.e., stemmed words, which are stored in a file called a pre-processed document. The lexer mapper simply records each occurrence of a token; the accumulator reducer summarizes the frequency of the occurrence of each token and generates a total ordering of tokens based on the decreasing order of token's inverse result. As well as, the signature of each pre-processed document is computed offline by the signature generator [3].

The prefix position mapper processes pre-processed documents to generate (*key*, *value*) pairs where *key* is the token, and *value* consists of (1) the identifier of the document in which *key* occurs and (2) the position of *key* in the total ordering (rather than the location of *key* in the document). The inverted indexer reducer isn't trivial. It generates (*key*, *value-list*), where *key* is composed of the identifiers of two candidate duplicate documents and the position of *key* in the first document, and *value* is the concatenation of the positions of *key* in the two documents. The information of the first position is so useful in the following steps that it is stored in *key* to avoiding need to be repeated extracted from *value*.

The group partitioner mapper is an improvement of the identity mapper. In this mapper, all input pairs are partitioned and grouped by a prefix of *key*, which is just the concatenation of the identifiers of the two candidate duplicates. By this way of partitioning, pairs with the same document identifier combinations will *always* be sent to the

[1] `http://hadoop.apache.org/mapreduce/`

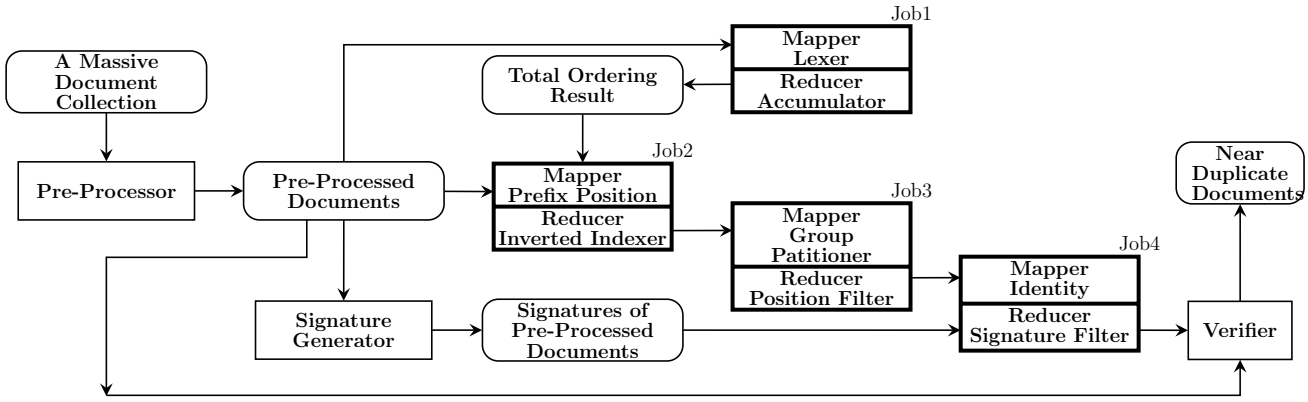

Figure 2: Near duplicate detection by the PPJoin paradigm in MapReduce

same computing node. The position filter reducer is a main component of MapDupReducer. It gets all the position information and implements the position logic adapted from the PPJoin algorithm in [5] to generate candidate pairs.

In the fourth MapReduce job, the mapper is just an identity mapper. The input of the signature filter reducer includes the location information of the signatures of the document set and the (*key*, *value-list*) pairs, where *key* is an identifier of a document and *value* is a list of identifiers of all candidate documents that are potentially similar to the document identified by *key*. For every candidate document pair, the signature filter reducer fetches and compares their signatures. When the distance of the two signatures is less than a given threshold, the pair is sent to the verifier.

For candidate document pairs that pass the signature filter reducer, the verifier fetches their corresponding pre-processed documents from the storage layer and compares them to generate the final NDD results. Thanks to the pruning of various filters we implemented above, the computational cost and network traffic are markedly decreased, compared with a straight-forward implementation that compares pair-wise document similarities.

3. DEMONSTRATION DESCRIPTION

In the demonstration, we will (1) motivate the near duplicate detection problem in the data management context, and demonstrate several novel uses of the system to satisfy a variety of information demands; (2) introduce several features of the MapReduce model and showcase their usage in helping user detecting near duplicate documents; and (3) demonstrate a set of experiments that evaluate performance of the system, and compare results of different implementations, including in single processor environment and MapReduce environment.

We use remote desktop connection in the demonstration. The server engine of MapDupReducer consists of 12 computing nodes and is physically away from the demonstration venue. A small dataset, whose size is 200 GB, randomly sampled from the BING dataset is prepared. In case of low connectivity at the demonstration venue, the system will be demonstrated as usual except that the MapReduce program runs on a single-node in a pseudo-distributed mode with the small dataset. We give further details below.

3.1 Introduction and Datasets

The first part of the demo will be an overview of the near duplicate detection topic in the data management context. We will systematically review recent research results of near duplicate detection and their extensive applications, such as effective document archiving, search engine results evaluation, and spam filtering. We will then give basic concepts on the MapReduce programming model and explain some classic examples (e.g., WordCount). We will also showcase some real massive datasets, including the BING dataset, and argue the necessity of detecting near duplicates within them.

We will then introduce the datasets used in the demo.

MEDLINE We extract and concatenate abstract, title, and author fields from MEDLINE documents. It contains 18.5 million records.

BING We sample a set of Web pages with an aggregated size of 2TB from the indexed Web page set of the BING search engine in July 2009.

Hotspot We use a subset of news Web pages from the BING dataset. Due to its nature, this set of documents has a high update frequency.

3.2 Demonstrating Near Duplicate Detection

Near duplicate detection in MapReduce is the core functionality of MapDupReducer. The design principle is to be intuitive and simplistic.

We will first motivate and introduce the system in the context of finding plagiarized documents in the MEDLINE dataset. Then, we will illustrate the PPJoin paradigm in MapReduce by the same dataset step by step, and run MapDupReducer to showcase the principle of the backend by analyzing the log information. Afterwards, we will let the audience interact with MapDupReducer by altering its arguments, e.g., the number of mappers/reducers used, to obtain understanding of the underlying technology. Also, we will let the audience select different options of the user application and run the demo system as follows.

In MapDupReducer's setup dialog, a user has the option to select a data source, set a parameter, and upload a custom stop word list. After pressing the "Run" button, MapDupReducer executes near duplicate detection process using the MapReduce-based PPJoin algorithm presented in the previous section. The output is visualized as a graph in

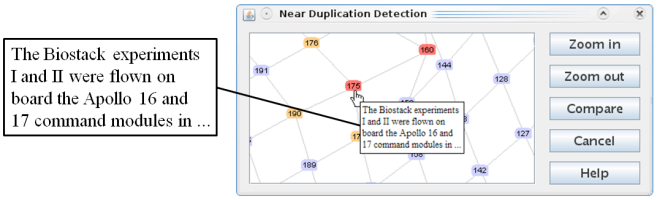

Figure 3: The near duplication detection interface of MapDupReducer

which two nodes connected with an edge if they are identified as near duplicates. For example, Figure 3 shows the result of near duplicate detection by the PPJoin paradigm in MapReduce for the MEDLINE dataset. Each document in the graph is represented by its unique document identifier (an integer). The user can zoom in/out the diagram as (s)he wishes. The beginning of a document is shown as the tooltip in the graphical interface when the user selects it; the differences between two duplicates are shown in color after the user selects them and clicks the button "Compare".

3.3 Demonstrating Searching for Similar Documents

Apart from running NDD in a batch mode as shown in the above demo, we will then motivate and introduce the system in the context of finding similar news articles against a large collection of indexed news documents. Users can upload a document or simply give a url to a Web page or text document. The system will perform certain preprocessing to extract the main textual content and use it as a query document D_Q. The NDD process is then performed with the restriction that we only consider candidates formed by D_Q and other documents in the database. The end result is returning a list of similar news articles for users to review. Users also have the options such as adding the query document to the database, deleting existing documents, or selecting one document as the representative among similar documents.

3.4 Demonstrating System Performance

We will demonstrate the performance of MapDupReducer interactively with audience on the Hotspot dataset. (1) We will let users select to open or close, respectively, the prefix, position, signature filtering to showcase their influences on the time cost of NDD in MapReduce. (2) We will also compare the performance between our PPJoin paradigm in MapReduce and other existing methods on computing pairwise similarity on document collections in MapReduce [4].

4. REFERENCES

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, California, USA, December 2004. USENIX Association.

[2] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[3] A. Kołcz, A. Chowdhury, and J. Alspector. Improved Robustness of Signature-based Near-replica Detection via Lexicon Randomization. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 605–610, Seattle, Washington, USA, August 2004. ACM Press.

[4] J. Lin. Brute Force and Indexed Approaches to Pairwise Document Similarity Comparisons with MapReduce. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 155–162, Boston, MA, USA, July 2009. ACM Press.

[5] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient Similarity Joins for Near Duplicate Detection. In *Proceedings of the 17th International Conference on World Wide Web*, pages 131–140, Beijing, China, April 2008. ACM Press.