MapReduce Style Algorithms for Extracting Hot Spots of Topics from Timestamped Corpus

Ashwathy Ashokan
University of Nebraska, USA

Parvathi Chundi
University of Nebraska at Omaha, USA

INTRODUCTION

We live in a world of ever-increasing amounts of information. This trend of "information overload" has quickly overwhelmed our capabilities to explore, hypothesize, and thus fully interpret the underlying details in these data. We make sense of the world around us by turning this data into meaningful information. Analyzing unstructured text documents such as blogs, news articles etc. for temporal information is an important data mining activity due to the pervasive nature of these data (Ashokan & Chundi, 2013). It is common for blogs and news sources to discuss/publish a few news stories intensely for a period of time. The topics covered in news stories may change frequently and be replaced with new topics, or they may stay active and the context surrounding the topic may change. This kind of coverage results in bursty patterns of stories/topics (Kleinberg, 2003). It has been well recognized that identifying periods of bursty activity of a topic may provide a lot of useful information that could be utilized by businesses, policy makers, and researchers. Extracting the *hotspot* of topics in a time-stamped corpus is one of the ways for identifying and analyzing such bursty patterns. The time periods of these bursty patterns for a particular topic/word are identified, and the time period of maximum burstiness of the topic/word is known as the *hotspot* for the topic/word. Methods such as term or document frequency can be used to compute the presence of a topic in a corpus. Each interval in the time period of the corpus is associated with a numeric value that we call the *discrepancy score.* A high discrepancy score indicates that the documents in the time interval are more focused on the topic than those outside of the time interval. A *hot spot* of a given topic is defined as a time interval with a highest discrepancy score.

As amount of data in our world has been exploding, the key to analyzing large data sets is efficient ways of computation. The MapReduce programming framework provides the building blocks necessary to split a task into subtasks, carry out the subtasks in parallel and combine the outputs of subtasks into one final output. Therefore, MapReduce style of computation of hot spots scales very well for big data.

This article discusses using the MapReduce style programming to improve the efficiency of the algorithms to compute hot spots. The section on Background gives an overview of hot spots and MapReduce with simple examples. A Main Focus section where details of MapReduce implementation for hot spots are discussed follows this section. The Future Trends section gives an insight into the future scope of this work and the Conclusion section concludes the article. Several important terms and their definitions are also included at the end of the article.

BACKGROUND

Hotspot Extraction Problem

A hotspot of a topic in a given data set of time stamped documents is a subinterval of the time period that contains significantly more documents that discuss the topic than the rest of the time period. Identifying a hot spot may provide a lot of useful information. To identify hot spots, a discrepancy score is assigned to

DOI: 10.4018/978-1-4666-5888-2.ch407

each of the $O(n^2)$ intervals during the time period of the corpus. A discrepancy score of an interval is a numerical value that captures the discrepancy between the presence of the topic in the documents of the interval and its presence in the documents outside the interval. There are many ways to compute discrepancy scores. The notion of a temporal scan statistic (Scan Statistics Website) is typically used to compute the discrepancy score of an interval.

We define the hotspot extraction problem as following: given a time stamped corpus and a topic, identify a time interval with the maximum discrepancy score. Note that there may be more than one such interval. One of those intervals is arbitrarily chosen as a hot spot. Extracting a hot spot requires calculating the discrepancy score of every interval in the time period of the corpus. A naive implementation runs in time $O(n^3)$, where n is the number of the time points of the corpus. A topic can simply be a keyword, a list of keywords, or may contains keywords connected with the logical operators AND, OR and NOT (Chen, Chundi 2011). In this article, a topic is assumed to be a simple keyword. However, the algorithms can be extended to more general notions of a topic.

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets. It is a distributed, parallel, fault-tolerant and scalable programming model. Today, it is largely being used for expressing distributed computations on massive amounts of data and an execution framework for large-scale data processing on clusters of commodity servers. It was originally developed by Google and built on well-known principles in parallel and distributed processing dating back several decades (Dean, Ghemawat 2004). MapReduce has since enjoyed widespread adoption via an open-source implementation called Hadoop, whose development was led by Yahoo (now an Apache project).

MapReduce builds on the observation that many information-processing tasks have the same basic structure: a computation is applied over a large number of records (e.g., Web pages) to generate partial results, which are then aggregated in some fashion. Naturally, the per-record computation and aggregation function vary according to task, but the basic structure remains fixed. Taking inspiration from higher-order functions in functional programming, MapReduce provides an abstraction at the point of these two operations. Specifically, the programmer defines a "mapper" and a "reducer" with the following signatures:

map: $(k1, v1) \rightarrow [(k2, v2)]$

reduce: $(k2, [v2]) \rightarrow [(k3, v3)]$

Key/value pairs form the basic data structure in MapReduce. The mapper is applied to every input key/value pair to generate an arbitrary number of intermediate key/value pairs. The reducer is applied to all values associated with the same intermediate key to generate output key/value pairs. This two-stage processing structure is illustrated in Figure 1. The data flow between map and reduce tasks is colloquially known as "the shuffle," as each reduce task is fed by many map tasks. And in additional the key/value pairs gets aggregated by the keys and sorted by the values before getting fed into the reduce task. This intermediate phase is known as "Shuffle and Sort."

Under the framework, a programmer need only provide implementations of the mapper and reducer. On top of a distributed file system, the runtime transparently handles all other aspects of execution, on clusters ranging from a few to a few thousand nodes. The runtime is responsible for scheduling map and reduce workers on commodity hardware assumed to be unreliable, and thus is tolerant to various faults through a number of error recovery mechanisms. The runtime also manages data distribution, including splitting the input across multiple map workers and the potentially very large sorting problem between the map and reduce phases whereby intermediate key/value pairs must be grouped by key. MapReduce allows for distributed processing of the map and reduction operations.

Provided each mapping operation is independent of the others, all maps can be performed in parallel – though in practice it is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase - provided all outputs of the map operation that share the same key are presented to the same reducer at the same time, or if the reduction function is associative. While this process can often appear inefficient compared to algorithms that are more sequential, MapReduce can be applied to significantly