Web Mining Accelerated with In-Memory and Column Store Technology

Patrick Hennig, Philipp Berger, and Christoph Meinel

Hasso-Plattner-Institut
University of Potsdam, Germany
{patrick.hennig,philipp.berger,office-meinel}@hpi.uni-potsdam.de

Abstract. Current web mining approaches use massive amounts of commodity hardware and processing time to leverage analytics for today's web. For a seamless application interaction, those approaches have to use pre-aggregated results and indexes to circumvent the slow processing on their data stores e.g. relational databases or document stores. The upcoming trend of in-memory, column-oriented databases is widely used to accelerate business analytics like financial reports, but the application on large text corpora remains unaffected. We argue that although in-memory, column-oriented stores are tailor-made for traditional data schemes, they are also applicable for web mining applications that mainly consists of raw text informations enriched with limited semantic meta data. Thus, we implement a web mining application that stores every information in a pure main memory data store. We experience an acceleration of current web mining queries and identify new opportunities for web mining applications. To evaluate the performance impact, we compare the run-time of general web mining tasks on a traditional row-oriented, disc-based database and a column-oriented, in-memory database using the example of BlogIntelligence, which serves exemplary for web mining applications.

Keywords: blog analysis, web mining, data mining, in-memory, column-layout.

1 Introduction

Since the information overload problem occurs in the mid of the 90s [8], the amount of available information in the Word Wide Web grows exponentially. Hence, the research area of web mining develops and gains increasing importance in today's businesses and life. Web mining essentially is the task of deriving knowledge out of the available information on the internet and delivering it to the user [7]. As the amount of data in the web continues to growth new challenges of data processing occur that have to be handled by today's web mining applications.

In general, web mining is the appliance of data mining techniques to web documents [4]. It tries to derive new knowledge of the vast pool of available data

M. Yao et al. (Eds.): ADMA 2013, Part I, LNAI 8346, pp. 205–216, 2013.

by automatically extracting and discovering information. According to Kosala et al. [7], web mining comprises the following sub tasks:

- **Resource finding** includes the retrieval of web documents based on the user's need like a Google search.
- **Information selection and pre-processing** consists of tasks of natural language processing and information abstraction like extracting receipts from web documents.
- **Generalization** is the automatic discovery of new patterns across a set of web pages like clustering or trend detection.
- **Analysis** reasons the recognized patterns and gives interpretations.

We introduce a tailor-made web mining application focused on weblogs and social media, called BlogIntelligence. During the development of this application we tested different kind of web-scale data stores. Based on our experience from various analytical algorithms, we identified the need for a relational data schema. Although it is hard to apply recursive link analysis algorithms, it supports delivers a high performance for aggregates.

Based on BlogIntelligence, we identify the necessity to compare the benefits of different data stores by looking at typical tasks of web applications. BlogIntelligence helps us to discover the variety of tasks reaching from selecting the next urls to clustering of web pages in communities.

We argue that based on our experience, document stores offer too limited analysis capabilities and traditional disc-based relational databases cannot handle the massive amount of data produced by the social web.

Finally, we conclude that an in-memory, column-oriented data store as proposed by Plattner [12] offers better analytical performance and opens new ways of analytics. Therefore it is necessary that all analytical data can be stored in main-memory. Due to the massive amount of cheap main memory that is meanwhile available this is not a problem anymore.

This paper is structured as follows. In the next section, we describe related data store techniques with their pros and cons. In Section 3, we give an introduction to BlogIntelligence which is an exemplary web mining application. Section 4 describes the different application areas, which the in-memory technology extremely accelerates the computing time and where it has possible shortcomings. Furthermore, we propose new analysis techniques that extend the feature set of today's web mining applications. We test our assumptions in Section 5 by comparing the execution time of essential queries of a row-oriented and column-oriented database. We give a short outlook and propose adaption for the tested data store in Section 6. Finally, Section 7 summarizes our work and conclude our results.

2 Related Work

The related approaches to speed up the store and analytical components of web mining application roughly consists of three areas.

First, the traditional row-oriented disc-based database approaches like Postgres [9] and MySql [14]. These are based on a B-Tree structure and optimized for accessing patterns of traditional hard-drives. Further, they offer index mechanisms that are beneficial for data access and query processing (eg. cube constructs). Nevertheless, those structures are pre-aggregated and need a high amount of processing time to be created. In addition, traditional databases fail to scale up to the massive data load of the web.

The second area consists of distributed data stores that use a large number of commodity hardware like Google BigTable [3] or Apache Cassandra [5]. These are specially adapted to handle massive amounts of data and to provide fast search access. Hence, each inserted data gets preprocessed and categorized into the tailor-made meta structure. Although the access performance of those data stores is remarkable, the analytical algorithms applicable are limited or need to process the whole data set at once like MapReduce PageRank [1].

Thirdly, the usage of complex index building applications that aggregate beforehand data for predefined questions like Apache Lucene[1]. The idea behind those applications is to incorporate with traditional databases and deliver fast query results for various kinds of analytical queries. Furthermore, they offer the possibility to integrat natural language processing stages into their index building process. Nevertheless, complex analysis algorithms like topical clustering, ranking, or equally complex aggregations need to be facilitated by external components.

In contrast to the related work, we argue that with the availability of massive, inexpensive main memory, web mining can be leveraged by in-memory database with a tremendous performance gain. The advantage of those databases is that the data access is equally fast than the data aggregation. We implement and test web mining operations for an in-memory database and equally for a traditional database (see Section 5).

3 Blog Intelligence

Blog Intelligence is a web mining application tailor-made for blog mining with the objective to map, and ultimately reveal, content-oriented network-related structures of the *blogosphere* by employing an intelligent *blog crawler*. As described in [10], BlogIntelligence is able to harvest the pool of millions of interconnected blogs, called *blogosphere*.

We want to gain a better user experience and discover new analyses by using the benefits of in-memory computing column-oriented store for the unusual application area of web mining. The identified benefits are discussed and finally evaluated in this paper. We look at some critical parts in the overall system to evaluate the different techniques.

[1] `http://lucene.apache.org/`

4 Application Areas

In general, a web mining application consists of three components that are responsible for the three main tasks: crawling data, storing data, and analyzing data. The crawling of data is the actual process of downloading web pages from the internet. Next, storing the data is the task of managing the massive amount of data and prepare it for analytical queries. Finally, analyzing data consists of running diverse text mining and natural language processing algorithms including community discovery, influencer identification, and topic extraction.

The performance of the crawling and analyzing component depend dramatically on the performance of the intermediate data store. We try to understand the relation between the data store and the other two components, and the possible performance gain through a in-memory, column-oriented data store. Hence, we introduce the characteristics of these components using the example of *BlogIntelligence*. In addition, we give an overview of new analyses ways enabled by the usage of in-memory technology.

4.1 Crawling

The major prerequisite of web mining tools, like BlogIntelligence, is harvesting web pages. These crawling activities deliver the data that is necessary for the underlying analytics. During the development of our focused web mining application, *BlogIntelligence*, we experienced major benefits of the in-memory technology in both areas.

The crawling of web pages consists essentially of downloading pages and selecting new urls to crawl. The BlogIntelligence framework uses an intelligent and scalable tailor-made blog-crawler [10] to harvest blog pages.

Especially inserting data is a common task for harvesting weblogs. By design, the insertion costs of column-oriented database are comparable high. This is caused by the distribution of the column values in the main memory, which results in high insert costs. In contrast, a row-oriented layout enables the database to write one line sequentially into the main memory without caring about any specific place for the column values. Anyway, since each website is only inserted once, this disadvantage is almost negligible. An evaluation of the execution time of insert webpages is given in Section 5.

Beside inserting pages, selecting new pages is another frequently executed task during harvesting to identify the best pages for crawling. The complexity of the selection of new urls varies among different use cases. A general crawler simply selects the next not visited url from the database, whereas a specialized crawler has to ensure more complex selection constraints.

A constraint can dictate a specific priority for each web page depending on the rank of the source page, the last visiting date of the page itself, the expected content of the page, the type of the page or even restrictions of the content provider. We describe and evaluate an example query of our focused web crawler, which is part of BlogIntelligence, in Section 5.

4.2 Analytics

The analytics of BlogIntelligence are at a magnitude more complex than the crawling. In the following, the major algorithm types that get applied in BlogIntelligence are introduced.

One typical application area is a link analysis algorithm. The most prominent algorithm is PageRank [11]. It calculates a rank for a web page based on the rank of all incoming web pages. For Blog Mining this gets even more important since it is very important to select the most important and interesting blog. Therefore such a ranking function can be improved to take the structued data of the blogosphere into account as described in the previous work [6]. Caused by the inherent complexity of recursive ranking algorithms in databases, we simplified the example ranking query to an subcomponent of the rank calculation. Therefore we want to demonstrate this simply by calculating a count of incoming links of blogs. We expect this select and count aggregation to perform fast on a column-oriented layout.

Sentiment analysis handles the problem of sentiment extraction from sentences in the web page's text. Thereby, an algorithm applies a set of predefined language specific rules that identify whether a word has a positive or negative meaning. These extraction as well as an entity extraction is done asynchronously by inserting data into the database. Therefore these aspects can be retrieved afterwards pretty fast via SQL and can be used in very simple way for other calculations. Nevertheless, this is a database-specific extension that can also be executed by external application.

In addition by inserting new data into the database some additional structures can be filled and therefore kept up-to-date all the time. For example a compressed Document-Term matrix helps to calculate similar terms or similar documents based on the well-known TF-IDF [13] measure. In the evaluation part some execution runs are shown for this analysis.

Compared to link analysis algorithms, blog rank algorithms incorporate a set of additional factors. Another metric BlogIntelligence provides is to rank blogs according to the consistency of the content they are writing about. For example if an author is writing about the same topic all the time, he should have a big knowledge in this topic. This is accomplished by looking at the usage of the tf-idf measure.

Last but not least, another important part is to identify top emerging trends within the blogosphere. Since the metric is basically specified the user can get trends for his own topic space and change the metric according to his special interests.

4.3 New Application Area

In former times the analytics were carefully designed beforehand. Afterwards the execution of these analytics produced the results, which got visualized in a meaningful way. The execution of the complete analytics often took several days, even with a limited data set.

As a result two major problems occurred. The first major problem of this life cycle is that it contradicts the ever changing nature of the World Wide Web. Therefore, the results are out-dated by the time they are produced.

The second major problem is that the analytics have to be set up-front. This can be hidden from the end-user by offering smart filtering options for the analytical results. Nevertheless, the user is not able to change the underlying metrics and get immediate feedback of his change via a freshly calculated result.

In addition, it is very important to give the user the possibility to adapt metrics according to his own interest. If a user wants to get the best and most interesting blogs, he often does not want to use a general metric. Since no results are pre-aggregated, the user can change a metric until it fits to his needs. The user can even limit the analyses to a specified topic space.

Furthermore, it was not feasible to ask the database questions like the following:

- Which are the blogs with the most incoming links in a specific topic?
- Who are the authors with the most posts writing about politics?
- What are the most recent posts which the user is interested in?
- How is a topic discussed in its community or within a certain time frame or within the most popular blogs?

By using in-memory technology together with a column store we are able to answer these questions without knowing the question beforehand. As well, we can immediately provide results for his latest analytical questions at design time.

In order to get a better impression, we want to take a deeper look at the basic extraction, crawling and analyses tasks in the next section.

5 Evaluation

In this Section we evaluate the performance of the PostgreSQL database[2] and an in-memory database from SAP on the presented web mining operations.

5.1 Data Set

The data set for this evaluation consists of 73 221 376 webpages. 2 473 898 of these webpages are parsed. Parsed webpages are already downloaded and processed by the web mining applications that results in additional informations like raw text, feed items, authors, categories and other semantic data.

After the classification BlogIntelligence identified 15 327 blogs with 818 865 posts. The link graph of the web is also represented as a table that consists of 200 000 000 entries. The space used for the two main tables, webpage and link, is 200 gigabyte. Hereby, the webpage table uses 151 gigabytes and the link table uses 49 gigabytes.

All tables are stored as column-oriented tables with a primary index. Some columns are compressed by the database which leads to the small amount of

[2] `http://www.postgresql.org/`

main memory needed for the dataset. As a big advantage of the column store the table containing all blog pages, the webpage table, is able to handle 79 columns including the original HTML page, the extracted text, authors, the publishing date as well as other information.

This data set is the result of a 3-week-run from August 2012. We use this data set because we like to provide real results to the users of our prototype and measure the performance more accurate by using a data set that is not cleaned or adapted in any way.

5.2 Setup

Both databases get evaluated on the same server hardware. The database server has 32 cores one terabyte of main memory. The full file system size is 20 terabyte. The file system is located on a EMC storage, which uses SSDs for storing. During the execution of the experiments the server is exclusively used by the databases and only the minimal SUSE Linux basic daemons are running beside the database.

Since the in-memory database is operating exclusively on main-memory the time to load data into main memory is negligible. Therefore the tables has to be re-constructed from the disk only if the database server was shutdown completely. Fore more details about the concepts of an in-memory database the Book of Prof. Hasso Plattner [12] provides deeper insights.

For this setting we want to look at different aspects that specially depend on the performance of the data stores for web mining applications.

5.3 Crawling

Heavy Inserts. As already mentioned a column-oriented layout is not the best choice for processing a lot of different insert statements. Nevertheless it provides such an improvement benefit in the analytical part that this can be neglect. However, the in-memory database from SAP used in the experiment is optimized to absorb these disadvantages.

For the experiment 200 000 insert commands were sent to the database, filled with generated content. The web page table has columns from almost every type, from binary columns over integers to big text columns.

For the execution of the 200 000 insert commands a parallelized Java application running with 50 threads is used together with a JDBC driver for both databases. The execution results are shown in the following Table 1.

As we can see by looking at Table 1 the database is able to absorb the low insert performance of a traditional column-oriented data store. In average the in-memory database is still 20 seconds faster than the traditional row-oriented layout.

Selection of Next Job. The selection of the next urls to crawl, called job, is one of the most frequently executed tasks during crawling. As described in

Table 1. Execution of 200 000 insert queries

Run	SAP Hana	PostgreSQL
1	1m 29s 155ms	3m 8s 466ms
2	2m 32s 622ms	3m 8s 466ms
3	2m 32s 622ms	3m 50s 583ms
4	3m 53s 147ms	3m 22s 176ms
5	3m 58s 291ms	4m 29s 566ms

Section 4.1, a focused crawler for blogs has to incorporate complex constraints for the selection.

In former times the complex selection process as described previously was a pretty time consuming task. With the change to an in-memory database this task can be accomplished within a few seconds. The actual crawler uses a query with more than 10 constraints. We simplified the query to one constraint that uses the pre-calculated fetchtime for a URL 1.1. The fetchtime is calculated while the URL was inserted based on our own heuristic, when it is most promising to download this URL.

Listing 1.1. Selection of next job

```
SELECT ID FROM WEBPAGE
WHERE FETCHTIME < (currentTimeInMillis)
ORDER BY SCORE DESC LIMIT 10000
```

The results of this experiment is shown in Table 2 after several executions on each database.

Table 2. Selection of next urls

Run	SAP Hana	PostgreSQL
1	0m 21s 919ms	3m 46s 666ms
2	0m 21s 554ms	3m 46s 007ms
3	0m 21s 832ms	3m 41s 213ms
4	0m 21s 745ms	3m 56s 505ms
5	0m 21s 811ms	3m 56s 198ms

By looking at Table 2 we can see that the selection of the next urls is up to 8 times faster with the in-memory database. Furthermore in the real scenario this difference would even more important.

5.4 Analytics

Blogs with the Most Incoming Links. As already mentioned the ranking got very simplified for this experiment, by counting the incoming links and order them accordingly by the top blog with the most incoming links.

Listing 1.2. SQL Query most incoming links

```
SELECT COUNT(*) as incomLinks, toHost
FROM link
GROUP BY toHost ORDER BY COUNT(*) DESC
```

The execution time for each database of this experiment is shown in Table 1.3.

Table 3. Execution of the link query

Run	SAP Hana	PostgreSQL
1	0m 11s 800ms	1m 46s 140ms
2	0m 11s 955ms	1m 43s 245ms
3	0m 11s 735ms	1m 44s 058ms
4	0m 11s 780ms	1m 42s 443ms
5	0m 11s 940ms	1m 45s 194ms

Table 1.3 shows that the execution time of this query could in average decreased by factor 10.

Big Blogs with Additional Information. At the first look it sounds pretty easy to retrieve information from the biggest blogs. But in order to get additional information of each blog, a join between the table containing blogs and the table containing links is necessary. This makes it more complicated for the database to handle.

Listing 1.3. Blogs with additional information

```
SELECT POSTTITLE, POSTAUTHOR
FROM WEBPAGE INNER JOIN (

SELECT toUrl, COUNT(*) as links
FROM link
GROUP BY toUrl

) as link
ON link.toUrl = webpage.ID
WHERE type='POST'
ORDER BY links desc
```

Table 4. Execution of the addition info query

Run	SAP Hana	PostgreSQL
1	1m 12s 544ms	208m 02s 737ms
2	0m 1s 722ms	199m 51s 321ms
3	0m 1s 868ms	233m 42s 876ms
4	0m 2s 243ms	201m 33s 128ms
5	0m 1s 925ms	214m 55s 057ms

The execution time of this experiment is shown in Table 4. The measurement reveals a significant difference in the execution time. The in-memory database is faster by a few orders of magnitude. This difference is caused by the layout of the database. Since we are just looking at columns, it is not a problem if a table has lot of columns. For a row-oriented database this is a big problem. Therefore the PostgreSQL database has to touch 150 gigabyte of data which of course takes some time. Another interesting point is, that the in-memory database obviously is performing some kind of caching. Thus, the execution gets really fast if the database stays untouched.

Calculate Similar Terms / Similar Documents. In addition we take a short excursion at text mining area that is essential for web mining applications. As discussed, the tested database provides a compressed Document-Term matrix and the functionality to calculate similar terms or similar documents with the well-known tf-idf measure.

Performing a search for similar terms can be accomplished on this data set within short time. The following Table 5 shows the execution time for finding similar terms for given terms. In this experiment the text-mining index contains 400 million entries with 7 million unique terms.

Table 5. Execution of similar term search

terms	time
Apple	1025 ms
Apple or Microsoft	1057 ms
Obama	630 ms
Merkel or Obama	950 ms
Politik or Wirtschaft or Bildung	1620 ms

The experiments showed that the in-memory technology together with a column-oriented layout performs faster on simplified web mining tasks than a traditional row-oriented disc-based store. We expect that this performance gain applies also for the compound analytical tasks of web mining applications. Further, insert commands, which are expected to be cost-intensive tasks in a column-oriented layout, also perform very fast due to database optimizations specific to SAP HANA.

6 Future Work

In this paper we evaluated some basic data store task for crawling and analysis. Although we show performance improvements for these tasks, we need to dive deeper into complex analysis algorithms and their behavior on our storage. Thus, we investigating current analysis approaches and try to transfer them to our system. The following two directions support this objective and promise insight about complex tasks.

In order to benefit from this technique in the future, some additional functionality should be improved inside the database. As mentioned above, the internal Document-Term matrix helps to calculate text analysis in almost real time based on the latest data. Thus, we suggest to use similar structures to store a up-to-date clustering for blogs and related terms. A hierarchical clustering fit these requirements, because the amount of clusters is not known at creation time. This clustering provides the possibility to access a certain hierarchy level or the subtree of a specific topic space. We want to build a prototype using such a structure and evaluate this concerning the real-time performance.

Furthermore, we currently investigate a personalized rank editor. Ranking mechanisms for blog posts are a very interesting research topic as stated by Bross et al. [2]. Nevertheless, it is not possible to define one universal rank for every user of a search engine. Thus, we try to integrate the user into the ranking process and let the user decide what is important. The user gets the ability to define his own ranking formula based on intuitive factors. A first prototype of this ranking editor is already available at BlogIntelligence[3].

Since the user expects immediately feedback when defining a new ranking equation, the computational effort is tremendous. For example, if a user wants to search for blog posts ranked by a quotient between incoming and outgoing links, the whole link table has to be scanned every time. The system is still under construction, but we were able to get the first prototype running really fast. With a traditional database this would be unthinkable. We want to improve that functionality in the future and provide as many as possible ranking criteria for the editor.

In addition, we are working on a personalized trend detection interface. Hereby, a user can search for trends in his topic of interest and can select a certain time window to detect trends. This helps to understand discussions and trends in more detail and gives the user the possibility to predict trends in the future.

7 Conclusion

As shown in this paper we were able to apply the in-memory technology in conjunction with a column-oriented layout and gained an for the tested web mining application including crawling, analysis and visualization of big amount of web pages in this case blogs. We emphasize that the whole crawled web data including extracted meta data is stored in main memory without helping structures on disc. In addition, the optimization of the database used in the experiment is able to absorb the low performance when inserting thousands of tuples. This helps to make use of the tremendous performance improvements of the in-memory database when executing analytics.

Furthermore, we demonstrate that advanced analysis like similar term extraction can also be performed very fast by integrating the specific data structure like a document term matrix directly into the database.

[3] `http://www.blog-intelligence.com`

Finally, the change from an traditional row-based database to an in-memory column-oriented database provides BlogIntelligence complete new kind of analysis like the described ranking editor where users are able define their own ranking equation and immediately get the results for their ranking equation. BlogIntelligence is now able to provide personalized real-time analyses for each user separately according to the user's special topic of interest.

References

1. Bahmani, B., Chakrabarti, K., Xin, D.: Fast personalized pagerank on mapreduce. In: Proceedings of the 37th SIGMOD International Conference on Management of Data, pp. 973–984 (2011)
2. Bross, J., Richly, K., Kohnen, M., Meinel, C.: Identifying the top-dogs of the blogosphere. Social Netw. Analys. Mining 2(1), 53–67 (2012)
3. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) 26(2), 4 (2008)
4. Etzioni, O.: The world-wide web: quagmire or gold mine? Communications of the ACM 39(11), 65–68 (1996)
5. Hewitt, E.: Cassandra: the definitive guide. O'Reilly Media, Incorporated (2010)
6. Bross, J., Kohnen, M., Richly, K., Kohnen, M., Meinel, C.: Identifying the top dogs of the blogosphere. Social Network Analysis and Mining. Springer LNSN (2011)
7. Kosala, R., Blockeel, H.: Web mining research: A survey. ACM Sigkdd Explorations Newsletter 2(1), 1–15 (2000)
8. Maes, P., et al.: Agents that reduce work and information overload. Communications of the ACM 37(7), 30–40 (1994)
9. Momjian, B.: PostgreSQL: introduction and concepts, vol. 192. Addison-Wesley (2001)
10. Hennig, P., Berger, P., J.B.C.M.: Mapping the blogosphere - towards a universal and scalable blog-crawler. In: Proceedings of the Third IEEE International Conference on Social Computing (Social Com2011), pp. 672–677. IEEE CS, MIT, Boston, USA (2011)
11. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web (1999)
12. Plattner, H.: A course in In-Memory Data Management. Springer, Berlin (2013)
13. Sparck Jones, K.: A statistical interpretation of term specificity and its application in retrieval, pp. 132–142 (December 1988)
14. Widenius, M., Axmark, D., MySQL, A.: MySQL reference manual: documentation from the source. O'Reilly Media, Incorporated (2002)