APPROACHES FOR EFFICIENT HANDLING OF LARGE DATASETS

Renáta Iváncsy, Sándor Juhász
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Goldmann Gy. ter 3., Budapest, Hungary

ABSTRACT

Efficient handling of large datasets is a challenging task since in most cases the data to be processed do not fit into the memory, thus the high number of slow I/O operations will dominate the performance. There exist several methods for making data handling more efficient by compressing, partitioning, transforming the input data, suggesting more compact storage structures or increasing cache friendliness. Our paper investigates and categorizes these approaches and gives an overview of their benefits and drawbacks when using them in different stages of the data processing pipeline of a general information system. The performance demand of long running data processing is often coupled with operability requirements (like fault tolerance and monitoring).

KEYWORDS

Out-of-core data processing, efficient data handling, cache, parallelism

1. INTRODUCTION

Efficient handling of large amounts of data is a key issue in recent data processing systems since an increasing number of applications have to deal with a massive number of input records produced by automated systems. The most problematic feature is the memory limitation. Although during the recent years memory sizes grew significantly, this increase could not follow the growth of the sizes of the input data. Thus the data does not fit into the memory, thus serious considerations have to be made for enhancing reasons.

The complete process of data handling consists of several steps. To enhance this process the key points and bottlenecks have to be found and treated in an adequate way. Our work will enumerate and compare different ways to reach this goal and gives a systematical overview of the related work, in which we show, what kind of issues are used by the various researchers for enhancing data mining processes.

The organization of the paper is as follows. Section 2 introduces the main steps of a typical data processing task. Section 3 introduces the different approaches that can be used in order to speed up a data handling application. Section 4 maps the processing steps with the approaches to improve them and shows an overview of related work from the performance improvement perspective. Conclusion can be found in Section 5.

2. OVERVIEW OF A TYPICAL DATA HANDLING SYSTEM

This section introduces the internal structure of a typical data handling system, and seeks for key points, where some enhancement can be achieved. A typical data handling solution consists of the following steps: (i) data acquisition (ii) data selection (iii) data cleaning (iv) data transformation (v) executing the processing task and (vi) results interpretation and verifying.

Data acquisition means gathering the necessary data from one or more data sources. It is an important question what kind of data should be gathered for what kind of purpose. When planning an efficient data handling system, this is the first question that has to be answered.

Data selection means selecting the relevant data from the source file. Although in the data acquisition phase, thanks to the careful planning, only the necessary data are gathered, in most cases not all information has to be used by the different processing tasks. In most cases multiple processing tasks are executed on the same data collection for different purposes, that means, different parts of the data is be used by the various data handling tasks. For this reason the data selection phase is clearly an important point where the whole process can be enhanced. Only those data chunks have to be selected, that is really necessary for the target task.

Data *cleaning* is the phase in data handling process where the errors and inconsistencies are detected and removed form data in order to improve the quality of data (Rahm and Do, 2000). Data cleaning can reduce the search space when removing false entries, or for example by correcting misspells and so on.

The *data transformation* step prepares and adapts the data to fit the processing task. However beside this purpose performance enhancement considerations can also require transformation of the data into a different representation.

In the *processing task execution* step the main algorithm is applied to the preprocessed data. Of course, after the preprocessing steps, the main phase of the data handling process can contain several points where the data handling can be enhanced. As we will show later in Section 3, not only the algorithm to be used has to be chosen carefully, but also the nature and the properties of execution environment has a serious influence in the overall performance.

3. APPROACHES FOR DATA HANDLING IMPROVEMENTS

This section enumerates the different special purpose data processing approaches that aim to increase the efficiency of data handling as well. This means, that next to their basic goals (the detailed description of which is not in the scope of this paper) these methods can be used for enhancing the performance of the following steps as a pleasant side effect.

The data handling approaches mentioned below can be classified into two main groups. First class includes the methods targeting only a limited part (one or few fields) of each record at once (typically one or more fields of a record). We call these approaches field level methods. The second class, called record set level methods, contains the methods operating on a relatively wider part of the whole dataset (multiple records at the same time). These approaches are described in the following subsections.

3.1 Field Level Methods

Vertical decomposition means splitting the original input file by fields. As different steps of the whole process do not always work with the same fields, thus when reading the input records the fields of the record may be grouped according to the needs of the following algorithms. By vertically decomposing the records, the size of the data to be handled in the next steps is reduced significantly, as only those fields are selected for further processing, that are really needed by the processing task.

Smoothing means the removal of the noise from data. The techniques belonging to this form of data transformation are binning, clustering and regression (Han and Kamber, 2000).

Binning groups sorted data into “bins” by taking the neighborhood of the data into consideration. Clustering assigns similar objects to the same group, and separates dissimilar objects into different groups. Data can be smoothed by fitting the data with a function, such as linear or multiple linear regression. The aim of all the smoothing methods is to remove noise, thus enhance the performance and the accuracy of the subsequent steps.

The method *aggregation* means applying counting, summary or other aggregation operations (minimum, maximum, deviation) to the data. This can accelerate further processing steps where this type of information is used often and as well reduce the amount of data to be handled.

Generalization aims at replacing low level data with higher concepts. This technique can be regarded as a compression method.

Normalization means scaling the attribute data to fall within a specified range. This technique is also used for data compression purposes.

Attribute construction is the process of generating new attributes based on the existing ones to help the processing and improve the accuracy. Despite the increased amount of data generated, this method can significantly enhance the whole process by combining it with vertical decomposition.

Coding the attributes can make their handling becomes more efficient, or lower the storage requirement of the coded form.

3.2 Record Set Level Methods

Dataset conversion means converting the input data or their parts into a new format that can be handled easier or more efficient. Converting the whole record set into a different representation can also enhance the performance of the following processing steps. A typical example is converting the records into tree format (Han et al., 1999). Because of its high storage requirements, in most cases, this approach is used for handling large dataset in combination with other methods such as sampling or partitioning.

Instead of handling the complete dataset at once *sampling* selects a well chosen part of the whole dataset, which is read into the memory, and the further tasks are executed only on this small part of the dataset. Usually a final verification is required to check the results by using the complete original dataset.

The advantage of the approach is that the dataset to be handled fits into the memory, thus the I/O costs are minimized. The drawbacks are, however, that the results are based on the sampled part of the dataset, thus only partial results is obtained. This can produce approximate or even inaccurate results. Furthermore, the right heuristics for the sampling phase is not trivial to find in many cases.

Partitioning is a widely used approach of out-of-core data handling. Here, the input data is split into smaller blocks fitting into the memory, and the processing algorithm is executed on these parts of data successively. The main difference between sampling and partitioning is that in case of partitioning all the input records are used, that is, the union of the blocks completely covers the original input dataset, and all original records are guaranteed to be used once and only once.

The partitions themselves may contain subsequent or arbitrary selected (grouped) records. The processing task is first executed on the distinct partitions, where the results are written independently to the disk, and the global result, that is, the result characterizing the whole input dataset, is created in a subsequent step by merging the results produced from the individual partitions .The way the local results are used for generating the global results, is related to manner the local results were created. In some cases the global result is a simple union of the local ones (Grahne and Zhu. 2004). In other cases a light-weight merging task has to be accomplished (Savasere et al., 1995) (Lin and Dunham, 1998) (Nguyen et al., 2005) (Nguyen et al., 2006) (Lucchese and Perego, 2006), or some complex processing task has to be executed for generating the global result.

Current execution environments posses a memory hierarchy (disk, main memory, registers and caches between them) differing significantly in size and speed. This issue can be addressed at several levels. While the reduction of I/O steps (achieved by e.g. sampling, partitioning) is well discussed *memory cache handling* gained much less emphasis. When analyzing the performance of data intensive applications estimations based exclusively on the number of algorithmic steps can be misleading, while the memory access pattern becomes a key issue (Binstock, 1996) (Juhász and Dudás, 2008).

The importance of the cache friendly behavior of algorithms and data structures increases gradually as the data grow significantly larger than the size of the data cache (Heilman and Luo, 2005). Cache friendliness is related to the compactness of the data structures and the locality of subsequent data accesses. (Black et al, 1998) demonstrated this effect with a hash table based recoding of a large amount of data using the cache friendly linear probing and the double hashing. Despite the fact that this later required much less steps to take, it proved to be slower than linear probing as the randomly chosen memory locations took much less benefit of memory caching.

Slow data processing algorithms are natural candidates for *parallelization*. Two basic types of parallelization are distinguished in the literature: data and functional parallelism (Bustard, 1990) (Foster, 1995). Data parallelism runs the same processing operations on different execution units (processors, computers) feeding each with different parts of the original data set. The data distribution can be done following the master-worker or the divide and conquer patterns (Foster, 1995). Functional parallelism distributes the processing operations instead of the data between the execution units. In case of data processing application the most natural distribution units are the different processing stages that can be

organized into pipelines. Here using different computers and different processors of a single computer have both merits and can be combined as well.

4. MAPPING APPROACHES TO PROCESSING PHASES

In this section we will examine, in what phase exactly these approaches are applicable during the whole process. The approach and phase assignment is shown in Table 1. Table 2 gives an overview of some data mining related works (frequent itemset mining algorithms, web mining algorithms and complete systems and frameworks), where various data processing algorithms are shown with the optimization approaches their authors applied.

Table 1. Mapping Approaches to Processing Phases

		Acquisition	Selection	Cleaning	Transformation	Executing
Field level methods	Vertical decomposition		✓			
	Smoothing			✓	✓	
	Aggregation			✓	✓	
	Generalization			✓	✓	
	Normalization	✓		✓	✓	
	Attribute construction	✓			✓	
	Coding	✓		✓	✓	
Record set level methods	Dataset conversion				✓	
	Sampling	✓	✓			✓
	Partitioning	✓	✓		✓	✓
	Cache handling				✓	✓
	Parallelization	✓	✓	✓	✓	✓

Table 2. Systematic Overview of Related Work

		Sampling algorithm (Toivonen, 1996)	Partition algorithm (Savasere et al, 1995)	AS-CPA (Lin and Dunham, 1998)	RSAS-CPA (Lin and Dunham, 1998)	(Nguyen and Orlowska, 2005), (Nguyen and Orlowska, 2006)	FP-growth (Han et al, 2000)	Diskmine, (Grahne and Zhu, 2004)	WEBMINER (Cooley et al, 1999)	DCI_CLOSED_OOC (Lucchese et al, 2006)	ClickWorld (Baglioni et al, 2003)	(Heilman and Luo, 2005)	(Punera and Ghosh, 2008)	WAT (Iváncsy and Juhász, 2007)	(Benczúr et al, 2005)
Field level methods	Vertical decomposition								in use		in use			in use	in use
	Smoothing										in use		in use		
	Aggregation													in use	in use
	Generalization														in use
	Normalization										in use				in use
	Attribute construction								in use		in use				in use
	Coding								in use		Hash			Hash	Huffman, Hash
Record set level methods	Dataset conversion						FP-Tree	FP-Tree	user session		user session				star/snowflake schema
	Sampling	in use													
	Partitioning		Linear partitions, sort-merge	Linear partitions, sort-merge	Linear partitions, sort-merge	Cluster partitions, sort-merge		Projected partitions, union		Linear partitions, sort-merge				Block Periodic Partial Result Merging	in use
	Cache handling											in use		in use	
	Parallelization		Cluster											Pipeline	Pipeline

5. CONCLUSIONS

This paper summarized several methods aiming at improving the performance related behavior. After introducing the main steps of a typical data handling application the various methods are described that can be used for improving the efficiency of the process. The different approaches were then mapped to the various processing steps in order to have a clear view of the possibilities.

ACKNOWLEDGEMENTS

This work was completed in the frame of Mobile Innovation Centre's integrated project Nr. 3.2. supported by the National Office for Research and Technology (Mobile 01/2004 contract).

REFERENCES

Baglioni. M., Ferrara, U., Romei, A., Ruggieri, S., Turini, F. and Buonarroti, V.F., 2003, *Preprocessing and Mining Web Log Data for Web Personalization*, 8th Italian Conf. on Artificial Intelligence vol. 2829, pp. 237-249.

Benczúr A. A., Csalogány K., Lukács A. Rácz B. Sidló Cs., Uher M. and Végh L., *Architecture for mining massive web logs with experiments*, In Proc. of the HUBUSKA Open Workshop on Generic Issues of Knowledge Technologies

Binstock, A. 1996, *Hashing rehashed: Is RAM speed making your hashing less effcient?* Dr. Dobb's Journal, 4(2), April

Black, Jr. J.R, Martel, Ch. U, and Qi H. 1998 *Graph and hashing algorithms for modern architectures: Design and performance*. In Kurt Mehlhorn, editor, Proceedings of the 2nd Workshop on Algorithm Engineering (WAE'98), Saarbrücken, Germany, August

Bustard, D.W. 1990, *Concepts of Concurrent Programming*, SEI-CM-24, Carnegie Mellon University, Software Engineering Institute, USA, 1990.

Cooley, R., Mobasher, B. and Srivastava, J., 1999, *Data Preparation for Mining World Wide Web Browsing Patterns*, Knowledge and Information Systems, Vol. 1. No. 1., pp. 5-32

Foster, I., 1995, *Designing and Building Parallel Programs*, Addison-Wesley Inc., Argonne National Laboratory, USA

Grahne, G. Zhu J. 2004, *Mining frequent itemsets from secondary memory*, ICDM '04. Fourth IEEE International Conference on Data Mining, pp. 91-98.

Han J., and Kamber, M. 2000 : *Data Mining: Concepts and Techniques* , Morgan Kaufmann

Han, J., Pei, J., and Yin, Y. 1999 *Mining frequent patterns without candidate generation*. In Chen, W., Naughton, J., and Bernstein, P. A., editors, Proc. of ACM SIGMOD International Conference on Management of Data, pages 1-12.

Heileman, G.L and Luo, W., 2005, *How Caching Affects Hashing*, ALENEX/ANALCO, pp. 141-154.

Iváncsy, R. and Juhász, S. 2007, *Analysis of Web User Identification Methods*, Proc. of IV. International Conference on Computer, Electrical, and System Science, and Engineering, CESSE 2007, Venice, Italy, pp. 70-76.

Juhász, S. and Dudás, Á. 2008, *Optimising large hash tables for lookup performance*. Proceeding of the IADIS International Conference Informatics 2008, Amsterdam, The Netherlands, pp. 107-114

Knuth, D. E. 1973, *Searching and Sorting*, volume The Art of Computer Programming. Addison-Wesley, Reading, MA, 3rd edition

Lin J. and Dunham M. H 1998., *Mining association rules: Anti-skew algorithms*, In 14th Intl. Conf. on Data Engineering, pp. 486-493.

Nguyen Nhu, S., Orlowska, M. E. 2005, *Improvements in the data partitioning approach for frequent itemsets mining*, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-05), pp. 625-633.

Nguyen S. N. and Orlowska M. E. 2006, *A further study in the data partitioning approach for frequent itemsets mining*, ADC '06 Proceedings of the 17th Australasian Database Conference, pp. 31-37.

Punera, K.; Ghosh, J., 2008, *Enhanced Hierarchical Classification via Isotonic Smoothing*, 17th International World Wide Web Conference (WWW), Beijing, China, pp.151-160,

Rahm E. and Do H. H., 2000, *Data Cleaning: Problems and Current Approaches* IEEE Data Engineering Bulletin

Salvatore C. L., Perego O. R. 2006, *Mining frequent closed itemsets out-of-core*, 6th SIAM International Conference on Data Mining, pp. 419-429.

Savasere A., Omiecinski E. and Navathe S. 1995, *An efficient algorithm for mining association rules in large databases*, VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases, pp. 432-444

Toivonen H., 1996 *Sampling Large Databases for Association Rules*, Morgan Kauffman, pp. 134-145.