Out-of Core Processing in Preparation Phase of Data Mining Tasks

Tamás Schrádi, Sándor Juhász

Budapest University of Technology and Economics, Hungary
schraditamas@aut.bme.hu, sanyo@aut.bme.hu

Abstract: This paper compares out-of-core approaches used in data pre-processing phase of data mining task dealing with vast datasets. Due to its high cardinality the dataset exceeds the limits of straightforward processing methods taking advantage of the main memory only. To demonstrate the problem of limited memory and high cardinality we present the weaknesses of an eager method. To keep the memory and the run time manageable we introduce and analyse two out-of-core algorithms: Periodic Partial Result Merging and a K-way Merge based method. Beside the core version of the algorithms above, optimalized versions are suggested to meet the performance requirements. Using these versions the processing of the dataset is relatively fast, fault-tolerant and the memory usage remains under complete control even on an average PC, which is unachievable by eager approaches.

Keywords: out-of-core algorithms, Periodic Partial Result Merging, k-way merge, performance analysis, eager method, weblog mining

1 Introduction and Motivation

The aim of data mining related tasks is knowledge discovery, which is revealing structure or rule extraction from large unknown dataset. Globally a lot of data are stored in different databases or on different servers in a raw form. During everyday workflow it is easier to store the data at a lower abstraction level and to transform it on demand to a higher abstraction level during an analysis. Regarding the different phases of data mining task this transformation, the data preparation step, can cover a high percentage of the whole process time. Thus this step is a key point in speeding-up the data mining process.

Log analysis is a frequently applied technique when trying to derive useful information from web traffic. Logs store elementary data record containing information about various facts (e.g. user identifiers or the timestamp of a request). From these logs the activity of various users can separated and identified with cookie-based techniques [1], [2]. After identification user behaviour profiles can be created based on the information belonging to each specific user. Usually it is not the individual data of each user which is interesting, but the typical profile types hold knowledge valuable for web service providers. Our motivation is based on a real life project dealing with weblog processing [1]. The typical profiles in our scope are created from a huge dataset containing more than 6 billions of elementary records. Such cardinality in most of the cases calls for out-of-core processing methods. This paper focuses on out-of-core pre-processing algorithms that raise the abstraction level of datasets by aggregation. Our raw dataset contains user identifiers and timestamps, which have to be transformed into a more complex structure, called temporal profile by aggregating the clicks belonging to the same user.

The organisation of this paper is as follows. Section 2 enumerates the general requirements of large scale data processing and reveals the necessity of using out-of-core processing methods by showing the limits of in-memory approaches. Section 3 presents the two algorithms and their analysis. Periodic Partial Result Merging and a K-Way Merging use the background storage in different way to overcome the size limitations of the main memory. Section 4 shows the performance analysis of algorithms using real data, while last section summarises our work and presents the possible further works.

2 The Necessity of the Out-of-Core Processing

Efficient handling of real datasets with vast cardinality and information extraction from them is a challenging technical problem even for today's high-performance computers. Filtering, transforming, processing this datasets and its visualisation requires other methods than the ones used commonly in information systems.

Our dataset, like many other large datasets, is created in an automated manner, continuously. Thus is essential for any out-of-core processing algorithm to ensure a higher processing speed than the data creation speed, which helps to avoid the agglomeration of raw data. This implies that every procedural step should preferably have a linear, or nearly linear runtime complexity. As there exist several problems with higher than linear runtime complexity, thus the above requirement cannot be satisfied in every case.

Another factor, which has to be taken into consideration in any out-of-core algorithm, is the higher access time of external storages against main memory. Due to this, keeping I/O instructions at a minimum level is a requirement for an efficient out-of-core method. If we can keep the number of storage accesses linear in number of points we can avoid data extrusion, getting a nearly so effective algorithm like the one mentioned in previous paragraph. This means that such algorithms will be effective, which read only constant times the input dataset (preferably once).

Beside these there are other factors which can increase runtime performance of out-of-core approaches. Cache efficiency could mean a performance increase, while the parallelisability can result further achievement in performance.

In order to illustrate the weaknesses of methods using only the main memory, thus the need of an out-of-core approach, we present an eager, in-memory algorithm and its analysis. Due to high cardinality a data container is needed that ensures an efficient data lookup time, because we have to find one specified element among millions of others. To solve this problem a hash table was used. The eager algorithm stores this hash table in the main memory, it is extended and modified continuously during the processing; and the content of memory is only saved to the disk after finishing the whole dataset.

The processing is fast, but the increasing memory demand is the main disadvantage of this algorithm, which makes it applicable only on smaller datasets. When the physical memory reaches its end and virtual memory paging start to dominate the runtime, this soon gets unmanageable long.

When handling large datasets we have to count with an immense runtime, which could be expressed better in hours and days, than in seconds. Thus unexpected errors or critical updates can stop the processing phase. The eager method is sensible for such aborts, because the already processed results, stored only in memory, get lost. This fact suggests a further weakness of this approach and the need of fault-tolerance in out-of-core approaches.

The presented disadvantages of this method stress the need for other methods, with controlled memory consumption. Out-of-core approaches mean a possible solution for this problem.

In order to process large datasets and to do it effectively we present the Periodic Partial Result Merging and a K-Way Merging based algorithm. These algorithms follow the partitioning principle, where the input dataset is split into smaller blocks to fit in the main memory [4]. In case of partitioning all the input records are used, the complete dataset is covered and each element is used only once in a partitioning. The partitions can be subsequent or arbitrary groups of records. After processing every partition, local results are written to the disk and the global result is created by merging the local results.

The creation of global result can follow different principles: in some cases the global result is only the union of the local ones [7], [8], [9]. In other approach a simple merging has to be done on the local data to get the global data. [4], [5]. There are cases when a more complex algorithm is used to get the global result [10].

A partitioning algorithm requires the size of blocks to be determined somehow. Basically, the successively equal-sized partitioning is a trivial, but working method [4], [5], [6], although there are cases when a sophisticated partitioning approach increases the out-of-core processing performance [8].

3 Out-of-Core Data Handling Approaches

In this section two out-of-core algorithm will be discussed and analysed: the first one is the Periodic Partial Result Merging algorithm, which creates the global result by continuous processing of blocks and their merging; the second one is a K-Way Merge based method, when the final result is generated using a merging tree, propagating processed blocks on different levels, getting closer to the global result.

3.1 Periodic Partial Result Merging

As first step the input dataset has to be split horizontally in equal chunks. Let's assume the cardinality of dataset to be n and we split it into chunks containing m records, which results s pieces of blocks. On an m-sized block the processing step (calculations) is done, which results the so called local partial result. The first local partial result is the global partial result too. The elements of the partial result have to be ordered in a way, which ensure a linear merge run time complexity with other elements. As next step the novel local partial result is merged with the partial global result. Due to ordering this can be done in effective way reading the records of two datasets only once. In the merging step there are elements with no impact on each other, they have to be written out without any modification, while from the elements with impact on each will be created a new element. After the merging step finished a new global partial result will be created, while the previous results have to be deleted. The last partial global result is regarded as the global result.

The merging phase of periodic partial result merging algorithm is presented in the Figure 1.

In order to keep I/O access at a lower level we can process more blocks at the same time in the memory, thus obviating some I/O accesses. This block-based modification of the presented algorithm makes it more effective in run-time. But the number of blocks processed in the main memory at the same time is a very sensible parameter of this modified approach and hard to determine in advance. To avoid virtual memory handling caused paging, we have to control the memory need of the algorithm, processing only so many data which fit in main memory. This modification results the so called adaptive periodic partial result merging. The performance tests suggest the efficiency of this approach.

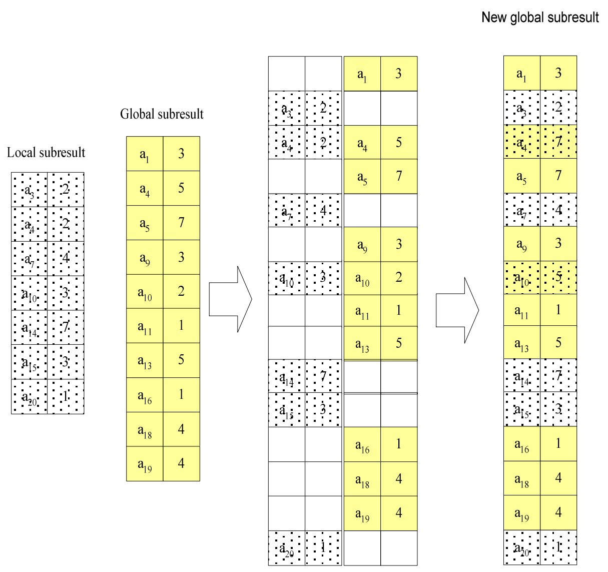


Figure 1
The sketch of merging phase

Analysing the runtime complexity of the Periodic Partial Result Merging algorithm, basically two main components has to be calculated: the complexity of I/O instructions and complexity of the processing. The I/O cost of the algorithm consists of reading the records from local result and from the global result and writing out the new result to the storage. Assuming that processing *x* records of data generates *αx* amount of result (usually *α*«1), the cost of processing block *j* can be calculated using the following equation:

$$k_{disk(j)} = k_{data\,read(j)} + k_{result\,read(j-1)} + k_{result\,write(j)} k_{disk(j)} = m + m(j-1)\alpha + mj\alpha = m(1+\alpha(2j-1))$$

Thus, the total I/O cost complexity of the algorithm can be given it the following way:

$$k_{disk} = \sum_{j=1}^{s} k_{disk\,(j)} = \sum_{j=1}^{s} m(1+\alpha(2j-1)) = ms + m\alpha\sum_{j=1}^{s}(2j-1) = ms + m\alpha s^2 = n + n^2\frac{\alpha}{m}$$

In order to achieve nearly linear disk cost the shrinking factor (α) has to be a small enough and the size of blocks (m) large enough.

The total processing cost can be evaluated using following formula:

$$k_{proc} = \sum_{j=1}^{s} k_{proc\,(j)} = \sum_{j=1}^{s}\left(f(m) + jm\alpha\beta\right) = sf(m) + \frac{s(1+s)}{2}m\alpha\beta$$

$k_{proc} = n\dfrac{f(m)}{m} + \left(\dfrac{n}{m} + \dfrac{n^2}{m^2}\right)\dfrac{m\alpha\beta}{2} = n\left(\dfrac{f(m)}{m} + \dfrac{\alpha\beta}{2}\right) + n^2\dfrac{\alpha\beta}{2m}$, where *f(m)* denotes the time complexity of processing of m sized block. The given formula contains a quadratic member in number of points due to this out-of-core approach, but the multiplying factor can be small enough to minimise the effect of this member, because α, β are typically closer to 0 than to 1, and it is divided with the size of block.

The presented algorithm can be suspended or restarted without losing the previously processed results. This fault-tolerance is a very favourable property for processing large datasets.

There is a prerequisite to apply this algorithm: the disk demand of the global results has to be significantly smaller than disk demand of the records from the result is created, namely the $\alpha << 1$. If our dataset fulfil this prerequisite (typically most of the dataset fulfil this prerequisite) we can draw the conclusion that the presented algorithm can be effectively applied in out-of-core pre-processing.

3.2 K-way Merge Based Processing

In this approach the processing is divided into two steps. Our dataset is split into m-sized blocks like in previous approach. First the processing of input data blocks is done in a similar way presented in the description of eager method. In the main memory a hash based container is maintained to store the processed data. There is a crucial difference between the eager method and this: here only a small amount of data is stored at the same time in the main memory. The processed blocks are saved on the storage. The result is created from these chunks of processed data using a k-way merge. In order to keep the merging time manageable an ordering is needed on the processed data before saving, which ensures linear merging.

In the complexity investigation of this algorithm takes two factors into consideration: the I/O cost and the procedural cost. To create the processed version of input chucks we have to read all the data and write the processed data back to storage. This can be expressed using following formula: $k_{disk1} = (1+\alpha)n$ The merging phase can be presented using the merging tree. At a level in this tree during the merging phase the I/O cost consist of the reading of the created temporary results and writing of the merged results. Thus the total I/O cost is given by the following equation:

$$k_{disk} = (1+\alpha)n + \sum_{j=0}^{\lceil \log_k s \rceil + 1} \left(\alpha \left\lceil \frac{s}{k^j} \right\rceil \cdot m \cdot k^j + \alpha \left\lceil \frac{s}{k^{j+1}} \right\rceil \cdot m \cdot k^{j+1}\right)$$

$$k_{disk} \approx n(1 + 3\alpha - 2\alpha \log_k(m)) + 2 \cdot \alpha \cdot n \cdot \log_k(n)$$

This disk demand could be considered nearly linear (somewhere between linear and $n \cdot \log_k(n)$).

Beside disk demand the procedural cost affects the whole processing, too. This can be calculated as follows:

$$k_{proc} \approx n \cdot \log_k(m) + \alpha \cdot n \cdot (k-1)\log_k(n)$$

4 Performance Results

Figure 2 depicts the behaviour of the eager method in memory limited environment. It shows how important it is to take control of the memory consumption of an out-of-core processing. As it is shown the run time of the eager algorithm becomes several time higher when paging occurs, while all the periodic partial result merging versions have only normal increases in run time.

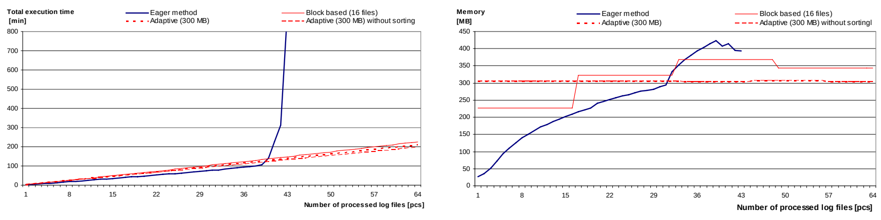
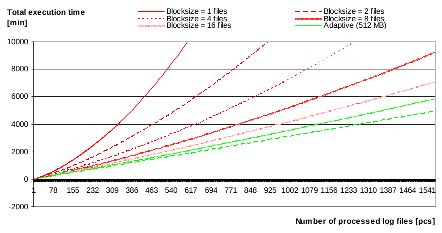


Figure 2
Runtime (left side) and memory consumption (right side) of eager method in memory limited environment compared with periodic partial result merging versions

Figure 3 presents different versions of periodic partial result merging: with varying size of blocks and the different adaptive versions. As the memory consumption graphics shows the faster the processing is, the bigger the memory demand of the process is. It is observable that the memory consumption graph of block-based versions of periodic partial result merging algorithm have an altering characteristic. This mean if we would increase the number of blocks processed at the same time, the memory demand could exceed the permissible bound and the run time of the algorithm get out of our control. As the graphic of memory consumption shows, the adaptive version of this algorithm ensures a stable, controlled memory consumption to avoid virtual memory handling induced paging.



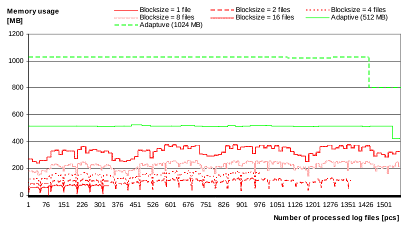


Figure 3

Run time (left side) and memory consumption (right side) of different periodic partial result merging version

Figure 4 shows the run time performance of merging phase on an "artificial" dataset. It is created from the real dataset, but all the overlapping is removed in order to see the correlation between number of levels in merging tree and runtime. This overlapping has an additional effect shown on the next figure. In this case the number of opened files (these blocks of input data are stored in files) means an other influencing factor. For small values of k the run time graphics is varying according to number of levels in merging tree (the broken line on Figure 4), but for larger k the processing becomes slower because of multiple reads from storage.

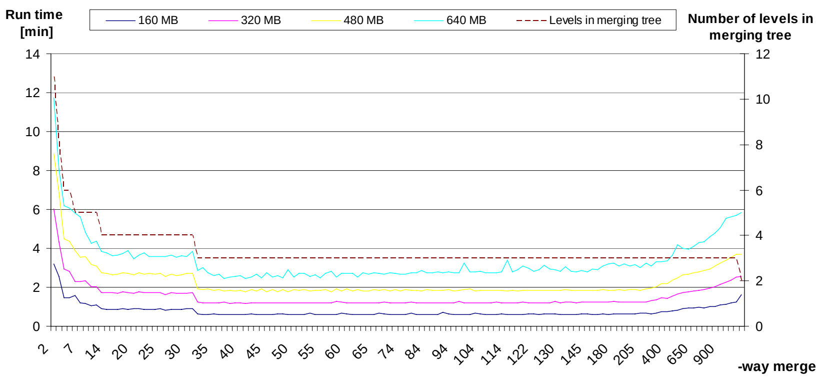


Figure 4

Run time of k-way merge based algorithm using artificial data and number of levels in merging tree

Figure 5 shows the run time of merging phase using real data, with overlapping (the element with the same id can be found in more blocks). As the previous figure suggests the k should be chosen from the neighbourhood of 32, because the merging time here is the lowest and the number of storage accesses are minimal. According to Figure 5 the choice of 128 would be optimal, which is significantly higher than in the previous case. This occurs due to the overlapping in dataset;

handling records at the same time in the main memory, which have to aggregated causes performance increasing, by avoiding multiple secondary storage accesses. Thus the runtime performance in real dataset is influenced by the three factors: number of levels in merging tree, number of I/O accesses and overlapping ratio.

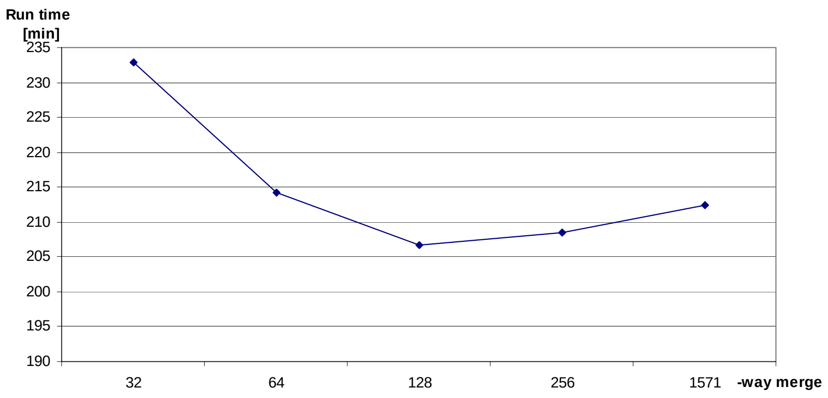
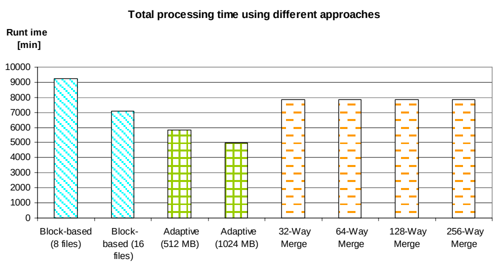


Figure 5
Run time of k-way merge based algorithm using a real dataset

Figure 6 is a summarising graphics showing the most optimal versions of different approaches on real dataset, containing more then 6 billions of records, while the comparative table suggests the advantages and disadvantages of different approaches. The presented version of eager method contains no fault-tolerance, while periodic partial result merging and the k-way merge based algorithm have it. According to experiments' results the eager method is applicable on smaller datasets, while the others on large datasets, too. Regarding the result of merging only the PPRM creates such local results, which can be taken as complete result of the processed blocks.



	eager	PPRM	k-way
Complexity	*n*	n^2	*nlogn*
Checkpoints	-	+	+
Dataset size	S	B	B
Result	E	C	E

Figure 6
Total processing time of different out-of-core approaches using a real dataset and a comparative table of techniques

Conclusions

In this paper we presented two approaches to process large amount of data efficiently, in a fault-tolerant way. Using a real dataset we have shown that eager approaches operating only in the main memory are not applicable. We presented optimalization possibilities for both of algorithms in order to keep memory consumption and run time manageable. The adaptive version of Periodic Partial Result Merging algorithm proved to be the most suitable to process large datasets effectively and in a fault tolerant-way, using even an average PC.

A further optimalization can be done in field of presented algorithms. A parallel version of Periodic Partial Result Merging could achieve gain in runtime performance. Detecting the optimal number of parallel processes, each of them realising the distributed version of Periodic Partial Result Merging algorithm and its influencing factor is a further research area.

Cache oblivious versions of the presented algorithms could result better runtime performances by optimalized cache handling. Thus a funnel-sort based approach could ensure better performances compared to k-way merge based approach.

References

[1] Iváncsy, R. and Juhász, S. 2007, *Analysis of Web User Identification Methods*, Proc. of 4th International Conference on Computer, Electrical, and System Science, and Engineering, CESSE 2007, Venice, Italy, pp. 70-76

[2] 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 Proceedings of the HUBUSKA Open Workshop on Generic Issues of Knowledge Technologies

[3] Sándor Juhász and Renáta Iváncsy: *Out-of-core Data Handling with Periodic Partial Result Merging* in Proc. of the IADIS European Conference on Data Mining 2009, part of the IADIS Multiconference of Computer Science and Information systems 2009, Algarve, Portugal, pp. 50-58, June 2009

[4] 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

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

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

[7] Grahne, G. Zhu J. 2004, *Mining frequent itemsets from secondary memory*, ICDM '04, 4th IEEE International Conference on Data Mining, pp. 91-98

[8] 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

[9] 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

[10] Tang, P., Ning, L., and Wu, N. 2005. *Domain and data partitioning for parallel mining of frequent closed itemsets*. In Proceedings of the 43rd Annual Southeast Regional Conference - Volume 1 (Kennesaw, Georgia, March 18 - 20, 2005). ACM-SE 43. ACM, New York, NY, 250-255. DOI= http://doi.acm.org/10.1145/1167350.1167423