Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Information Systems 29 (2004) 365–384

www.elsevier.com/locate/infosys

Memory-adative association rules mining☆

Alexandros Nanopoulos, Yannis Manolopoulos*

Department of Informatics, Aristotle University, Thessaloniki 54124, Greece

Received 4 July 2002; received in revised form 4 March 2003; accepted 16 April 2003

Abstract

New application areas resulted in an increase of the diversity of the workloads that Data Base Management Systems have to confront. Resource management for mixed workloads is attained with the prioritization of their tasks, which during their execution may be forced to release some of their resources. In this paper, we consider workloads that consist of mixtures of OLTP transactions and association rule mining queries. We propose and evaluate a new scheme for memory-adaptive association rule mining. It is designed to be used in the case of memory fluctuations, which are due to OLTP transactions that run with higher priority. The proposed scheme uses dynamic adjustment to the provided buffer space. Thus, it avoids the drawbacks of simple but naive approaches; namely the thrashing due to large disk accesses that can be caused by the direct use of virtual memory or long delay times due to suspension. Detailed experimental results, which consider a wide range of factors, indicate the superiority of the proposed scheme.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Association rules; Memory fluctuations; Resource management; Data mining

1. Introduction

The development of new application areas resulted in an increase of the diversity of workloads that Data Base Management Systems (DBMS) have to confront. On one hand, we have the important class of tasks that process transactions, which comprise the OLTP workload. On the other hand, there exist the class of tasks that process complex, long-running queries for Decision Support applications. The latter class comprises the DSS workload and has achieved legitimacy as a business necessity during the recent years. Each of these two types of workloads has its own objectives and requirements. Nevertheless, the natural question arises whether one can provide DSS access to the data stored for OLTP in the same DBMS. In other words, can the mixture of OLTP and DSS workloads lead to consistent performance, or is the OLTP workload prone to receive such a contention from the DSS one, that will compromise its mission-critical nature? The vice versa question is also of much interest: i.e., whether the performance of the DSS workload can remain acceptable during the mixture with the OLTP one.[1]

☆Recommended by Kenneth A. Ross, Area Editor.

*Corresponding author. Department of Informatics, Aristotle University, Thessaloniki 54006, Greece. Tel.: +302310996363; fax: +302310996360.

E-mail addresses: alex@delab.csd.auth.gr (A. Nanopoulos), manolopo@delab.csd.auth.gr (Y. Manolopoulos).

[1]Although data warehouses, that have been adopted to a significant extent, separate the two types of workloads, the

One of the important issues that one has to consider in order to successfully mix OLTP and DSS workloads, is how to best allocate resources (e.g., processors, memory, disk) between them. The governing of mixed workloads is attained through the prioritization of their tasks that compete for resources. Therefore, scheduling algorithms have to be used, which will dynamically decide the distribution of resources [1]. On the basis of this paradigm, the tasks in the OLTP and DSS workloads can be forced (during their execution) to release some/all of their resources, or in contrast they may be given additional ones [2]. Evidently, the definition of a priority scheme depends on the particular type of OLTP or DSS applications. A common case is to consider OLTP as mission-critical application type, therefore the OLTP workload will be assigned the highest priority. In contrast, there exist some cases where real-time Decision Support is needed, for which the DSS workload will get higher priority.

Data mining queries, and association rules mining queries in particular [3], correspond to a significant type of DSS tasks today. Their characteristic is that they are resource intensive, since they require large CPU time and main memory space, and (usually) several database scans. For this reason, a large number of stand-alone, specialized algorithms and data mining systems have been developed. However, such approaches lead to a loose-coupling with the DBMS, which incurs a significant overhead when mining large operational databases [4]. To address this problem, several methods have been proposed that achieve a tighter-coupling of association rules mining queries with the DBMS [4–6]. Nevertheless, the issue of mixing a workload that contains association rules mining queries, with a OLTP workload has not been examined by most of the aforementioned methods. Recently, Riedel et al. [7] proposed a disk scheduling algorithm for mixed workloads that consist of OLTP transactions and general data mining tasks (that can be easily applied to association rules). According to this approach, the OLPT workload provides a consistent portion of its bandwidth to a *background* data mining task, without impacting OLPT transactions. The prioritization scheme in [7] assigns the highest priority to the OLPT workload, while trying to advocate the DSS workload as much as possible at the same time. As described previously, this kind of prioritization corresponds a logical choice and applies to most cases of interest.

Besides disk, main memory is another critical resource that affects the performance of mixed workloads. However, this factor has not been examined so far in the context of mixed workloads that contain data mining queries and OLTP transactions. Several association-rules mining algorithms assume the existence of unbounded main memory. In contrast, there exist algorithms that take into account the case of limited memory and use specialized buffering schemes like the ones in, e.g., [8–10]. However, even these approaches assume the allocation of a *fixed* amount of buffer space to the mining query throughout its lifetime. Therefore, for the purposes of prioritized execution of mixed workloads (as described previously), none of these algorithms can adapt (during their execution) to possible requests for releasing memory back to the DBMS. Instead, they may hold an amount of main memory until their completion, whereas, at the same time, higher-priority OLTP transactions may not be able to execute due to memory shortage. Clearly, the existing approaches will violate prioritization schemes that consider OLTP as mission-critical, rendering mixed workloads unfeasible.

It is important to notice that the previously described issues form a new problem (c.f., Section 3), which calls for the development of schemes that will allow association-rule mining queries to dynamically adapt to memory fluctuations. This is analogous to the introduction of memory-adaptive schemes for external sorting [2], which differ from regular external sorting algorithms (i.e., those not considering memory fluctuations). However, the problem of memory-adaptive sorting is much different in comparison to the one

(*footnote continued*)
question whether mixed workloads can run against the same DBMS, still worths attention. The reason is that in this case, data will be directly available to DSS queries, thus their copying to the warehouse and their maintenance to assure consistency can be entirely avoided.

examined in this paper. Moreover, the development of dynamic memory-adaptive schemes for association rule mining differs significantly from the problem of developing efficient algorithms for the stand-alone execution of association rule mining (i.e., when the queries are not mixed with OLTP transactions), which has been the focus of recent research. The reason is that in the context of mixed workloads, the execution of mining queries may be impeded by OLTP transactions with higher-priority, and the buffer space provided to these queries will not be constant. Therefore, the execution time of a mining query does not depend solely on the efficiency of the corresponding (designed for stand-alone execution) algorithm that is used for the query. The effectiveness of the memory-adaptive scheme, the use of which is necessary in this context, is also of equal importance, as it determines how the execution of the query proceeds under memory fluctuations. Hence, the role of the two aforementioned factors is complementary. Moreover, simplistic memory-adaptive schemes, like the suspension of the query until sufficient memory is available or the direct use of virtual memory, are not a viable solution. As will be shown, such simplistic schemes impact the efficiency even of the best stand-alone mining algorithms, since they lead to large delays or thrashing.[2]

1.1. Contribution and layout

In this paper, we are interested in mixed workloads consisting of association rule queries and OLTP transactions, where the latter are considered as mission-critical. Therefore, through the use of a prioritizing scheme, memory fluctuations occur that move buffer space from the mining queries to the OLTP transactions (and vice versa). The main objective of the paper is the development of memory-adaptive schemes for association rule algorithms, which will allow them to handle memory fluctuations. To the best of the authors knowledge, no previous work has addressed this issue so far, which emerges when considering the described context of mixed workloads.

The technical contributions of the paper are the following:

- An effective memory-adaptive scheme, which compared to simplistic ones, does not incur large delays or thrashing (in terms of disk I/O). Therefore, the proposed method can be used towards the problem of making mixed workloads feasible.
- The description of ways to apply the proposed scheme to a significant class of association rules mining algorithms, especially those designed for tight-coupling with a DBMS.
- A detailed experimental comparison, which considers a wide range of factors. The experimental results help to understand the performance tradeoffs of each scheme and to identify the advantages of the proposed one. Moreover, they show that the proposed method compares favorably against simplistic schemes even when the latter ones are combined with very efficient (in terms of stand-alone execution) mining algorithms, a fact that indicates the new requirements of the examined context.

The remainder of this paper is organized as follows. Section 2 presents the related work and Section 3 contains the problem description. In Section 4, we develop a new algorithm that addresses the problem of limited buffer size and serves as the framework for the development of the proposed memory-adaptive scheme. The latter is given in Section 5, along with two other schemes that are given for comparison purposes. The results on the performance evaluation of the described schemes are illustrated in Section 6. Finally, Section 7 provides the conclusions.

2. Related work

The problem of association rule mining has been proposed in [3] and since then, a large number of algorithms have been developed to address its

[2]Also, it has to be considered that memory fluctuations cannot be circumvented by simply pre-assigning very large buffer space to the mining query in an aggressive manner (so as to never run out of memory space and to avoid the need for a memory-adaptive scheme), since this will burden the OLTP workload.

different aspects. These algorithms can be generally categorized in two paradigms, according to the way they prune the search space and perform pattern generation. The first paradigm is denoted as *Candidate set Generation and Test* (CGT), which is based on the iterative generation of a set of candidate patterns, followed (at each iteration) by the counting of their support. The basic pruning of the search space is done with the Apriori-criterion [8], but more recent CGT algorithms use several additional criteria. The CGT paradigm contains a considerable number of algorithms, for instance [8,11–14]. Although CGT algorithms use a diversity of techniques, they are essentially based on the same core of iterative candidate generation and testing, whereas their differences stem from the specific criteria and methods they use for these procedures.

The second paradigm consists of recently proposed algorithms, like the FP-Growth [9], TreeProjection [15] and H-mine [16], which entirely avoid the generation and testing of candidates. Instead, they are based on pattern fragment growth, using complex data structures. We refer to this paradigm as *Pattern Growth* (PG). Experimental results in [9] show that TreeProjection has the overhead of transaction projection and is outperformed by FP-Growth, whereas H-mine is a hybrid algorithm that develops certain optimizations for FP-Growth. However, it is allowed to switch to FP-Growth, to overcome shortcomings that the latter avoids.

The data structures that are used by PG algorithms, require a significant main memory size (e.g., the FP-tree [9]). This cost pays off in cases of dense data and patterns with large numbers of items, or for very low minimum support thresholds. Experimental results in [9] illustrate the advantage of FP-Growth over Apriori [8] in these cases. It is worth noticing that the CGT paradigm contains algorithms that also address such cases and outperform Apriori; for instance, the algorithm in [17] is efficient for mining dense, correlated data, or the algorithms in [11,12], which significantly improve Apriori for low support thresholds (by reducing the number of candidates). Nevertheless, a generalized comparison between PG and CGT algorithms has not been performed.

A different categorization of association rule mining algorithms can be done according to the database layout. The first category uses *horizontal* layout, where each transaction is a list of items. This category contains the majority of existing association rule mining algorithms. The second category contains the algorithms that use *vertical* layout, where each column corresponds to an item. MaxClique [18] was the first algorithm that proposed the vertical layout, whereas a recent algorithm in this category is Viper [19]. As described in [19], the disadvantage of MaxClique is that it requires large main memory.[3] In contrast, Viper does not present this drawback, and it compares favorably with MaxClique and Apriori. Although Viper significantly outperformed Apriori, it was not compared with other CGT algorithms.[4] Nevertheless, experiments in [19] (the comparison with the Oracle method) indicate the advantage of the vertical-layout approach, in general.

All the aforementioned algorithms were mainly proposed for stand-alone execution in a specialized system, i.e., not in the context of a DBMS. The integration of association rule mining with a DBMS requires the consideration of several additional issues, like query optimization, cost estimation, etc. For this reason, focus has been given by other works on the tight-coupling of association rule algorithms with Relational DBMS, using mechanisms like stored procedures or user-defined functions [4–6]. Thomas and Chakravarthy [5] have studied the optimization of joins for the case of support counting in RDBMS. Geerts et al. [20] provide upper bounds on the number of candidate patterns, which can be used for optimization purposes (an issue of vital importance for a RDBMS). It has to be noticed that, up to now, most of the approaches for integrating association rule mining with a DBMS have been based on algorithms from the CGT paradigm.

[3] A clique must be entirely held in memory, whereas the recursive decomposition results to large disk traffic [19].

[4] Based on [19], at a macro-level, Viper can be classified as CGT, since it is multi-pass, proceeds bottom-up, and support counting is performed in each pass. However, it has distinguishing differences from the other members of the CGT paradigm.

Although the latter works consider the integration with a DBMS, they did not focus on mixed workloads. Riedel et al. [7] described one of the first approaches to addresses mixed workloads of data mining queries and OLTP transactions. They propose a disk scheduling policy for general data mining tasks and differently from [1,21] (which considered general DSS tasks), they assume that OLTP transactions are the most crucial tasks of the DBMS, thus they are given the highest priority. Therefore, the disk scheduler serves: (a) requests for OLTP transactions that are satisfied as soon as possible and (b) requests for one or more data mining queries that are satisfied when convenient.

Evidently, the size of available memory is crucial for association rule algorithms. Several algorithms assume the availability of all required memory, since they focus on stand-alone execution. However, there exist approaches that take into account the case of limited memory; for instance, the buffering methods in [8,10] for CGT algorithms, or the partitioning methods in [9,16] for PG ones. Nevertheless, as described (see Section 1), the latter approaches cannot handle memory fluctuations, because they do not dynamically adapt to varying memory size and may keep a fixed size of reserved memory whereas, at the same time, OLTP transactions may lack of necessary memory.

Besides the mining of regular association rules, other works have studied more specialized problems like the mining of long patterns, for instance [22], which is based on the CGT paradigm but finds only the maximal frequent itemsets ([22] describes the use of a randomized, incomplete algorithm, in order to search for the regular association rules). Such approaches mainly focus on databases that contain very long patterns.

Finally, related work includes the management of resource allocation among different types of workloads [21,1,23]. These works have focused on the combination of large, relational join queries with small OLTP transactions, and on how to allocate resources (main memory, disk) to these tasks. The latter works, however, did not consider data mining queries, which are very different from relational join queries. Faloutsos et al. [24] described flexible buffer allocation, but they did not focus on diverse workload types. Pang et al. [2] have examined memory-adaptive external sorting algorithms that address memory fluctuations. This work clearly provides a motivation to our approach, but the problem of sorting is very different from the mining of association rules.

3. Problem description

The context studied in this paper considers mixed workloads consisting of association-rule mining queries and OLTP transactions, where the latter ones are mission-critical (i.e., take the highest priority). For this reason, under a prioritization scheme, main memory can be dynamically reallocated between the association rule queries and OLTP transactions. Therefore, the size of buffer memory provided to association rule queries fluctuates during their execution, and the queries may be forced to release an amount of their buffer space to the DBMS. Considering that association rule queries can be memory intensive (e.g., for low support thresholds), they may come to a point where they lack sufficient memory. This problem has not been previously studied. As described in Section 2, existing algorithms assume that (a) either all required (i.e., unlimited) memory is available, or (b) a constant (i.e., limited) buffer space is provided. Nevertheless, as explained, both cases may violate the required prioritization scheme under repeating memory fluctuations, since they do not address the need to release buffer space back to the DBMS. Moreover, they cannot take advantage of possible extra buffer space that may be provided to them from the DBMS.

What is, therefore, needed is dynamic schemes that will allow association rule queries to become memory-adaptive. These schemes will decide how the execution of the association rule algorithm will proceed at the time points that an amount of main memory must be released from the query to the DBMS, or how the algorithm will exploit extra memory that is given to it. Evidently, none of the existing algorithms for association rule mining can be directly used in the examined context (in order not to violate the described prioritization scheme), unless it adopts a memory-adaptive scheme. In particular, the roles of the memory-adaptive scheme and the association rule algorithm are complementary. This is because they address different requirements. Existing association rule algorithms have been designed for stand-alone execution (i.e., not for memory fluctuations) and contain optimizations for this case. However, in the examined context, their execution times will not depend only on how efficient they are in the stand-alone case, but also on the effectiveness of the memory-adaptive scheme that will be used, since a not good memory-adaptive scheme can render deficient any good algorithm. Therefore, the examined problem is not to develop an association rule algorithm for stand-alone execution, but to propose effective memory-adaptive schemes that will help towards the objective of having viable mixed workloads in the same DBMS.

To determine the examined execution model, we assume an integration of association rule mining queries within the DBMS based on extensions like the user-defined functions (UDF) [4]. In this context, the mining query is provided with buffer memory allocated in the address space of the DBMS (unfenced option [4]), and this buffer memory is subject to memory fluctuations throughout the execution of the mining query. Following the approach of [7], the execution of association rule queries is performed in the background, whereas the highest priority is given to OLTP transactions that run in the foreground and cause the memory fluctuations.

For the purposes of the examined problem, we mainly focus on CGT association rule algorithms. As described in Section 2, the tight-coupling with a DBMS has been examined in a larger extent for CGT algorithms, e.g., [4–6,20,25], and this factor is important when considering mixed workloads in the same DBMS. Since there exist more advanced CGT algorithms than Apriori, we generalize our approach to work with such algorithms (Section 4.2). Clearly, it is worth developing memory-adaptive schemes for other types of association rule algorithms as well, like PG or vertical-layout algorithms. However, since the roles of the association rules algorithms and of the memory-adaptive schemes are complementary, we consider CGT algorithms as a good starting framework to test the viability of the latter in the examined context of mixed workloads. Nevertheless, for comparison purposes, we examine the combination of a PG algorithm (FP-Growth) with less advanced memory-adaptive schemes (that can be directly applied to this algorithm), to demonstrate that, in the examined context, the execution time of association rule queries depends significantly on the effectiveness of the memory-adaptive scheme. Evidently, one may expect that the development of dynamic memory-adaptive schemes that can be combined with other, non-CGT algorithms will improve further the execution times. For this reason, this topic is considered as an interesting future work.

4. The framework algorithm

In this section, we describe the framework algorithm upon which we will develop the proposed memory-adaptive scheme (that is given in the following section). The framework comprises a basic algorithm that takes into account the case of constrained available memory. First, for reasons of clarity in presentation, we develop a basic form of the framework in terms of the general structure of Apriori algorithm, and then we describe ways to extent it to other, more advanced algorithms of the CGT paradigm.

4.1. *Basic form of the framework*

Let that at a given time point the available main memory allows for the accommodation of m candidates. Memory fluctuations cause the decrease or increase of m during the execution of the association rule query. Although [8] proposes a buffering method that can be followed by CGT algorithms, this method proceeds the merging of a phase only if all its candidates can be kept in memory, a fact that can cause memory under-utilization. The reason is that, although the main memory may be sufficient to hold a large part (but not all) of the candidates during a phase, it is left under-filled. Moreover, in the case of memory limitation inside a phase and the breaking of a candidate set into groups that are separately counted, the last group may not entirely fill the available main memory.

To resolve the aforementioned problems, in the proposed framework the handling of memory limitation within a phase could be directly followed by phase-merging, in order not to under-utilize the available main memory. Therefore, the cases of memory limitation (when not all candidates of a phase can be kept in available memory) and phase merging (when available memory can keep the candidates of more than one phases) are not treated separately. In the former case, we have to move on to phase-merging in order to exploit all available memory. In the latter case, the merging stops and the supports of candidates are counted. Then, we have to move on by treating the stopped phase as being under the case of memory limitation (since not all of its candidates can be held in main memory). Whenever the available memory is exhausted, the candidate generation procedure stops and support counting is performed. The resulting algorithm is depicted in Fig. 1.

In order to keep a synchronization in passing from one case to the other, the algorithm maintains some extra information for each node in the trie data structure (which stores the candidates). Additionally, we prefer to also keep in the trie the large itemsets (i.e., the candidates that have been counted and found large).[5] Of course, the large itemsets are taken into account in the number m of itemsets that are held in main memory. Each trie node has an associated information that denotes its status. A node that is 'active' denotes a candidate with support that has to be counted. A node that is 'extendible' corresponds to a counted itemset that was found large and can be extended to generate further candidates. A large itemset becomes non-extendible when it cannot generate any other candidates. In this case, as described, it is maintained in the trie but it does not further participate in the candidate generation procedure. Candidates that have been counted and found non-large are pruned and are not maintained in the trie.

```
L = {large items}
k = 2
do
{
    $C$ = CandidateGen($L, k$)

    //Memory limitation
    while $|C| == m$ {
        forall $t \in D$ {
            $C_t$ = subset($C, t$)
            forall $c \in C_t$
                $c$.count++
        }
        $L+ = \{c \in C \mid c.\text{count} \geq \text{MINSUP}\}$
        $C$ = CandidateGen($L, k$)
    }

    //Phase merging
    while $C \neq \emptyset$ and $|C| \leq m$ {
        $k$++
        $C$ = CandidateGen($L, k$)
    }
    forall $t \in D$ {
        $C_t$ = subset($C, t$)
        forall $c \in C_t$
            $c$.count++
    }
    $L+ = \{c \in C \mid c.\text{count} \geq \text{MINSUP}\}$
} while $C \neq \emptyset$

return $L$
```

Fig. 1. The basic form of the proposed framework algorithm.

Memory fluctuations can take place either when support counting is performed or during candidate generation. Assume that the association rule query has to release an amount of main memory during support counting. Also, let m' be the resulting number of trie nodes that can be stored in memory (where $m' < m$ after memory reduction). If $|C| > m'$, then memory shortage occurs. The required number of active nodes is stored on disk, so that m' candidates are being left in the trie structure and the resulting free buffer space is returned to

[5] Large itemsets are kept in the trie so as to have one memory-management scheme; otherwise, the set of large itemsets would require separate treatment.

the system. It must be noticed that, during support counting with forthcoming transactions, a further release of memory from the association rule query to the system may be required. In such case, the above procedure is repeated. The way of proceeding support counting after memory shortage depends on the particular memory-adaptive scheme that can be followed. Each scheme determines if the overall support counting procedure is stalled or not, and how the candidates that are moved to secondary storage participate in support counting. Specific schemes are described in detail in the following section. Moreover, to take advantage of the case when the reserved memory is given back to the association rule query, each scheme determines how the candidates from secondary storage are restored back in the trie.

Evidently, a paging mechanism is required to move candidates to secondary storage and release buffer space to the system. This mechanism is part of the general algorithmic framework that is described above, and it is common for all memory-adaptive schemes that will be described in the following.

When the association rule query has to release memory during candidate generation, assuming that for the new m' it holds that $|C| > m'$, then the number of 'active' candidates is reduced to m' and the extra generated candidates are moved to secondary storage. Therefore, in terms of the proposed framework, the case of memory limitation is handled by considering the m' candidates as a separate group for which support counting has to be performed. However, the way that extra candidates (those moved to secondary storage) are handled and how the procedure moves on to support counting, depends on the particular memory-adaptive scheme. In the opposite case, when extra memory is given to the association query, any candidates stored on disk are restored back in the trie and new candidates are being generated in order to fill all available memory. The adaptive association rule algorithm may have to move on from the last group of candidates of the memory-limitation case to merging with the forthcoming phase. This illustrates the advantage of the developed framework, which does not separate these two cases, in order to support memory-adaptive schemes and to utilize all available main memory.

4.2. Extension to other CGT algorithms

The basic form of the framework (Section 4.1) is based only on the high-level structure of the Apriori algorithm. An analogous structure is followed by most of other CGT algorithms, since they all perform iterative candidate generation and testing. Their differences come from the way that these procedures are implemented.[6] Nevertheless, the exact implementations of these procedures do not modify the CGT paradigm that all these algorithms are based on.

To extend the basic framework to ones that are based on other CGT algorithms, we do not have to modify the way they perform candidate generation and testing. The only required change is to add within these procedures, methods that will handle the changes in buffer size. As will be described (c.f., Section 5), the proposed scheme adds within the candidate generation procedure the moving of a number of candidates to secondary storage, when an amount of memory must be released, and then passes control to the candidate-testing procedure. This addition does not involve details like the exact pruning criteria, etc, used by the candidate generation procedure of the specific CGT algorithm. For the candidate-testing procedure, the proposed scheme uses a technique that handles the testing of candidates stored on secondary storage. This addition is included within the candidate-testing procedure regardless of the way that the latter is implemented (e.g., whether it operates on the entire database or a part of it, etc). Therefore, the extension to frameworks that use other CGT algorithms can be attained by the addition of the techniques used by the memory-adaptive schemes within the corresponding places of the candidate generation and testing procedures.

[6] Another issue is whether or not candidates of different lengths are allowed in the same iteration. However, this may not impact the high-level structure of the algorithms, since it uses a set of candidates no matter their lengths. Notice that the basic form of the framework uses phase-merging, therefore it can allow for candidates with different lengths in an iteration.

For instance, consider some known CGT algorithms like DHP [12], Partition [13], or the algorithm that is based on sampling [14]. DHP reduces the number of candidates with a hashing technique, and transaction trimming is related to the specific support-counting procedure. As explained previously, these techniques are encapsulated in the candidate generation and testing procedures and the memory-adaptive scheme will not have to modify these techniques. For Partition, the memory-adaptive scheme will first handle the candidate set in each partition according to the basic framework, and then the same will apply for the second phase as well (when testing against the entire database) for the union of candidates selected from each partition (i.e., candidates found frequent in at least one partition). In a similar way the memory-adaptive scheme will handle the candidate sets generated by the algorithm of [14], both in the first (when testing against the sample) and second (when testing against the entire database) phases. Analogous reasoning can be applied to other CGT algorithms, e.g., [11] and [17] (for its first phase in particular, which finds the frequent closed candidates and is the most computationally intensive).

Therefore, the proposed scheme (that is detailed in the following section) can be combined with the broad family of CGT algorithms. For purposes of illustration, we selected as representative cases the extension to the frameworks that are based on the algorithms of [12,14]. The performance results for these methods are reported in detail in Section 6.

5. Memory-adaptive schemes

In the previous section we described the algorithm-framework for the development of memory-adaptive schemes. These schemes determine the different policies that can be followed to handle memory fluctuations. For comparison purposes, we first describe two simplistic approaches and next we develop the proposed dynamic scheme.

5.1. Suspension

Consider the case of memory shortage during the support counting stage of the association rule query. As described, a number of candidates is moved to secondary storage, and buffer space is released to the DBMS. A straightforward approach to proceed from this point is to suspend the execution of the association rule query. When enough memory is given back to the query and all the moved candidates can be restored back to main memory (in the trie structure), then support counting continues. The only information to resume the procedure is the position of the transaction in the database at which support counting was suspended. While the query is suspended, it may be required to release to the DBMS even more buffer space. This can further increase the suspension time.

When memory shortage occurs during candidate generation, the suspension scheme moves candidates to secondary storage (so as to release the required buffer space) and suspends the generation of any additional candidates, until enough memory is given back to the association rule query.

Although the aforementioned issues correspond to the use of the suspension scheme by the CGT paradigm, it has to be noticed that it can be easily modified so as to apply for non-CGT algorithms as well. The reason is that it does not take any actions besides the simple suspension of the mining query, which can be done regardless of the specific type of the association rule algorithm that is used by the query.

5.2. Paging

In order to avoid suspension in case of memory shortage, one could directly use a paging scheme (virtual memory). In fact, this scheme can utilize the paging mechanism that was described in the framework algorithm of the previous section, which moves candidates to/from the secondary storage. Let us focus on memory shortage during support counting. After a number of candidates has been moved to disk, the paging scheme can continue the support counting by considering the candidates stored on secondary storage as being ‘normal’ entries in the trie structure. Pointers in the ancestor nodes of such trie nodes will refer to their address on disk. The direct probing of a trie that has several nodes stored on disk, evidently, can result to a prohibitive cost. Therefore, we introduce an LRU buffer to cache the frequently accessed nodes that have been moved to disk. It has to be noticed that the size of the LRU buffer is taken into account together with the remainder main memory that is used by the trie structure. Thus, it results in a reduction of the memory provided for normal (i.e., stored in main memory) trie nodes. When memory is given back to the association rule query, the corresponding number of trie nodes are restored back to main memory (also, they are flashed out from the buffer).

The paging scheme can easily handle repeated memory requests from the DBMS. Each time, the association rule query moves nodes to disk and releases the requested main memory. Considering memory shortage during the candidate generation stage, the paging scheme stops generating any more candidates and, differently from the suspension scheme, it does not wait until enough memory is given back. Instead it treats the generated candidates as a separate group of candidates (see the framework algorithm in the previous section) and proceeds to their support counting.

As in the case of the suspension scheme, the paging scheme can be adapted to be combined with non-CGT algorithms as well. However, this adaptation is not as straightforward as in the case of the suspension scheme, since it has to consider the specific representation of patterns in main-memory (e.g., the used data structures, etc), so as to apply their paging. Nevertheless, following the specific details of each approach, such an adaptation is possible.

5.3. Dynamic scheme

The proposed scheme for memory-adaptive association rule mining follows a different approach in comparison to the suspension and paging schemes. Its objective is to dynamically adapt to memory fluctuations, neither by stalling the mining query nor by having to directly resort to virtual memory. Therefore, as it will be discussed in the following, it avoids the drawbacks of the previously described schemes.

Suppose that memory shortage occurs during support counting. The dynamic scheme proceeds as follows:

1. The required number of candidates is moved to disk, based on the framework algorithm.
2. The dynamic scheme marks those candidates as ‘moved’,[7] which denotes that these nodes have been moved to disk at least once. Also, let t_m be the transaction ID number at which the memory shortage occurred (i.e., the one for which the support counting procedure would probe the trie before memory shortage). The dynamic scheme stores t_m along with each moved candidate.
3. Support counting continues regularly. However, differently from the paging scheme, the candidates that have been moved to disk do not participate, i.e., their support is not being counted while they are stored on secondary storage.

When the association rule query is requested to release more memory while being in memory shortage, more candidates have to be moved to disk. In contrast, when memory is given back to the query during support counting, a number of candidates can be restored back to main memory. Let one such candidate be c. Also let t_m be the transaction ID at which c was moved to disk, and t_r be the current transaction ID (at which c is restored). The dynamic scheme increases the (so far) counted support of c by $t_r - t_m$ (assuming that for the t_r transaction we can perform support counting for c).

Lemma 1. *The dynamic scheme does not produce false negatives* (*i.e., false-dismissals*).

Proof. Let c be a candidate that remains on secondary storage during support counting for transactions with ID ranging from t_m to t_r. Although its support is not being counted for these intermediate transactions, it cannot be larger than $t_r - t_m$, since the latter corresponds to the

[7] ‘Moved’ is an extra option for the status information that is kept for each candidate, as described in Section 4.

case where all transactions from t_m to t_r contain c (i.e., it is the upper bound for the support increase of c for these transactions). Let $s'(c)$ be the counted support of c (using the previous method) and $s(c)$ its actual support. Hence, $s'(c) \geqslant s(c)$. If $s(c) \geqslant \text{MINSUP}$, then $s'(c) \geqslant \text{MINSUP}$. Therefore, no large itemset is missed, and the dynamic scheme does not produce false negatives. □

When support counting (for one or more phases) is finished, then the large candidates are determined (see the framework algorithm of Section 4). If, under the dynamic scheme, a candidate has support larger than MINSUP, then we have to examine if it had been moved at least once to disk (i.e., we test if its status information denotes 'moved'). If it had not been moved, then it is a large candidate. Otherwise, we cannot decide whether it is actually large or not, since its actual support may be smaller than MINSUP. In this case, the support of the candidate is reset to zero and it remains in the trie structure as a normal candidate, so as to recount its support (also the 'moved' status is cleared). In contrast, when the support of a candidate is less than MINSUP, then it is pruned regardless if it had been moved to disk. Based on the aforementioned issues, we can prove the following.

Lemma 2. *The dynamic scheme does not produce false positives.*

Proof. Let c be a candidate that had been moved at least once to disk. Also, let $s'(c)$ be its counted support under the dynamic scheme and $s(c)$ its actual support ($s'(c) \geqslant s(c)$). If $s'(c) \geqslant \text{MINSUP}$, it does not necessarily hold that $s(c) \geqslant \text{MINSUP}$. For this reason the support of c has to be counted again. Thus, if $s(c) < \text{MINSUP}$, the dynamic scheme will not report c as large. In contrast, if $s'(c) < \text{MINSUP}$, then it holds that $s(c) < \text{MINSUP}$. Hence, c is correctly pruned by the dynamic scheme and will not be reported as large. Therefore, the dynamic scheme does not produce false positives. □

With Lemmas 1 and 2 it is shown that the dynamic scheme generates all (no false negatives) and only (no false positives) the large itemsets. It has to be noticed that, when support counting has been terminated, several candidates may remain on secondary storage due to lack of buffer space. While candidates are being pruned as non-large, buffer space is freed. The candidates that are stored on disk, are read using this freed buffer space,[8] and their support is updated according to the dynamic scheme (using the difference of transaction IDs), so as to examine if they can be pruned or not. Evidently, since these candidates are read from disk, they cannot actually be determined as large because their status will definitely be denote as 'moved'.

Regarding memory shortage during candidate generation, the dynamic scheme follows the same approach as the paging scheme. The candidate generation procedure stops generating any additional candidates. It treats the generated candidates as a separate group of candidates (see the framework algorithm in Section 4), and then proceeds to the counting of their support.

5.4. *Qualitative comparison*

The suspension scheme is simple to implement. However, it has the drawback of introducing possibly long delay times, due to stalling. Depending on the system load, i.e., how often OLTP transactions cause memory shortage to the association rule query, the total execution time of the query can be significantly impacted.

Although the paging scheme does not introduce delay times, it may cause trashing, i.e., the rapid increase of accesses to secondary storage. Depending on the system load, a large number of trie nodes may be moved to disk. Moreover, it has to be considered that the access-patterns (in terms of the trie probing) during support counting can be quite scattered. Therefore, the LRU buffer may not always be able to prevent thrashing, and the execution time of the association rule query can be severely impacted. If we use a very large LRU

[8] In the case that no buffer is released by pruning, then the candidates on disk are read using a single page buffer (that has to be requested from the DBMS), and then are stored back to disk.

buffer, then the number of trie nodes that can be kept in main memory will reduce substantially (since the total available main memory is the sum of the size of this LRU buffer and the size of the trie). Evidently, this does not present a clear solution, because it affects the number of candidates that can be generated in each phase (i.e., will result to the split of each phase in a large number of sub-phases). Thus, one can expect the paging scheme to efficiently handle small fluctuations (with respect to the magnitude of requested memory), since it will avoid suspensions with a small cost of few accesses to secondary storage. Moreover, for small fluctuations, the LRU buffer can be effective. However, for medium and especially for large fluctuations, the performance of the paging scheme is expected to degenerate.

The dynamic scheme, differently from the suspension scheme, does not introduce any stalling. When memory shortage occurs during support counting, it dynamically adjusts to the available memory by continuing the procedure for the remaining candidates. For memory shortage during candidate generation, it adjusts by proceeding to the support counting of the candidates that can fit in the available memory. Thus, delay times are avoided. Moreover, differently from the paging scheme, the dynamic scheme avoids the involvement of candidates that, due to memory shortage, have been moved to disk. Hence, it avoids the scattered disk accesses and the possibility of thrashing. It also takes advantage of memory that is given back to the association query, so as to restore moved candidates from disk to main memory by assigning to them the upper bound for their support. Conclusively, the dynamic scheme does not present the drawbacks of the other two schemes. The superiority of the proposed scheme is illustrated by the performance results that is given in the next section.

6. Performance evaluation

In this section we present the experimental results on the performance. All the described schemes where implemented in C, using common components. Henceforth we refer to the memory-adaptive schemes as follows: to the suspension scheme as 'SPND', to the paging scheme as 'PGNG', and to the dynamic scheme as 'DNMC'. In our measurements we considered the following factors: the magnitude and the rate of memory fluctuations, the duration of period for each memory release (memory given from the association rule query to the DBMS), the buffer size, and the sensitivity against support threshold and database size. We also considered the described extensions for CGT algorithms and we examined the impact that simplistic memory-adaptive schemes may have on a non-CGT algorithm. For our experiments we used a simulation model. For this reason, we first give its description and then we present the results.

6.1. Simulation model

As explained, we adopt the approach of [7] and assume that the highest priority is given to OLTP transactions, whereas the mining queries run in the background. For simplicity, since we do not wish to address concurrency control issues, the OLPT transactions are assumed to perform 'read-only' operations. Also, following the approach of [2] so as to clearly focus on the management of main memory resource, we do not examine the impact of contention on secondary storage (disk waiting queues).

The simulation of a model that addresses the aforementioned issues is described as follows. It consists of an association rule query, which is initially given an amount of main memory that can store M nodes in the trie structure. We assume that requests for memory releases (due to OLTP transactions) have inter-arrival times which follow exponential distribution with mean μ_{int}. By first running the association rule query without being subject to memory fluctuations, we get the total execution time in this case. We determine μ_{int} as a fraction of the latter execution time. Therefore, a smaller fraction denotes higher memory fluctuation rate, whereas a larger denotes a lower rate. The size of memory fluctuation, called magnitude, follows uniform distribution in the range from 0 to a *MemThresh* (this size denotes the number of nodes that have to be moved to secondary storage). The duration of each memory release also follows exponential distribution, where its mean μ_{dur} is given as a ratio of μ_{int}. When this ratio is larger than 1, the association query may terminate while some requests for memory release may be pending. In this case, when the query terminates, it releases to the DBMS all its working main memory. In all cases, the query termination determines its total execution time.

To have a strict control on the amount of available main memory, we do not wish to directly consider I/O operations with means of the operating system, since buffering performed by the operating system may provide extra main memory for avoiding I/O operations. For this reason, following an approach analogous to [2], we model the access to secondary storage by considering disk page equal to 8 kB, disk seek time equal to 10 ms and page transfer time equal to 2 ms. Moreover, the net CPU time of the process is counted (all schemes use the same programming components). The total execution time is the sum of the CPU and I/O time calculated by the simulation.

6.2. Basic comparison of memory-adaptive schemes

We used the basket-data generator of [8]. We examined several instances with respect to transaction length and pattern size. Herein, for brevity, we present the results for T10.I6.D100K, since the other cases gave analogous results with respect to the relative performance of the examined schemes.

Our first experiment examines the magnitude of memory fluctuations. The support threshold was 0.2% and the initial buffer size was set to hold 20% of the total number of candidates that were generated in the case when no memory fluctuations occur (see the simulation model). The fraction of mean interarrival time was set to 10% (corresponding to a relatively low system load). The ratio of mean duration time to mean interarrival time was 0.8. The magnitude of memory fluctuation depends on the value of *MemThresh*, which is given as a percentage of the initial buffer size. Fig. 2 illustrates the results with respect to the *MemThresh* percentage.

As shown, for low fluctuation magnitude all schemes present comparable performance. However, the performance of PGNG quickly decreases, starting after 5% magnitude. This is because larger memory fluctuations cause many candidates to be moved to disk. Since PGNG involves these candidates for support counting, as expected, it cannot avoid the thrashing that appears for larger magnitudes (more than 7%). In contrast, the other two schemes are not affected by the magnitude of memory fluctuations, since they avoid the involvement of candidates that are moved to secondary storage.

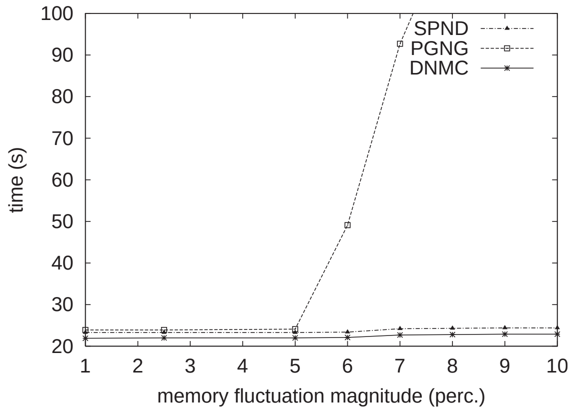

Fig. 2. Execution time vs. the mean magnitude of memory fluctuations.

We now move on to examine the impact of the duration of memory fluctuations. We used a small magnitude, equal to 1%, the fraction for mean interarrival time was set to 2.5% and the remaining parameters were the same as in the previous experiment. Fig. 3 illustrates the results with respect to the ratio of mean duration time.

SPND is clearly affected by the increase in mean duration time (i.e., larger fraction values). The reason is that when memory fluctuations have larger duration, SPND is forced to suspend the execution of the association rule query for larger periods of time. Thus, the total execution time is burdened significantly. In contrast, DNMC and PGNG are not affected by this factor, since they do not introduce any suspension during the processing of the association rule query.

From the above two cases it is evident that, as explained in Section 5.4, DNMC avoids the drawbacks of the other two schemes and combines

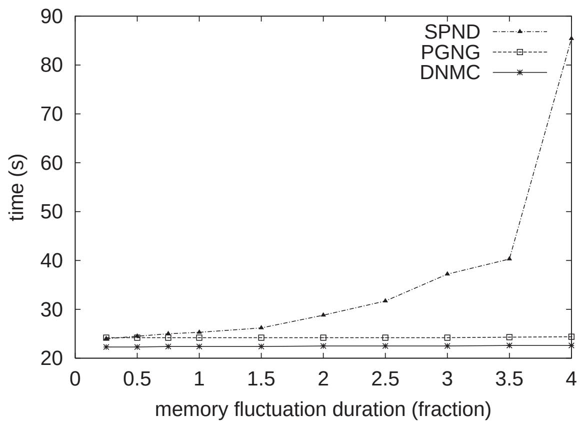

Fig. 3. Execution time vs. the mean duration time of fluctuations.

their advantages. Therefore, it offers the appealing characteristic of robustness against the aforementioned factors.

6.3. Sensitivity against parameters

Next, we examined the impact of the rate of fluctuations, given by the mean interarrival time. We used *MemThresh* equal to 5%, whereas the ratio of mean duration was set to 0.8 (both values are relatively small according to the previous two experiments in order to avoid the problematic cases for both PGNG and SPND). The other parameters were the same as the ones is the previous measurements. Fig. 4a gives the results with respect to the mean interarrival time (given as a percentage). As depicted, for very frequent rates (i.e., with small interarrival time) the effect is much more pronounced for PGNG. This is due to the large I/O traffic produced by PGNG, which is the result of the frequent (and random) access to candidates that are stored on disk. SPND is less impacted, because it does not require to access candidates that are stored on disk, but it is always significantly worse than DNMC, since it frequently has to suspend the execution of the association rule query. For rates that correspond to medium frequency of fluctuations, PGNG performs better than SPND, because in such rates the number of random accesses to disk is not as large, and pays-off since it avoids suspension. Nevertheless, PGNG still performs worse than DNMC, because DNMC entirely avoids the random accesses to disk-stored candidates. As expected, with decreasing frequency of rate all schemes converge to low execution times (i.e., since very few memory fluctuations occur, all schemes become equivalent).

To summarize all the aforementioned factors and examine their combined impact, we performed a measurement where we separated four cases with respect to the values of fluctuations rate (measured by mean interarrival time, where lower mean time value denotes higher rate), the ratio of mean duration (higher ratio denotes longer duration)

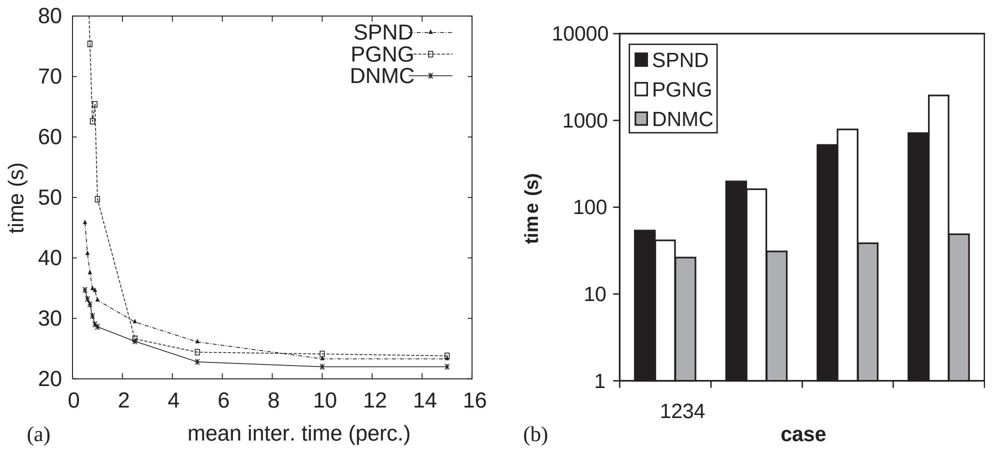

Fig. 4. (a) Execution time vs. mean interarrival time (%) of fluctuations and (b) execution times for the four separated cases.

and the magnitude. The four cases are described with respect to the parameter values (respectively) as follows: (1) 2.5%, 2.5, 4%, (2) 1%, 3.5, 5%, (3) 0.5%, 4, 5%, (4) 0.5%, 4, 8%. In brief, case 1 has relatively low values for the three parameters, case 2 has a significant increase in duration, case 3 has significant increase both in rate and duration, and case 4 has an increase in all parameters. The results are depicted in Fig. 4b (the time axis is plotted logarithmic). In case 1 DNMC performs better that the other two methods, but the performance differences, in general, are moderate, because this case corresponds to fluctuations that are relatively small, short, and not very frequent. In case 2 PGNG outperforms SPND, due to the significant increase in rate that impacts SPND. However, since magnitude in this case is not negligible, DNMC outperforms PGNG. Evidently, case 3 illustrates the drawback of PGNG when both duration and rate are increased, because (as described in previous experiments) in this case PGNG results to large I/O traffic. SPND comes second best, but it is also affected by the combined increase in both factors. Finally, in case 4, where all factors are increased, PGNG and SPND are outperformed by DNMC by an order of magnitude. This is because in this case, fluctuations are relatively long, large and occur often, a fact that significantly affects both PGNG and SPND. Therefore, as the values of all factors get high values, DNMC maintains its good performance, whereas the performance of the other two schemes degenerates. This makes DNMC to become the memory-adaptive scheme of choice.

We now turn our attention to the sensitivity against the initially available buffer size. We set interarrival time to 2.5%, magnitude to 4% and duration ratio to 2.5 (in order to avoid the problematic cases for PGNG and SPND). The other parameters had the same values as those in the previous experiments. Fig. 5a illustrates the results with respect to the buffer size (as a percentage). For low values of initial buffer size, PGNG presents the worst performance. This is because memory shortage (resulting from the combination of small buffer size and memory fluctuations) cause PGNG to frequently access many candidates stored on disk. SPND is also affected by a small buffer size, because it frequently causes suspension. However, since the I/O overhead of PGNG is excessive in this case, SPND performs better. DNMC is less affected by small buffer sizes, since it avoids the shortcomings of both the latter approaches. For medium values, PGNG and SPND perform similarly. The reason is that the I/O overhead, due to PGNG, relatively reduces. Nevertheless, both approaches are still impacted by the limited buffer size and are outperformed by DNMC. For large initial buffer sizes, as expected, all schemes converge to the same execution time, since the impact of memory fluctuations in this case is not significant.

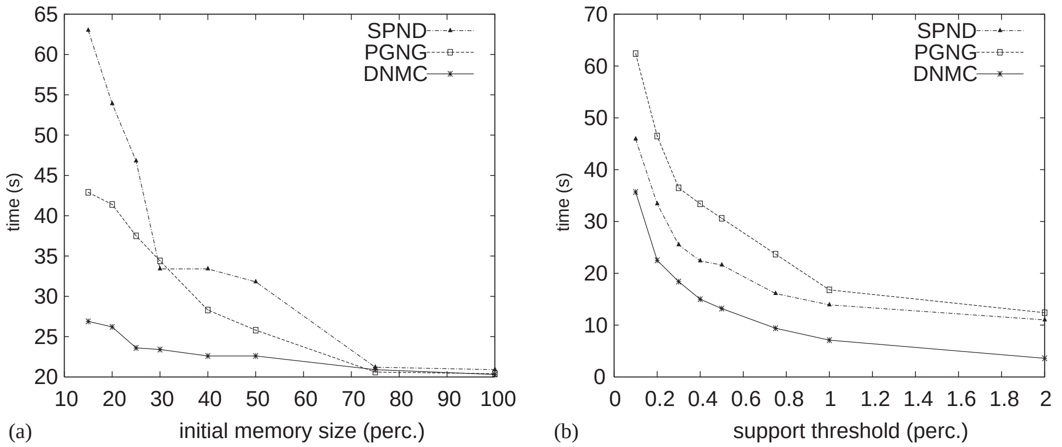

Fig. 5. Execution time vs.: (a) Initial buffer size and (b) support threshold.

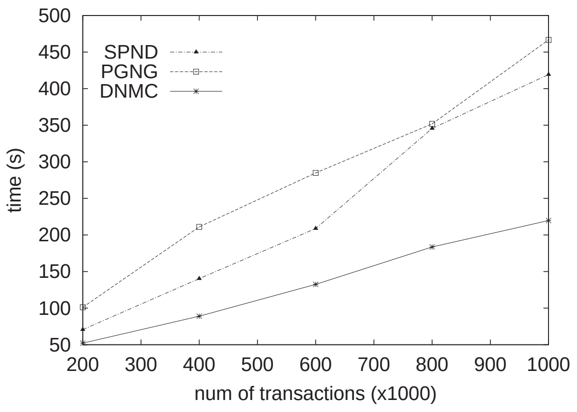

Fig. 6. Execution time vs. number of the customer transactions in the database.

Also, we measured the impact of support threshold. We set the initial buffer size to 25%, whereas all other parameters had the same values as the ones in the previous experiment. The results with respect to support threshold are given in Fig. 5b. Evidently, execution time reduces (for all schemes) with increasing support threshold. The relative performance of all methods is explained by the factors described in the previous measurements.

Finally, we examined the scalability of all schemes to the database size. We used analogous datasets and parameter values as in the ones in the previous experiments (where support threshold was set to 0.2%), and we varied the number of customer transactions in the datasets. Fig. 6 depicts the results. As shown, DNMC has a linear scale-up to the database size and clearly outperforms the other two methods, for the reasons already explained. SPND is better than PGNG for smaller databases, whereas for larger ones they perform similarly. The reason is that for large databases both the delay due to suspensions and the I/O overhead are of equal importance.

6.4. Comparison of extended GCT frameworks

As explained (Section 4.2), we have combined the memory-adaptive schemes with extended frameworks that are based on other CGT algorithms. Herein, as representative cases, we focus on the algorithms of [12,14]. When we separately applied all the schemes (i.e., SPND, PGNG, and DNMC) to each of the extended frameworks, we found results analogous to those presented earlier, in terms of relative performance (i.e., DNMC outperforms the other two schemes in all cases). For brevity we omit these results, and we move on to compare the basic with the extended frameworks, under the DNMC scheme. DNMC-B denotes the basic form of the framework, whereas DNMC-H and DNMC-S the extended frameworks that use the algorithms of [12,14], respectively.

Fig. 7a illustrates the execution time with respect to support threshold (given in percentage). The other parameters had values that were the same as those in the previous experiments. DNMC-H and DNMC-S clearly outperform DNMC-B. This is expected due to the extended frameworks they use, which achieve a reduction in the cost of generating and testing candidates ([12] reduces the number of candidates and [14] the number of passes). Comparing the DNMC-H and DNMC-S schemes, it is noticed that they perform similarly. Nevertheless, DNMC-S presents a slight improvement for medium and high support values, which is explained by the reduced number of passes relative to DNMC-H (since the number of candidates for such support values is not as large, the improvement of DNMC-H over DNMC-S in terms of this factor is moderated).

It has to be noticed that the extended frameworks may present increased memory requirements, compared to the basic one. This issue becomes important in the context of memory fluctuations. For instance, DNMC-H maintains a main-memory hash table that is used for candidate pruning. Nevertheless, the reduction in the number of candidates, which is attained by DNMC-H, clearly compensates the memory reserved by the hash table. On the other hand, in its first stage (when testing against the sample), DNMC-S uses a reduced support value, so as to reduce the number of missed candidates in the second stage. This can increase the required memory, since a larger number of candidates may be generated, compared to DNMC-B. In stand-alone execution, when sufficient memory is provided, this cost pays-off due to the reduction in the number of

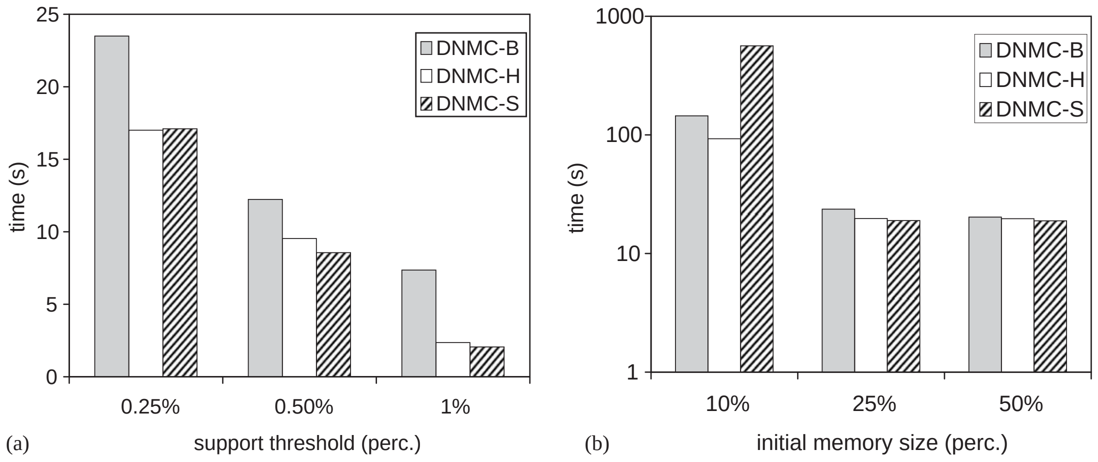

Fig. 7. Execution time for basic and extended frameworks vs.: (a) support threshold and (b) initial buffer size.

phases. However, in the examined context, buffer size is constrained and fluctuates, thus this requirement of DNMB-S may impact its performance.

For the aforementioned reasons, we measured the execution time with respect to the initial buffer size. The results are depicted in Fig. 7b (time axis is plotted in log-scale). As shown, a low value of initial buffer size impacts DNMC-S. The reason is that, as explained, the larger number of candidates, produced by DNMC-S, burdens its performance during memory fluctuations (i.e., many of them have to be stored on disk to release memory). In contrast, DNMC-H is not affected that much, since its hash table requires only a small amount of memory. As expected, when moving to larger values of initial buffer size, the execution time of DNMC-S reduces, because it is not impacted by the latter problem (i.e., the larger number of candidates in the first stage can be better handled under memory fluctuations, and they help in reducing the number of phases). Clearly, when a large (50%) initial buffer size is provided, memory fluctuations have a very small affection, and the performance of all methods is similar. It should be noticed that the aforementioned results with respect to the initial buffer size, exemplify the different requirements of the examined problem compared to the ones of existing approaches that have been designed for stand-alone execution.

6.5. *Examination of a GP algorithm*

As mentioned, for purposes of comparison, we examined the combination of memory-adaptive schemes with a non-CGT algorithm. The objective of this measurement (see also the description in the end of this section) is to demonstrate that the non-dynamic schemes (i.e., SPND and PGNG), which can be applied to non-CGT algorithms, significantly impact their efficiency. In particular, we focus on the FP-Growth algorithm, because it is efficient in terms of stand-alone execution, and a characteristic algorithm of the PG paradigm. We have used the PGNG scheme combined with FP-Growth, according to the kind of group accessing mode that is described in [9]. However, in the presence of memory fluctuations and due to the significant main memory requirements that FP-Growth presents (see Section 2), it results to large I/O overhead. For this reason, herein we focus on the use of SPND scheme combined with FP-Growth, and the resulting method is denoted as FP-S.

As described, the data structure (FP-tree) that is used by FP-Growth needs considerable memory. For fair comparison, in our experiments we provided to FP-S an initial buffer size that is larger (additional 10%) than that required for the accommodation of FP-tree entirely in main memory. We compared this method with

DNMC-H, i.e., the DNMC scheme that uses the extended framework described in the previous experiment (DNMC-H gets the same initial buffer size with FP-S).

We first measured the impact of memory-fluctuation magnitude, whereas the other parameters contained the same values as those in the previous experiments. The results are given in Fig. 8a. For very low magnitude values, the impact of fluctuations is not significant. Therefore, this case is similar to the stand-alone execution, since the algorithms are given adequate memory and they do not undergo significant memory fluctuations. Therefore, as expected, FP-S clearly outperforms DNMC-H in this case. The reason is its efficient technique of pattern fragment growth, which avoids overheads that CGT algorithms, in general, introduces. Nevertheless, as magnitude values increase, the impact of fluctuations becomes more significant. FP-S, due to its larger memory requirements, is more clearly affected. The release of significant amounts of memory to the DBMS reduces the memory that is available to FP-S, a fact that restrains its execution. In contrast, the performance of DNMC-H remains about the same, because the initially provided amount of buffer size (combined with its smaller main-memory requirements) is adequately large so that it can handle larger memory fluctuations.

We also examined the impact of fluctuation rate. As previously, we assign to FP-S an initial buffer size larger than that needed to accommodate the FP-tree. According to the previous experiment, we selected a low magnitude for memory fluctuations, i.e., 2%, so as to clearly examine the impact of fluctuation rate. The results are depicted in Fig. 8b with respect to mean interarrival time (given as percentage), where smaller values of this parameter correspond to larger rates of fluctuations (see Section 6.1). As expected, when memory fluctuations are rare (i.e., higher values of interarrival time), FP-S outperforms DNMC-H. The reason is the same as in the previous experiment, since these cases are similar to the stand-alone execution. However, as fluctuations become more frequent (i.e., lower values of interarrival time), the execution time of FP-S increases. Although the magnitude of fluctuations is not large, their increased frequency restricts the memory that is available to FP-S. Thus, its execution time is impacted. Similar to the previous experiment, DNMC-H is not impacted, because the initially provided buffer size is large enough to compensate the fluctuations.

The two previous measurements indicate the argument stated earlier, that in the context of mixed workloads with memory fluctuations, the execution time of association rule queries depends significantly on the effectiveness of the memory-adaptive scheme. Therefore, a not good memory-adaptive scheme can impact the performance of an efficient (in terms of stand-alone

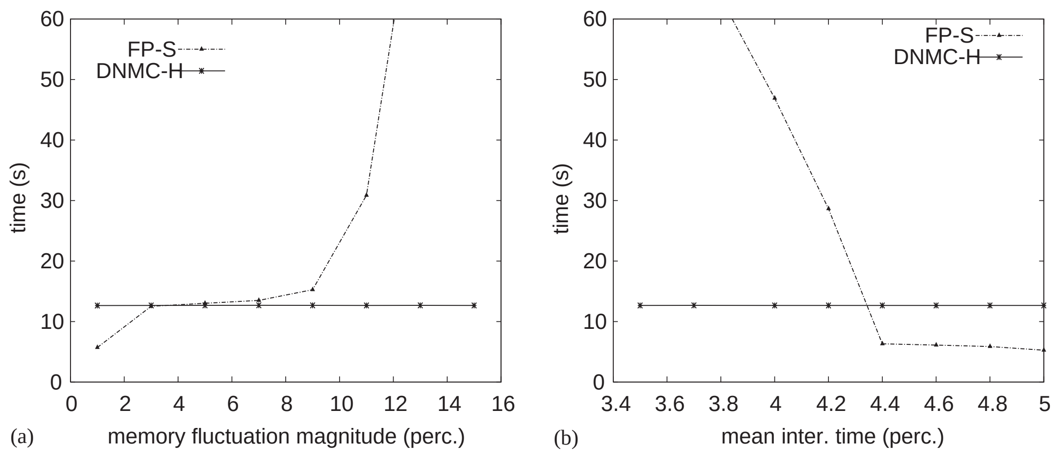

Fig. 8. Execution time of FP-S and DNMC-H vs.: (a) memory fluctuation magnitude and (b) mean interarrival time (%).

execution) association rule algorithm. As already described, we address as a topic of future work the development of dynamic memory-adaptive schemes for non-CGT algorithms, which are expected to reduce further the execution times. However, it must be noticed that the development of dynamic schemes for such algorithms has to address the increased main memory requirements that some of them present.

7. Conclusions

We have examined the main memory management for mixed workloads, consisting of OLTP transactions and association-rule mining queries. We adopt a prioritization scheme that assigns the largest priority to OLTP transactions and considers that mining queries run in the background. Therefore, the latter are presented with the problem of having to adapt to varying buffer size.

We have developed a novel memory-adaptive scheme for association rule mining queries. To our knowledge, no prior work has considered this case. The proposed scheme provides efficient and dynamic adjustment to memory fluctuations, and avoids the drawbacks of simplistic approaches (namely, the suspension and the direct resort to virtual memory). We have also proved the correctness of the proposed scheme, and we have extended it to be combined with a broad class of association rule algorithms.

The performance of the proposed scheme was examined with detailed experiments. Based on a simulation model, we have evaluated the impact of several factors, like the magnitude and the rate of memory fluctuations, the duration of period for each memory release, the initial buffer size, and finally the sensitivity against support threshold and database size. All measurements indicate the superiority of the proposed scheme.

Future work will include the examination of other prioritization schemes. For instance, constrained association rule queries [26] may present reduced memory (and CPU) requirements. Thus, they may obtain priority over some OLTP transactions. Also, we will focus on an examination of other performance factors, like delays introduced by disk queues. Finally, we will work on the development of dynamic memory-adaptive schemes that can be combined with non-CGT association rule algorithms.

References

[1] K. Brown, M. Carey, M. Livny, Managing memory to meet multiclass workload response time goals, Proceedings of the International Conference on Very Large Databases (VLDB'93), 1993, pp. 328–341.

[2] H. Pang, M. Carey, M. Livny, Memory-adaptive external sorting, Proceedings of the International Conference on Very Large Databases (VLDB'93), 1993, pp. 618–629.

[3] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, Proceedings of the ACM International Conference on Management of Data (SIGMOD'93), 1993, pp. 207–216.

[4] S. Sarawagi, S. Thomas, R. Agrawal, Integrating association rule mining with relational database systems: Alternatives and Implications, Proceedings of the ACM International Conference on Management of Data (SIGMOD'98), 1998, pp. 343–354.

[5] S. Thomas, S. Chakravarthy, Performance evaluation and optimization of join queries for association rule mining, Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99), 1999, pp. 241–250.

[6] K. Rajamani, A. Cox, B. Iyer, A. Chadha, Efficient mining for association rules with relational database systems, Proceedings of the International Database Engineering and Applications Symposium (IDEAS'99), 1999, pp. 148–155.

[7] E. Riedel, C. Faloutsos, G. Ganger, D. Nagle, Data mining on an OLTP system (Nearly) for free, Proceedings of the ACM International Conference on Management of Data (SIGMOD'00), 2000, pp. 13–21.

[8] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, Proceedings of the International Conference on Very Large Databases (VLDB'94), 1994, pp. 487–499.

[9] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, Proceedings of the International ACM Conference on Management of Data (SIGMOD'00), 2000, pp. 1–12.

[10] Y. Xiao, M. Dunham, Considering main memory in mining association rules, Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99), 1999, pp. 209–218.

[11] S. Brin, R. Motwani, J. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, Proceedings of the ACM International Conference on Management of Data (SIGMOD'97), 1997, pp. 255–264.

[12] J. Park, M.-S. Chen, P. Yu, Using a hash-based method with transaction trimming for mining association rules,

IEEE Trans. Knowledge Data Eng. (TKDE) 9 (5) (1997) 813–825.

[13] A. Savasere, E. Omiecinski, S. Navathe, An efficient algorithm for mining association rules in large databases, Proceedings of the International Conference on Very Large Databases (VLDB'95), 1995, pp. 432–444.

[14] H. Toivonen, Sampling large databases for association rules, Proceedings of the International Conference on Very Large Databases (VLDB'96), 1996, pp. 134–145.

[15] R. Agarwal, C. Aggarwal, V. Prasad, A tree projection algorithm for generation of frequent item sets, J. Parallel Distrib. Comput. 61 (3) (2001) 350–371.

[16] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, D. Yang, H-mine: hyper-structure mining of frequent patterns in large databases, Proceedings of the IEEE International Conference on Data Mining (ICDM'01), 2001, pp. 441–448.

[17] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Efficient mining of association rules using closed itemset lattices, Inform. Systems 24 (1) (1999) 25–46.

[18] M. Zaki, S. Parthasarathy, M. Ogihara, W. Li, New algorithms for fast discovery of association rules, Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'97), 1997, pp. 283–286.

[19] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, D. Shah, Turbo-charging vertical mining of large databases, Proceedings of the ACM International Conference on Management of Data (SIGMOD'00) 2000, pp. 22–33.

[20] F. Geerts, B. Goethals, J. Van den Bussche, A tight upper bound on the number of candidate patterns, Proceedings of the IEEE International Conference on Data Mining (ICDM'01), 2001, pp. 155–162.

[21] K. Brown, M. Carey, D. DeWitt, M. Mehta, Resource allocation and scheduling for mixed database workloads, Technical Report, University of Wisconsin, 1992.

[22] R. Bayardo, Efficiently mining long patterns from databases, Proceedings of the ACM Internationl Conference on Management of Data (SIGMOD'98), 1998, pp. 85–93.

[23] J. Paulin, Performance evaluation of concurrent OLTP and DSS workloads in a single database system, Master's Thesis, Carleton University, 1997.

[24] C. Faloutsos, R. Ng, T. Sellis, Predictive load control for flexible buffer allocation, Proceedings of the International Conference on Very Large Databases (VLDB'91), 1991, pp. 265–274.

[25] S. Chaudhuri, Data mining and database systems: where is the intersection?, IEEE Data Eng. Bull. 21 (1) (1998) 4–8.

[26] R. Ng, L. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained association rules, Proceedings of the International ACM Conference on Management of Data (SIGMOD'98), 1998, pp. 13–24.