Identifying Hot/Cold Data in Main-Memory Database using Frequent Item set Mining

Ghada M.Afify[1], Ali El Bastawissy[2], Osman M.Hegazy[3]

[1,3]Department of Information System, Faculty of Computers and Information, Cairo University, Egypt
[2] Faculty of Computer Science, MSA University, Cairo, Egypt

ABSTRACT

The growth in main-memory capacity has fueled the development of main-memory database systems. Due to the explosive growth of data, it is crucial to deal with data overflow. Recent research works tackle this problem by developing approaches that track attribute accesses to identify hot/cold attributes. Distinctly, we introduce a novel algorithm called HC_Apriori, which adapts the classic Apriori algorithm, employing a new optimization measure. To the best of the authors' knowledge, this is the first initiative to employ Frequent Item set Mining in order to classify hot/cold attributes. Our objective is to enhance the performance in terms of two dimensions: storage space and execution time. We implemented our algorithm using trie data structure and compared it with classic Apriori algorithm. Two popular real-world datasets were used. Experimental results show that, our proposed HC_Apriori reduces the storage space by average of 59–97% and reduces the execution time by average of 60-100%.

Keywords: Main-Memory Database, Hot/Cold Data Management, Frequent Item set Mining, Apriori Algorithm, Trie Data Structure.

1. INTRODUCTION

Existing disk-based systems can no longer offer timely response due to the high access latency to hard disks. To meet the strict real-time requirements for analyzing huge amounts of data, a main-memory database that keeps the data entirely in the random access memory (RAM) is necessary. Although there is substantial increase in memory size and sharp drop in its price, main-memory databases cannot match the rapid growth of big data, which makes it essential to deal with data overflow (where the size of the data exceeds the size of main-memory) [1]. Recent research works about optimization aspects on main-memory data management and processing focus on data overflow. Effective eviction techniques are adopted to replace the in-memory data when the main-memory is not sufficient. The more free memory, the larger systems to be stored in the database, which improves the performance and the cost efficiency. The main objective is to separate the data into frequent (hot) and infrequent (cold) data. The hot data will remain in main-memory and the cold ones will be evicted to a cheaper cold store (disk) [2].

The main difference among the existing approaches is the level of granularity in which the data is accessed and classified as hot or cold, namely: at the tuple-level, attribute-level or page-level. Most of these approaches use the Least Recently Used (LRU) and Least Frequently Used (LFU) techniques to distinguish between hot/cold data. Uniquely, we adopt a different data mining approach, specifically Frequent Item set Mining (FIM) in order to identify hot/cold attributes. FIM plays an essential role in many data mining tasks in order to find interesting patterns from databases. To the best of the authors' knowledge, this is the first initiative to use FIM to identify hot/cold attributes according to attribute accesses in main-memory database. To achieve this objective, we adapted the classic Apriori algorithm and introduced the new HC_Apriori algorithm which distinguishes between frequently accessed data (hot) and infrequently accessed data (cold). Moreover, we introduced a new measure, which we called Item set-Gain, to confine the hot list output from the algorithm. The Item set-Gain usage reduced the output space and defined the best candidate item sets to be stored in main-memory database, which in turn, reduced the storage space.

The major contributions of this work are:

1. Introduce novel HC_Apriori algorithm to identify hot/cold attributes and demonstrate its workflow via detailed case study.
2. Introduce new optimization measure, Item set-Gain, to reduce the output space storage.
3. Evaluate the effectiveness of the proposed algorithm using two real-world datasets.

The remainder of this paper is structured as follows. Section 2 surveys the recent related work and the background knowledge. Section 3 introduces the proposed algorithm. Section 4 describes a detailed case study to illustrate the proposed algorithm workflow. Section 5 reports the experimental evaluation results. Finally, Section 6 concludes the paper.

2. RELATED WORKAND BACKGROUND

2.1 Related Work

Recent development in hardware has led to rapidly dropping market prices of main-memory in the past years. This development made it economically feasible to use the main-memory as the primary data store of Database Management Systems (DBMS), which is the main characteristic of a main-memory DBMS. Recent research works focus on main-memory DBMS storage. Commercial systems include Oracle's Times Ten [3], IBM's solid DB [4] and Volt DB [5]. On the other hand, research systems include HYRISE [6], H-Store [7], HyPer [8] and Monet DB [9]. These systems are suitable for the databases that are smaller than the amount of the physical available memory. If memory is exceeded, then it will lead to performance problems. Due to the explosive growth of data, it is impossible to maintain all data in main-memory. This problem of capacity limitation of main-memory has been addressed by a number of recent works.

We conducted a comprehensive analysis of existing main-memory databases and their approaches that focus on hot/cold data management and solve the data overflow problem. To summarize its findings, HyPer is a main-memory hybrid OLTP and OLAP system [8]. It has a compacting-based approach used to handle hot/cold data [10]. In this approach, the authors use the capabilities of modern server systems to track data accesses. The data stored in a columnar layout is partitioned horizontally and each partition is categorized by its access frequency. Data in the (rarely accessed) frozen category is still kept in memory but compressed and stored in huge pages to better utilize main memory. HyPer performs hot/cold data classification at the Virtual Machine (VM) page level.

In [11], authors proposed a simple and low-overhead technique that enables main-memory database to efficiently migrate cold data to secondary storage by relying on the Operating System (OS)'s virtual memory paging mechanism. Hot pages are pinned in memory while, cold pages are moved out by the OS to cold storage. A comparable approach is presented in Hekaton [12], a SQL server's memory-optimized OTLP engine that manages hot/cold tuples. In Hekaton, the primary copy of the database is entirely stored in main-memory.

Hot tuples remain in main-memory while cold ones are moved to cold secondary storage. In [13], authors implemented hot/cold separation in main-memory database H-Store. The authors call this approach "Anti-Caching" to underline that hot data is no longer cached in main-memory but cold data is evicted to secondary storage. To trace accesses to tuples, tuples are stored in a LRU chain per table. We observe that the above approaches are different from our paper scope, [10, 11] perform hot/cold data classification at the VM page-level, while [12, 13] at tuple-level. Otherwise, our scope is to identify hot/cold data at attribute-level of granularity. Table 1 summarizes the comparison between approaches that classify hot/cold attributes in main-memory databases.

Table 1: Main-memory database approaches to classify hot/cold attributes.

Main-memory database approach	Main-memory physical layout	Technique for classification
Oracle 12c dual-format[14]	Columnar and Row	LRU
SAP HANA [15]	Columnar	LFU
Proposed HC_Apriori	Columnar	FIM

Oracle Database 12c In-Memory Option [14] is based on dual-format data store, suitable for use by response-time critical Online Transaction Processing (OLTP) applications as well as Online Analytical Processing (OLAP) applications for real time decision-making. Oracle in-memory column store uses LRU technique to identify hot/cold attributes. SAPHANA [15], is a columnar in-memory DBMS suitable for both OLTP and OLAP workloads. It offers an approach to handle data aging.

They created LFU ordering of all attributes. Attributes scanned sequentially are ranked highest priority to stay in main-memory, but attributes accessed randomly are prioritized according to their usage frequency. Contrarily, we proposed a novel approach named, HC_Apriori that uses data mining, in particular FIM to classify hot/cold attributes in main-memory database.

2.2 Background

Frequent Item set Mining: Frequent Item set Mining has been an essential part of data analysis and data mining. FIM tries to extract information from transactional databases based on frequently occurring events, which are considered interesting according to a user given minimum frequency threshold. Finding frequent item sets in a set of transactions is a popular method for market basket analysis, which aims at finding regularities in the shopping behavior of customers of supermarkets, mail-order companies, on-line shops etc. In particular, it is famous for identifying sets of products that are frequently bought together [16].

The concept of FIM was first introduced for mining transaction databases. Let I = {i_1, . . .,i_m} be set of all items and a transaction database D = {t_1, …….t_n} is set of transactions. A transaction database can list, for example, the sets of products bought by the customers of a supermarket in a given period of time. A k-item set α, which consists of k items from I, is frequent if α occurs in a transaction database D no lower than θ |D| times, where θ is a user-specified minimum support threshold (called min_supp), and |D| is the total number of transactions in D [17]. The candidate generation and the support counting processes require an efficient data structure in which all candidate item sets are stored, since it is important to efficiently find the item sets that are contained in a transaction or in another item set. Here, we compare between two different data structures: Classic Apriori and Trie structure for Apriori.

Classic Apriori: The most known implementation of FIM is Apriori algorithm [18]. It is firstly proposed by R. Agrawal and R Srikant in 1994. It is a data mining classic algorithm used to find frequent item sets in transactions database using an iterative level-wise approach based on candidate generation, where k-item sets are used to explore (k+1)-item sets. It uses the Apriori property to reduce the search space: All non-empty subsets of a frequent item set must also be frequent. The Apriori algorithm is based on two main steps [19]:

1. Generate Phase: In this phase, candidate (k+1)-item set is generated using k-item set;
2. Prune Phase: In this phase, candidate set is pruned to generate large frequent item set using "min_supp" as the pruning parameter.

Trie structure for Apriori

A trie is a rooted, labeled prefix-tree. Each label is an item. The root is defined to be at depth 0 and a node at depth d can point to nodes at depth d+1. A pointer is also referred to as edge or link. If node u points to node v, then we call u the parent of v, and v the child node of u. Nodes with the same parent are siblings and nodes that have no children are called leaves. Each node represents an item sequence that is the concatenation of labels of the edges that are on the path from the root to the node. So a path from root to each node represents an item set. The value of each node is the support count for the item set it represents. For each transaction record T in the database the trie (containing the candidate item sets) is recursively traversed and the value of each leaf node will be incremented if T contains the item set represented by that node. At the end, nodes with a support count less than the required minimum will be pruned. In the candidate generation phase, we add a leaf node to its left siblings to create new valid candidates, eliminating the need for further processing [20].

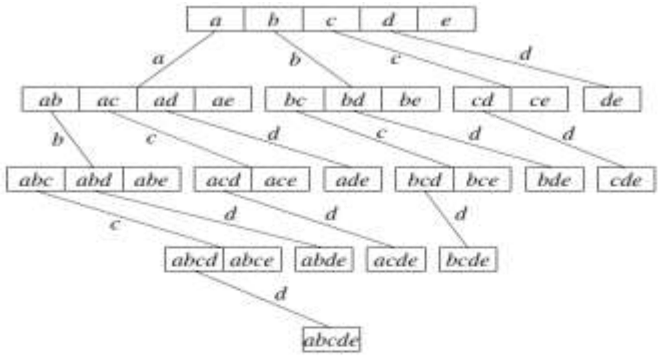


Figure1. Prefix tree for five items

The prefix tree in Fig. 1 is an example of five items. It is grown top-down level by level, pruning those branches that cannot contain a frequent item set. In this tree, those item sets are combined in a node which has the same prefix w.r.t. to some arbitrary, but fixed order of the items (e.g.: this order is simply a, b, c, d, e). With this structure, the item sets contained in a node of the tree can be constructed easily in the following way: Take all the items with which the edges leading to the node are labeled (this is the common prefix) and add an item that succeeds, in the fixed order of the items, the last edge label on the path. An in-depth study on the implementation details of a trie for Apriori can be found in [21].

3. PROPOSED HC_APRIORI ALGORITHM

HC_Apriori is a novel algorithm that distinguishes between frequent accessed attributes (hot) and infrequent accessed attributes (cold) in a transaction database. The outputs of the HC_Apriori algorithm are hot/cold lists of attributes from the database, and the in-memory list that contains the best candidate item sets to be stored in main-memory. We developed HC_Apriori algorithm using the trie data structure. It uses a horizontal transaction database representation, that each transaction is an array of the contained items. The algorithm is presented using pseudo code in Fig. 2. The algorithm traverses the prefix tree in breadth-first order (level by level). That is, it first checks item sets of size 1 (line 2) and that's how the root node is created. Then item sets of size 2 and so on (line 5). For a transaction, the support of item sets is determined by traversing all subsets of the current size and incrementing the corresponding item set counters (lines 6-12). Item sets with a support less than the minimum support (min_supp) will be pruned and stored in the cold list. Otherwise, item sets with a support more than the min_supp will be stored in the hot list (lines 13-19). We define a new measure called Item set-Gain, which reduces the output space and defines the best candidate item sets from the output hot list to be stored in main-memory.

$$\text{Item set-Gain } (i) = X / Y \qquad (1)$$

Where X represents the sum of supports for attributes in an item set (i) that are accessed in all the queries in the database, and Y represents the number of attributes in the item set (i). In (lines 22-33), for each item set (one-element item set) in the root node and, for each branch for this item set: First, we calculate the Item set-Gain measure for each item set (i)in that branch using (1). Second, we calculate the maximum Item set-Gain value (max-gain) for each branch. Finally, the item sets with the highest values (most interesting) are stored in main-memory hot list (IM-HL).

```
Algorithm: HC_Apriori, Apriori with Hot/Cold Lists.
 Input: D, transaction database;
min_supp, the minimum support count threshold.
Output: HL, Hot list contains frequent item sets in D;
          CL, Cold list contains infrequent item sets in D;
        IM-HL, In-memory hot list.
Method:
1.   Begin
2.   $L_1$ = find_frequent_1-item sets(D);
3.   k=2; max-gain=0; j=0;
4.   do
5.   Identify k-item candidate set $C_k$ from frequent item set $L_{k-1}$
6.   for each transaction $t \in D$
7.   for each item set $i \in C_k$
8.   if $i \in t$ then
9.   increment support
10.   end if
11.   end for
12.   end for
13.   for each item set i in $C_k$
14.  if $support_i \geq$ min_supp then
15.   $HL_k \cup i$
16.  else
17.  $CL_k \cup i$
18.  end if
19.  end for
20.  increment k
21.  while ($L_{k-1} \neq \emptyset$)
22.  for each (1-item set) r in root node in the tree
23.  for each branch b of item set
24.  for each item set i in branch
25.  calculate item set-gain g for item set i
26.  if $g_i \geq$max-gain then
27.  max-gain= $g_i$
28.  $IM\text{-}HL_j \cup i$
29.  increment j
30.  end if
31.   end for
32.   end for
33.  end for
34.  return HL, CL, IM-HL;
35.  End
```

Figure 2. Proposed HC_Apriori algorithm

4. CASE STUDY

In this section, a detailed case study is presented in order to demonstrate the proposed HC_Apriori workflow. Assuming there is a transaction database with 5 items and 10 transactions. The minimum support min_supp is set to 3. The transaction database is read from an input file.

4.1 Classic Apriori Algorithm: Frequent item set mining using classic Apriori algorithm is shown in the Table 2. We have subsets of frequent item sets of size 1, 2 and 3. The output file contains 15 frequent item sets with their support values (in brackets). The output file shows the group of 1-item sets first then, group of 2-item sets and finally group of 3-item sets.

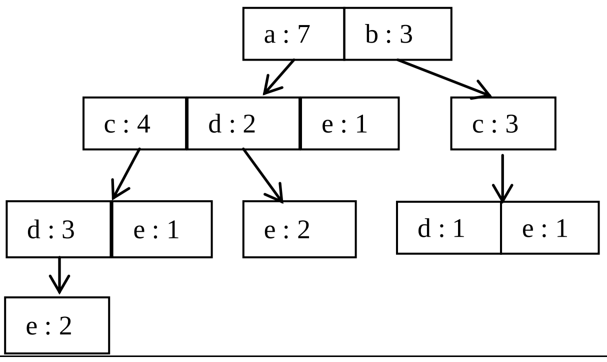
Table 2: FIM using classic Apriori algorithm

Input	Processing			Output
Transaction Database	1-item sets	2-item sets	3-item sets	Classic Apriori Frequent Item sets
a d e	a: 7	a c: 4	a c d: 3	b (3)
b c d	b: 3	a d: 5	a c e: 3	d (6)
a c e	c: 7	a e: 6	a d e: 4	e (7)
a c d e	d: 6	b c: 3		a (7)
a e	e: 7	c d: 4		c (7)
a c d		c e: 4		b c (3)
b c		d e: 4		d e (4)
a c d e				d c (4)
b c e				e c (4)
a d e				a c (4)
				d a (5)
				e a (6)
				d a c (3)
				e a c (3)
				d e a (4)

4.2 HC_Apriori Algorithm: Workflow of HC_Apriori algorithm

Input: As shown in the following Table 3, transaction items are sorted in order (a, b, c, d, e), the transactions are sorted lexicographically and a prefix-tree is constructed.

Table 3: FIM using proposed HC_Apriori algorithm

Input		
Transaction Database	Lexicographically Sorted	Prefix-tree Representation
a d e	a c d	 a : 7 b : 3 c : 4 d : 2 e : 1 c : 3 d : 3 e : 1 e : 2 d : 1 e : 1 e : 2
b c d	a c d e	
a c e	a c d e	
a c d e	a c e	
a e	a d e	
a c d	a d e	
b c	a e	
a c d e	b c	
b c e	b c d	
a d e	b c e	

Processing:

Step 1: we apply the HC_Apriori algorithm on the input prefix-tree. Then, the frequent item sets are written to an output file. In the output file, a symbol ($) is added to split it as branches for each (1-item set) in the root node. For example, for item set b, there is only 1 branch with 2 frequent item sets. For item set d, there are 3 different branches; branch 1: has 3 frequent item sets, branch 2: has 2 frequent item sets and branch 3: has 1 frequent item set.

Step 2: We calculate the Item set-Gain for each frequent item set in the output file. It’s written in the file between "[ ]" brackets. For example, Item set-Gain (b c) = (3 + 7) / 2 = 5, Item set-Gain (d e a) = (6 + 7 + 7) / 3 = 6.67.

Step 3: In order to reduce the number of frequent item sets generated in step 2. We calculate the max-gain value for each branch, which represents the highest item set gain value in that branch. For example, for branch {d [6.00], d e a [6.67], d e [6.50]}, the max-gain value is [6.67]. Then, the item set {d e a} is written in the IM-HL as the best candidate from this branch to be stored in main-memory. As a result, the number of frequent item sets becomes 7 item sets. Thus, the storage space is reduced by percentage of 53%.

Output: The output for each step is shown in Table 4.

Table 4: Output of the HC_Apriori algorithm

Output		
Step 1	Step 2	Step 3
b c (3) b (3) $ d (6) d e a (4) d e (4) $ d a (5) d a c (3) $ d c (4) $ e (7) e a (6) e a c (3) $ e c (4) $ a (7) a c (4) $ c (7)	b c [5.00] b [3.00] $ d [6.00] d e a [6.67] d e [6.50] $ d a [6.50] d a c [6.67] $ d c [6.50] $ e [7.00] e a [7.00] e a c [7.00] $ e c [7.00] $ a [7.00] a c [7.00] $ c [7.00]	b c [5.00] d e a [6.67] d a c [6.67] d c [6.50] e a c [7.00] e c [7.00] a c [7.00]

5. EXPERIMENTAL EVALUATION RESULTS

In order to validate the effectiveness of proposed HC_Apriori algorithm, we implemented it using trie data structure [22] and we compared it with classic Apriori algorithm. In the following sub-sections, we present the experiment setup, the workload, and finally the performance study is reported.

5.1 Experiment Setup: Experiments were run using the following resources:

a) Hardware: Intel Core TM i7 CPU (@ 2.60 GHZ) with 12 GB of RAM running on 64-bit Windows 8.1.

b) Software tools: Microsoft visual C++.

5.2 Workload: In order to accurately assess the proposed algorithm effectiveness, it was important to validate it using real-world datasets. We have used two popular publicly available datasets from the Frequent Item set Mining Implementations (FIMI) Repository [23]. The datasets are Mushroom and Chess. The input datasets are not compressed. The characteristics of these datasets are listed in Table 5.

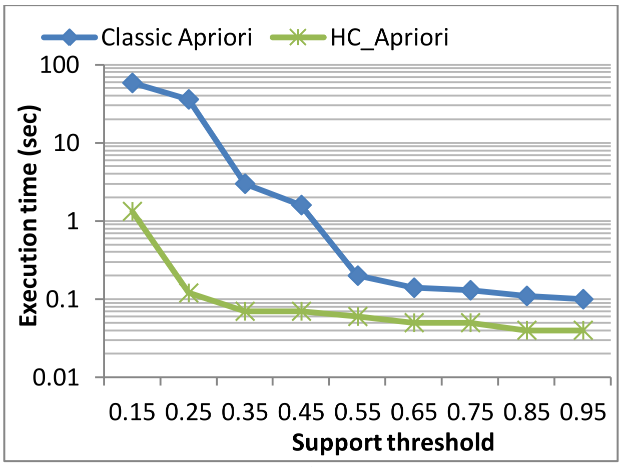
Table 5: Dataset characteristics

Dataset	#Transactions	#Items
Mushroom.dat	8124	119
Chess.dat	3196	75

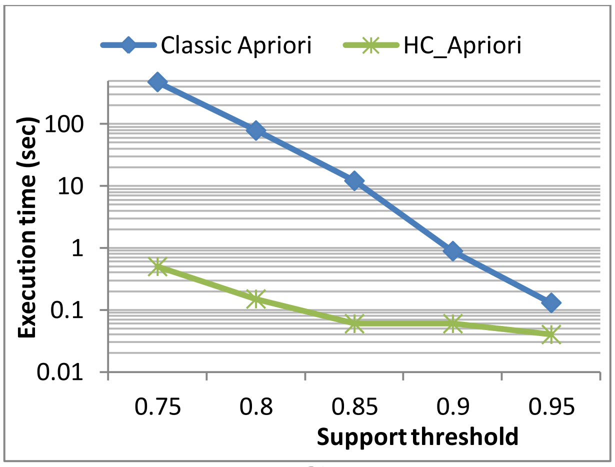
There are three main parameters for an application: First, the file that contain the transactions in market basket format. Second, an absolute minimum support threshold. Third, the output file the frequent item sets are sent to. The data items are space separated in the input file. Each line in the input file gives a new tuple of the transaction. The time function used for assessment was the system clock, which returned time in second. In this work, the support threshold is the minimum support (min_supp) divided by the total number of the transactions in the dataset. The value of the threshold lies between 0 and 1.

5.3 Performance Study: The effectiveness of proposed HC_Apriori algorithm is experimentally evaluated in terms of two performance dimensions: execution time and storage space. We have used different values of minimum support threshold, following recent research works [24, 25]. For Mushroom dataset, we used support threshold values from 0.15 to 0.95 using step of 0.1. For Chess dataset, we used support threshold values from 0.75 to 0.95 using step of 0.05.

5.3.1 Performance Dimension: Execution Time: In this experiment, we investigate the execution time of the proposed HC_Apriori algorithm compared to classic Apriori algorithm. As shown in Fig. 3, results show that the execution time behavior is similar for both datasets as it decreases with increasing values of support threshold. It is obvious that the proposed HC_Apriori outperforms the classic Apriori in both datasets. It can be noted that in Fig. 3 (a), HC_Apriori algorithm achieves execution time reduction on average of 60-70% for high values of support threshold and on average of 96-100% for low values of support threshold. In Fig. 3 (b), the HC_Apriori algorithm has execution time reduction on average of 69-100% compared to the classic Apriori algorithm.



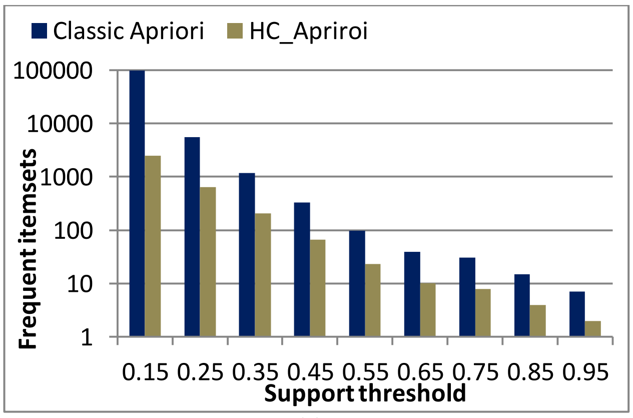
(a)



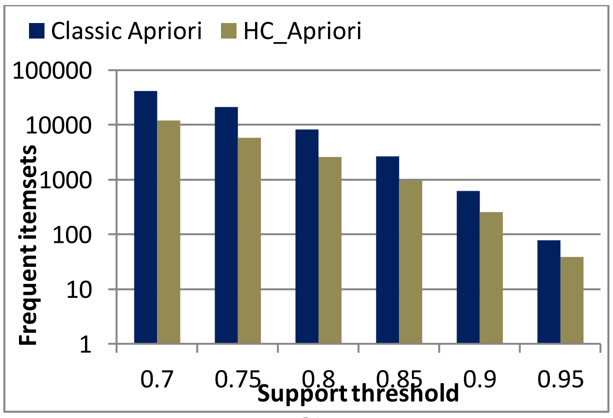
(b)

Figure 3.Execution time for (a) Mushroom dataset and (b) Chess dataset

5.3.2 Performance Dimension: Storage Space In this experiment, we investigate the storage space requirement of the proposed HC_Apriori algorithm compared to classic Apriori algorithm. Storage space here is defined as the number of frequent item sets resulted from the application. As shown in Fig. 4, results show that the storage space for both datasets decreases with increasing values of support threshold. It is obvious that the proposed HC_Apriori performance is superior to the classic Apriori in both datasets. It can be noted that in Fig. 4 (a), the HC_Apriori algorithm has storage reduction on average of 71–97% compared to the classic Apriori algorithm. In Fig. 4 (b), the HC_Apriori algorithm has storage reduction on average of 59–72% compared to the classic Apriori algorithm.



(a)



(b)

Figure 4. Storage space for (a) Mushroom dataset and (b) Chess dataset

CONCLUSION

Recent evolution in main-memory sizes has driven the development of main-memory big data. Nonetheless, main-memory is still a scarce resource and expensive compared to disk. Recent research works adopted effective eviction approaches to replace in-memory data when the main-memory is not sufficient. The main objective is to identify frequent accessed (hot) and infrequent accessed (cold) data. The hot data will remain in main-memory while the cold ones will be evicted to a cheaper cold store (disk). Most of these approaches track attribute accesses to identify hot/cold attributes using LRU and LFU techniques. In contrast, we use data mining specifically, FIM. We proposed a novel algorithm called HC_Apriori, which adapts the classic Apriori algorithm in order to classify hot/cold attributes, using a new optimization measure. We demonstrated its workflow through a detailed case study. We evaluated its effectiveness using two real-world datasets. Experimental evaluation proved that the proposed HC_Apriori algorithm is superior to classic Apriori algorithm in terms of performance metrics: execution time and storage space. The proposed HC_Apriori reduces the storage space by average of 59–97% in both datasets and reduces the execution time by average of 60-100% compared to classic Apriori algorithm.

REFERENCES

[1]. H. Zhang, G. Chen, B. C. Ooi, W. F. Wong, S. Wu and Y. Xia,"Anti-Caching -based elastic memory management for Big Data,"Data Engineering (ICDE), IEEE 31st International Conference on, Seoul, pp. 1268-1279, 2015.

[2]. Afify, G. M., El Bastawissy, A., & Hegazy, O. M.,"A hybrid filtering approach for storage optimization in main-memory cloud database," Egyptian Informatics Journal, vol. 16(3), pp. 329-337, 2015.

[3]. Lahiri T, Neimat M, Folkman S, "Oracle times ten: an in-memory database for enterprise applications," IEEE Data Eng Bull, vol. 36 (2), pp. 6–13, 2013.

[4]. Lindstroem J, Raatikka V, Ruuth J, Soini P, Vakkila K, "IBM solidDB: in-memory database optimized for extreme speed and availability," IEEE Data Eng Bull, vol. 36 (2), pp. 14–20, 2013.

[5]. Stonebraker M, Weisberg A, "The VoltDB main memory DBMS," IEEE Data Eng Bull, vol. 36 (2), pp. 21–7, 2013.

[6]. Grund M, Kruger J, Plattner H, Zeier A, Cudre-Mauroux P, Madden S., "HYRISE – a main memory hybrid storage engine," Proc VLDB Endow, vol. 4 (2), pp. 105–16, 2012.

[7]. Kallman R, Kimura H, Natkins J, Pavlo A, Rasin A, Zdonik S, et al, "H-Store: a high-performance, distributed main memory transaction processing system," Proc VLDB Endow, vol. 1 (2), pp. 1496–9, 2008.

[8]. Kemper A, Neumann T, "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," 27th International Conference on Data Engineering (ICDE), IEEE, pp. 195–206, 2011.

[9]. Boncz P, Zukowski M, Nes N, "MonetDB/X100: hyper-pipelining query execution," CIDR, vol. 5, pp. 225–37, 2005.

[10]. Funke F, Kemper A, Neumann T, "Compacting transactional data in Hybrid OLTP & OLAP databases," Proc VLDB Endow, vol. 5 (11), pp. 1424–35, 2012.

[11]. Stoica R, Ailamaki A, "Enabling efficient OS paging for main memory OLTP databases," Proceedings of the ninth international workshop on data management on new hardware, ACM, 2013.

[12]. Diaconu C, Freedman C, Ismert E, Larson P-A, Mittal P, Stonecipher R, et al, "Hekaton: SQL server's memory-optimized OLTP engine," Proceedings of the international conference on management of data, SIGMOD, ACM, pp. 1243–54, 2013.

[13]. DeBrabant J, Pavlo A, Tu S, Stonebraker M, Zdonik S, "Anticaching: a new approach to database management system architecture," Proc VLDB Endow, vol. 6 (14), pp.1942–53, 2013.

[14]. Colgan M, Kamp J, Lee S, "Oracle database in-memory," Oracle White Paper, 2014.

[15]. Archer S., "Data-aging strategies for SAP NetWeaver BW focusing on BW's new NLS offering for Sybase IQ," SAP BW Product Management, 2013.

[16]. Moens, S., Aksehirli, E., & Goethals, B., "Frequent item set mining for big data," Big Data, IEEE International Conference, pp. 111-118, 2013.

[17]. Rupayla, P., & Patidar, K., "A Comprehensive Survey of Frequent Item Set mining Methods," IJETAE, ISSN, pp. 2250-2459, 2014.

[18]. Agrawal, Rakesh, and Ramakrishnan Srikant, "Fast algorithms for mining association rules," Proc. 20th int. conf, very large databases, VLDB, 1994.

[19]. Ritu Garg and Preeti Gulia, "Article: Comparative Study of Frequent Item set Mining Algorithms Apriori and FP Growth," International Journal of Computer Applications, NY, USA, vol. 126 (4), pp. 8-12, 2015.

[20]. Bodon, F., "A fast apriori implementation," Proceedings of the IEEEICDM workshop on frequent item set mining implementations (FIMI'03), 2010.

[21]. Borgelt, Christian., "Efficient implementations of apriori and eclat," Proceedings of the IEEE ICDM workshop on frequent item set mining implementations, FIMI'03, 2003.

[22]. Christian Borgelt, Frequent Pattern Mining, software available athttp://www.borgelt.net/software.html

[23]. Frequent item set mining dataset repository, http://fimi.cs.helsinki.fi/data.

[24]. Li, Juan, Pallavi Roy, Samee U. Khan, Lizhe Wang, and Yan Bai, "Data mining using clouds: An experimental implementation of apriori over mapreduce," Proc. 12th International Conference on Scalable Computing and Communications (ScalCom'13), pp. 1-8, 2012.

[25]. Jin, Ruoming, Scott McCallen, Yuri Breitbart, Dave Fuhry, and Dong Wang, "Estimating the number of frequent item sets in a large database," Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, ACM, pp. 505-516, 2009.