Egyptian Informatics Journal (2015) **16**, 329–337

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

FULL-LENGTH ARTICLE

A hybrid filtering approach for storage optimization in main-memory cloud database

Ghada M. Afify [a,*], **Ali El Bastawissy** [b], **Osman M. Hegazy** [a]

[a] *Department of Information System, Faculty of Computers and Information, Cairo University, Egypt*
[b] *Faculty of Computer Science, MSA University, Cairo, Egypt*

Received 24 December 2014; revised 16 June 2015; accepted 30 June 2015
Available online 21 August 2015

KEYWORDS

Cloud computing;
Cloud storage;
Main-memory database;
Hot/cold data;
Cold data management

Abstract Enterprises and cloud service providers face dramatic increase in the amount of data stored in private and public clouds. Thus, data storage costs are growing hastily because they use only one single high-performance storage tier for storing all cloud data. There's considerable potential to reduce cloud costs by classifying data into active (hot) and inactive (cold). In the main-memory databases research, recent works focus on approaches to identify hot/cold data. Most of these approaches track tuple accesses to identify hot/cold tuples. In contrast, we introduce a novel Hybrid Filtering Approach (HFA) that tracks both tuples and columns accesses in main-memory databases. Our objective is to enhance the performance in terms of three dimensions: storage space, query elapsed time and CPU time. In order to validate the effectiveness of our approach, we realized its concrete implementation on Hekaton, a SQL's server memory-optimized engine using the well-known TPC-H benchmark. Experimental results show that the proposed HFA outperforms Hekaton approach in respect of all performance dimensions. In specific, HFA reduces the storage space by average of 44–96%, reduces the query elapsed time by average of 25–93% and reduces the CPU time by average of 31–97% compared to the traditional database approach.

1. Introduction

Main-memory database (MMDB) is a database management system that primarily relies on main-memory for computer data storage. It is contrasted with database management systems which employ a disk storage mechanism. Main-memory databases are faster than disk-based databases since the internal optimization algorithms are simpler and execute fewer CPU instructions. Accessing data in memory eliminates seek

* Corresponding author.
Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.eij.2015.06.007

time when querying the data, which provides faster and more predictable performance than disk [1].

Recent evolution in main-memory sizes has prompted huge increases in the prevalence of database systems that keep the entire database in memory. Nonetheless, main-memory is still a scarce resource and expensive compared to disk [2]. A major goal of recent research works is to improve main-memory storage optimization. The more free memory the larger systems to be stored in the database, which improves the performance and the cost efficiency. The objective is to separate the data into active (hot) and inactive (cold) data. The hot data will remain in main-memory and the cold ones will be moved to a cheaper cold store [3]. The main difference in the existing techniques is the level of granularity in which the data is accessed and classified as hot or cold; which in some databases is at the tuple-level and in others at page-level.

In the same context, cloud storage becomes more expensive because charges of "GB transferred" over the network vary with the amount of data transferred each month, conceivably with amazing and capricious variations. Moreover, extra hidden fees, such as connecting fees, maintenance charges, and data access charges can add up quickly [4]. Therefore, the concept of multi-temperature cloud storage (hot, cold) was developed to improve the economics of storing the enormous amounts of data. Frequently accessed (hot) data is available on fast, high performance storage, while inactive (cold) data is archived onto lower cost storage [5].

To the best of the author's knowledge, this is the first initiative to propose a Hybrid Filtering Approach (HFA) that horizontally filters the database by hot tuples and then, vertically filters the database by defining hot attributes, in the aspect of storage optimization (reducing storage space) in main-memory cloud database. Moreover, we prove its efficiency compared to the traditional approach using standard benchmark.

The contributions of this paper can be summarized as follow:

1. Comprehensive analysis of existing main-memory databases that focus on hot/cold data management.
2. Introduce the proposed approach and explain it through a detailed case study.
3. Evaluate the effectiveness of the proposed approach using a standard benchmark.

The remaining of this paper is organized as follow. Section 2 surveys the recent related work. Section 3 introduces the proposed hybrid filtering approach. Section 4 presents a detailed case study to illustrate the workflow of the proposed approach. Section 5 reports the experimental evaluation of the proposed approach. Finally, Section 6 concludes the paper.

2. 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 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 [6], IBM's solidDB [7], and VoltDB [8]. On the other hand, research systems are HYRISE [9], H-Store [10], HyPer [11] and MonetDB [12]. 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. This problem of capacity limitation of main-memory DBMS has been addressed by a number of recent works.

SAP HANA [13] is a columnar in-memory DBMS suitable for both OLTP and Online Analytical Processing (OLAP) workloads. It offers an approach to handle data aging [14]. Hot data refers to columns that are loaded into main-memory and can be accessed by the DBMS. Cold data is not loaded into main memory but is stored in the disk-based persistence layer. It uses the Least Recently Used (LRU) technique to distinguish between hot and cold data.

Oracle Database 12c In-Memory Option [15] is based on dual-format data store, suitable for use by response-time critical OLTP applications as well as analytical applications for real-time decision-making. Oracle in-memory column store uses LRU technique to identify hot/cold data.

HyPer is a main-memory hybrid OLTP and OLAP system [11]. It has a compacting-based approach used to handle hot and cold data [16]. 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 [17], 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.

In [18], the authors implemented hot and cold separation in the 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.

A comparable approach is presented in Hekaton [19], a SQL server's memory-optimized OTLP engine that manages hot and 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 [20].

Table 1 summarizes the comparison between hot/cold data management approaches in main-memory databases. We observe that SAP HANA [14] vertically filters the data in a columnar layout, which is a different context than the row layout employed in our HFA approach. Oracle 12c dual-format [15] stores the primary copy of the data on disk, and then uses the concept of hybrid filtering in its approach. However, it applies HF and VF on disk, and then moves these hot data into in-memory columnar store. In contrast, we apply hybrid filtering approach on data resident in main-memory. HyPer [16,17] perform cold/hot data classification at the VM page level, which is different from our scope. It is proved by [18] that it is best to make the classification at the same level of granularity that the data is accessed, which is at the

Table 1 Hot/cold data management approaches in main-memory databases.

Main-memory database approach	Main-memory physical layout	Hot/cold data classification	Horizontal Filtering (HF) by hot tuple	Vertical Filtering (VF) by hot column	Hybrid filtering
SAP HANA [14]	Columnar	Hot columns	NO	YES	NO
Oracle 12c dual-format [15]	Both	Hot tuples & hot columns	YES	YES	YES
HyPer [16]	Both	Hot pages	YES	YES	YES
Stoica et al. [17]	Row	Hot pages	YES	NO	NO
Anti-caching [18]	Row	Hot tuples	YES	NO	NO
Hekaton [19]	Row	Hot tuples	YES	NO	NO
Proposed HFA	Row	Hot tuples & hot columns	YES	YES	YES

tuple-level. Compared to Anti-Caching [18], it uses the LRU technique to horizontally filter the database, while our approach uses the "datetime" key filtering method. Finally, Hekaton [19] is the work closest to our approach as it uses the same horizontal filtering methodology by hot tuples which is using the application pattern "datetime" key to split the data [21]. Therefore, we chose to build on their work and extend their architecture in order to implement our HFA.

Our novel Hybrid Filtering Approach (HFA) is based on a row store main-memory database. Our primary copy of data is entirely stored in main-memory. First, HFA horizontally filters the data by hot tuples. Then, it vertically filters the data by hot columns.

3. Proposed hybrid filtering approach

Our proposed approach is composed of two phases as shown in Fig. 1. In the first phase, the offline analysis, we classify the hot and cold attributes. Hot attributes will remain in main memory and cold ones will be moved to a cheaper secondary storage. In the second phase, the online analysis, the system interacts with users. The user enters a query and receives a response to his query. In this paper, we focus on the offline analysis phase. Comprehensive details on the online analysis phase and query profiling process will be addressed in a separate publication.

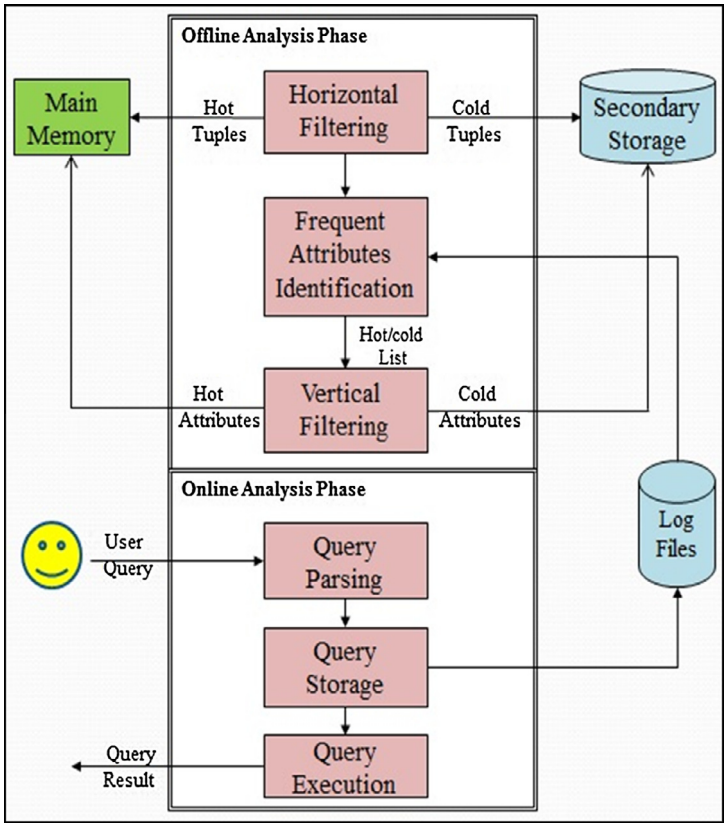

Figure 1 Proposed hybrid filtering approach.

3.1. Phase1: offline analysis

The offline phase is composed of three modules. Periodically, we run the offline analysis to define the hot and cold attributes in the log files and update the hot/cold attributes list. The duration is predefined by the system administration according to one of two factors either by time (i.e. number of months) or by database workload (i.e. number of queries).

3.1.1. Horizontal filtering

Similar to recent research work, the primary copy of database resides in main-memory and is horizontally filtered at tuple-level of granularity to hot and cold tuples. The hot tuples will remain in main-memory and the cold ones will be migrated to a cold secondary storage. In HFA, we use a horizontal filtering approach that depends mainly on the application business logic. Thus, we use the filtering pattern "datetime" key to split the data to hot/cold tuples [21].

3.1.2. Frequent attributes identification

In this module, we developed a novel technique to identify the hot/cold attributes. We analyze the queries stored in the log files to compute the frequency of occurrence for each attribute. The hot (most frequent) attributes are identified as the attributes that appear more than or equal to a pre-specified threshold. The results are stored to hot-attributes list. On the other hand, the hot attribute will be cold if its frequency is less than the pre-specified threshold. Cold attributes will be stored to cold-attributes list.

3.1.3. Vertical filtering

Vertical filtering of a table T splits it into two or more tables (sub-tables), each of which contains a subset of the attributes in T. Since many queries access only a small subset of the attributes in a table, vertical filtering can reduce the amount of data that needs to be scanned to answer the query. According to the hot-attributes list, the database in main-memory is vertically filtered at attribute-level of granularity to hot and cold attributes. The hot attributes will remain in main-memory, while the cold ones will be migrated to a cold secondary storage.

Table 4 Query log file.

Q_ID	Table name	Attributes
101	Items	Item_ID, Brand, Description, Price
101	Customers	Name, Phone
102	Customers	Name, Phone
102	Items	Item_ID, Brand, Description, Price
102	Employee	Name, Phone
103	Items	Item_ID, Brand, Description
103	Customers	Name
104	Items	Item_ID, Brand, Price, Cost
104	Customers	Name

Table 2 Items table in main-memory.

Item_ID	Brand	Description	Price	Cost	Size	UPC	Weight	Taxable
11	Nabisco	Cookies	2.25	1	20 × 2 × 18	124,576	23.5	1
12	Morries	Cigarettes	5	3	7 × 7 × 7	235,467	78	0
13	Kraft	Cheese	6	4	6 × 10 × 2	365,421	0.11	0
14	Kellog	Cereal	1	0.5	9 × 9 × 9	875,465	15	1
15	Quaker	Oatmeal	2.5	1	3 × 3 × 3	654,123	1.3	0
16	Nabisco	Crackers	4	2	4 × 4 × 4	412,678	2.4	0
17	Brand	Spagetti	0.99	0.5	2 × 2 × 2	127,896	3.4	0
18	Monte	Candy	0.5	0.1	2 × 2 × 2	345,346	6.3	1
19	Hershy	Candy	3.99	2	4 × 13 × 5	112,367	50.2	0
20	Kleenex	Tissues	2.99	1	2 × 16 × 3	224,643	32	0

Table 3 (a) Hot tuples in main-memory (b) Cold tuples on disk.

Item_ID	Brand	Description	Price	Cost	Size	UPC	Weight	Taxable
(a)								
11	Nabisco	Cookies	2.25	1	20 × 2 × 18	124,576	23.5	1
12	Morries	Cigarettes	5	3	7 × 7 × 7	235,467	78	0
13	Kraft	Cheese	6	4	6 × 10 × 2	365,421	0.11	0
14	Kellog	Cereal	1	0.5	9 × 9 × 9	875,465	15	1
15	Quaker	Oatmeal	2.5	1	3 × 3 × 3	654,123	1.3	0
(b)								
16	Nabisco	Crackers	4	2	4 × 4 × 4	412,678	2.4	0
17	Brand	Spagetti	0.99	0.5	2 × 2 × 2	127,896	3.4	0
18	Monte	Candy	0.5	0.1	2 × 2 × 2	345,346	6.3	1
19	Hershy	Candy	3.99	2	4 × 13 × 5	112,367	50.2	0
20	Kleenex	Tissues	2.99	1	2 × 16 × 3	224,643	32	0

Table 5 Attributes-frequency for Items table.

Attribute	Frequency
Item_ID	4
Brand	4
Description	3
Cost	2
Price	3
Weight	0
Shape	0
Taxable	0
Size	0
UPC	0

Algorithm 1: Query Execution
Input: User Query *Q*
Output: Query Result *R*
1. **Begin**
2. **if** all attributes in *Q* are cold attributes **then**
3. Return a view of *R* from cold storage
4. Increment attributes counters
5. **else if** all attributes in *Q* are hot attributes **then**
6. Return a view of *R* from hot storage
7. **else**
8. Return a view of *R* from both hot and
9. cold storage
10. **end if**
11. **for** each attribute in Q
12. **if** counter > = threshold **then**
13. Change cold attribute into hot attribute
14. Update hot and cold attributes lists
15. **else**
16. Change hot attribute into cold attribute
17. Update hot and cold attributes lists
18. **end if**
19. **end for**
20. Return *R* to User
21. **End**

3.2. Phase 2: online analysis

The online phase is composed of three modules.

1. *Query parsing:* This module receives the user query and parses it to identify the requested tables and attributes.
2. *Query storage:* This module stores the user query into the Log files.
3. *Query execution:* This module executes the query and returns the results to the user. The algorithm of the query execution is demonstrated using pseudo code in Algorithm 1.

4. Case study

In this section, a detailed case study is presented in order to demonstrate the proposed HFA workflow. Table 2 shows the Items table which consists of 9 attributes.

4.1. Phase1: offline analysis

1. *Horizontal filtering*: Items table is horizontally filtered to hot tuples (remain in main-memory) and cold tuples (moved to disk) as shown in Table 3.
2. *Frequent attributes identification*: Identify hot/cold attributes in the database, which involves two main steps.

Step 1: Scan the query log file shown in Table 4 in order to find the most frequent attributes in Items table.
Step 2: Employ a pre-specified attribute frequency threshold (θ) = 3 on the attribute-frequency table shown in Table 5. Thus, the hot-attributes list = [Item_ID, Brand, Description, Price] and the cold-attributes list = [Cost, Weight, Shape, Taxable, Size, UPC].

3. *Vertical filtering*: Hot tuples from Table 3(a) are vertically filtered by the hot-attributes list. Consequently, the data set for Items table in main-memory will have only hot attributes of hot tuples as shown in Table 6.

4.2. Phase 2: online analysis

Receive the user query and parse it to identify the requested tables and attributes.
Query 105:

Select Cost, Weight, Taxable
From Items;

After parsing the query, it is noticed that it fits the first case as all query attributes are cold attributes (Cost, Weight, and Taxable) (Lines 2-4 in pseudo code). We increment these attributes' counters, and then return a view of the query attributes from cold storage (disk) using the following T-SQL code sample.
First, we create a view called V1

```
CREATE VIEW V1
AS SELECT Cost, Weight, Taxable
FROM dbo.Cold_Table;
GO
```

Second, we run the view to verify its contents

```
SELECT * FROM V1;
GO
```

These attributes' frequencies will be incremented such as (Cost = 3, Weight = 1, Taxable = 1). The Cost attribute frequency is equal to the threshold (Lines 11–14) then it'll be added to hot-attributes list. Thus, the updated hot-attributes list = [Item_ID, Brand, Description, Price, Cost] and Cold-attributes list = [Weight, Shape, Taxable, Size, UPC]. Finally, the query is stored to the Log files.

5. Experimental evaluation of HFA approach

In order to systematically validate the effectiveness of our HFA approach, we have implemented it and the Hekaton

Table 6 Vertical filtering of hot tuples in main-memory.

Item_ID	Brand	Description	Price
11	Nabisco	Cookies	2.25
12	Morries	Cigarettes	5
13	Kraft	Cheese	6
14	Kellog	Cereal	1
15	Quaker	Oatmeal	2.5

approach. In this section, we present details of the experiment setup, the workload, the experiment scenario, and finally the performance study is reported.

5.1. Experiment setup

Experiments run using the following resources:

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

Software tools: Microsoft Server 2014 Enterprise Edition as a software tool to build our in-memory database and tables and run the queries. In addition, we use client statistics tool to monitor and compare the performance of our queries.

5.2. Workload

For all experiments, we use the well-known TPC-H [22] benchmark, which has been used in reputable research works [11,13,15]. The workload consists of two tables LINEITEM and ORDERS. We populated the tables with data by the official TPC-H toolkit, with scale factor SF = 1. LINEITEM table has 6,001,215 rows and the ORDERS table has 1,500,000 rows. Fig. 2 shows the tables schema. The LINEITEM table consists of 16 columns and the ORDERS table consists of 9 columns.

5.3. Experiment scenario

We base all experiments on a variant of the following query:

Select hotcol1, hotcol2...
From table
Where hotcol *operator* x;

The value of *x* is any valid value according to the data type of *hotcol*. The objective of the predicate in the *where* clause is to identify and retrieve the hot rows (e.g. *OrderDate*). The *select* clause vertically filters the table by the hot columns. The HFA workflow can be summarized as follow:

- We create two memory-optimized tables (ORDERS and LINEITEM) and store them entirely in the main-memory.
- We horizontally filter both tables by hot rows using different cases of hot rows: 25%, 50% and 75% of the original table.
- We vertically filter each horizontal table by hot columns using four different cases of hot columns. For ORDERS table: 2, 4, 6 and 8 hot columns. For LINEITEM table: 3, 7, 11 and 15 hot columns.

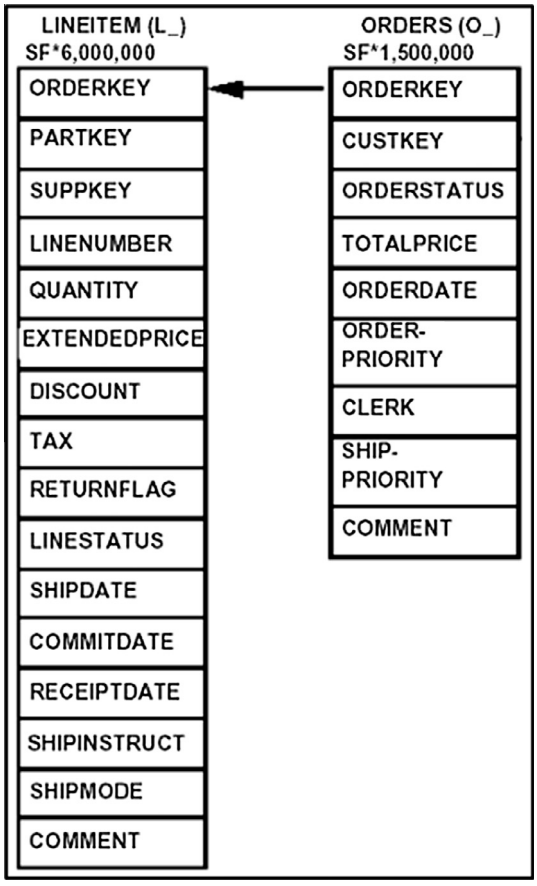

Figure 2 Tables schema.

5.4. Performance study

In this section, we experimentally evaluate the effectiveness of our HFA compared to Hekaton in terms of three performance dimensions: storage space, query elapsed time and CPU time. We have used different cases of hot rows 25–75% using step of 25 employing different cases of hot columns. In ORDERS table: HFA-2, HFA-4, HFA-6 and HFA-8 while in LINEITEM table: HFA-3, HFA-7, HFA-11 and HFA-15.

5.4.1. Storage space dimension

In this experiment, we investigate the storage space requirements of the proposed HFA compared to Hekaton in main-memory database. As shown in Fig. 3, results show that the storage space of all approaches increase with increasing number of hot rows. It is obvious that our HFA outperforms Hekaton in all cases of vertical filtering.

It can be noted that in Fig. 3(a), the best storage requirement for Hekaton is worse than the worst storage requirement for the proposed HFA approach using HFA-2 and HFA-4 hot columns. In Fig. 3(b), the best storage value for Hekaton is worse than the worst storage requirement for the proposed HFA approach using HFA-3 and HFA-7 hot columns.

As shown in Fig. 4(a), results show that our HFA outperforms Hekaton. The HFA approach has storage improvement on average 44–96% and Hekaton approach on average 25–

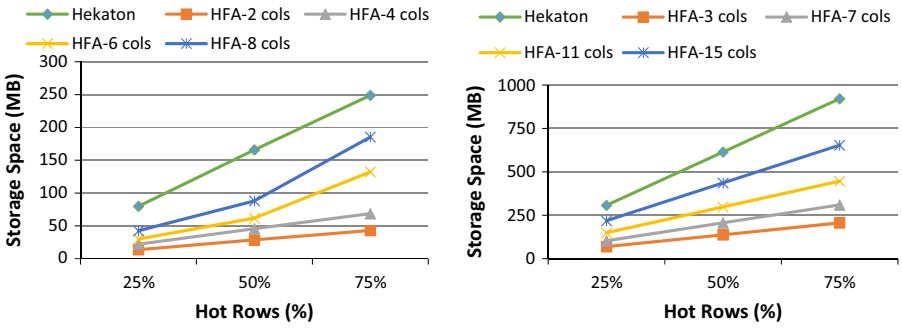

Figure 3 Storage space requirements (a) for ORDERS table (b) for LINEITEM table.

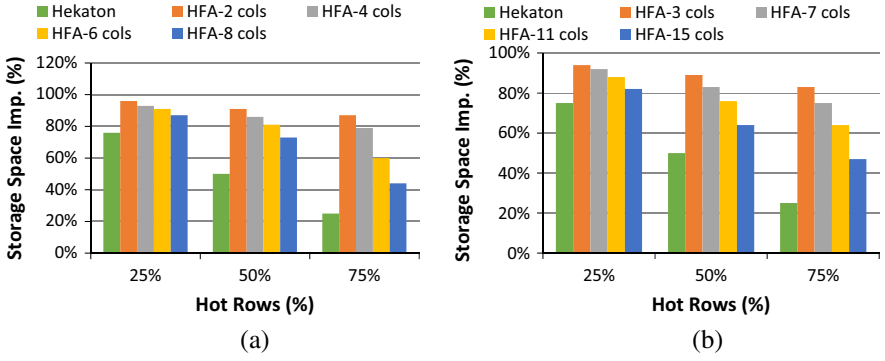

Figure 4 Storage space improvements (a) for ORDERS table (b) for LINEITEM table.

76% compared to the original ORDERS table. In Fig. 4(b), the HFA approach has storage improvement on average 47–94% and Hekaton approach on average 25–75% compared to the original LINEITEM table.

5.4.2. Query elapsed time dimension

In this experiment, we investigate the query elapsed time of the proposed HFA compared to Hekaton in main-memory database. As shown in Fig. 5, results show that the query elapsed time of all approaches increase with increasing number of hot rows. It is obvious that our HFA outperforms Hekaton in all cases of vertical filtering except in the case of HFA-8 hot columns in the case of hot rows less than 50%.

From Fig. 5(a), it can be noted that the best elapsed time value for Hekaton is worse than the best value for our proposed HFA approach using HFA-2, HFA-4 and HFA-6. In Fig. 5(b), the best elapsed time value for Hekaton is worse than the best value for our proposed HFA approach using HFA-3, HFA-7 and HFA-11 hot columns.

As shown in Fig. 6(a), results show that our HFA outperforms Hekaton. The HFA approach has elapsed time improvement on average 25–90% and Hekaton approach on average 12–74% compared to the original ORDERS table. In Fig. 6 (b), the HFA approach has elapsed time improvement on average 45–93% and Hekaton approach on average 40–81% compared to the original LINEITEM table.

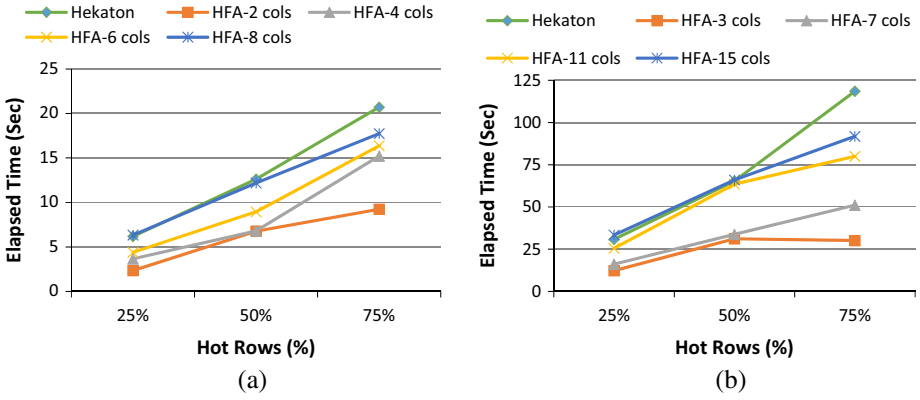

Figure 5 Query elapsed time (a) for ORDERS table (b) for LINEITEM table.

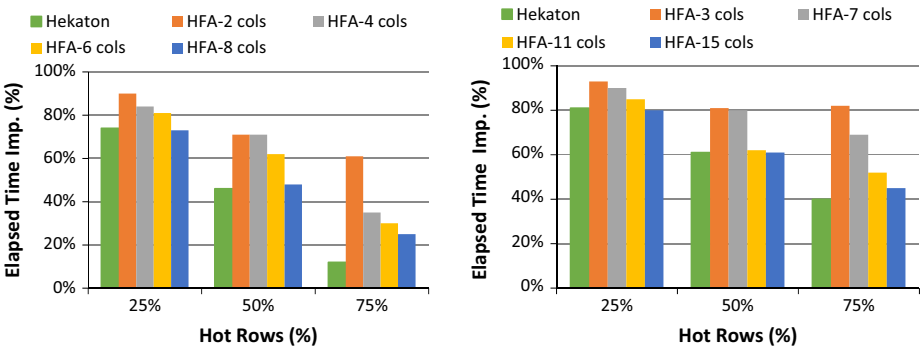

Figure 6 Elapsed time improvements (a) for ORDERS table (b) for LINEITEM table.

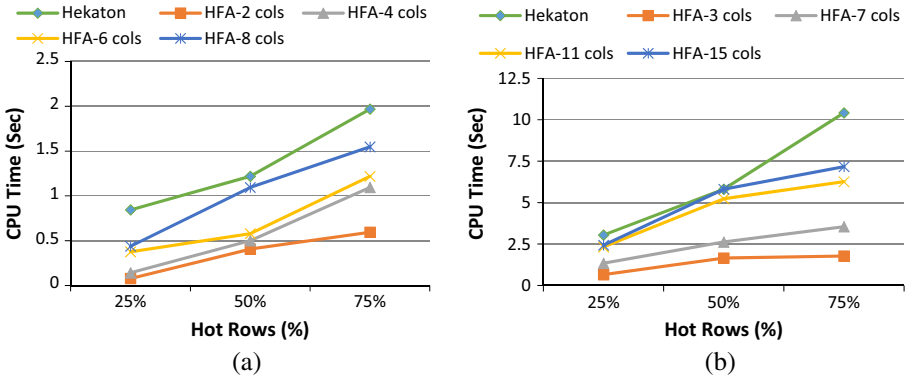

Figure 7 CPU time (a) for ORDERS table (b) for LINEITEM table.

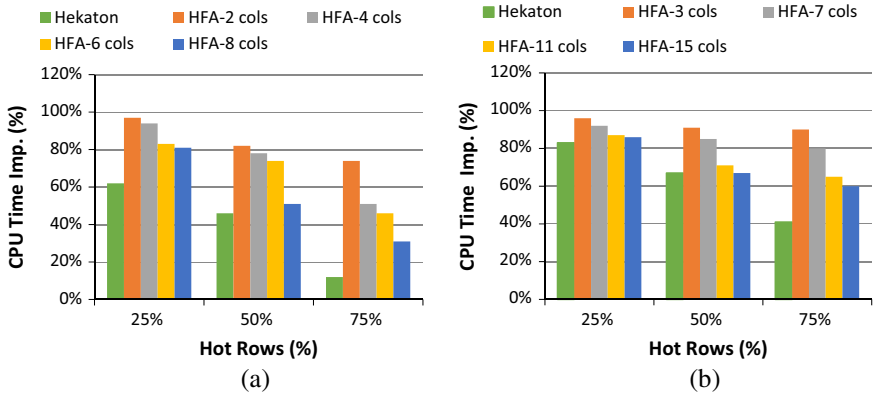

Figure 8 CPU time improvements (a) for ORDERS table (b) for LINEITEM table.

5.4.3. CPU time dimension

In this experiment, we investigate the CPU time of the proposed HFA compared to Hekaton in main-memory database. As shown in Fig. 7, results show that the CPU time of all approaches increase with increasing number of hot rows. It is obvious that our HFA outperforms Hekaton in all cases of vertical filtering except in the case of HFA-15 hot columns in the case of hot rows less than 50%.

From Fig. 7(a), it can be noted that the best CPU time value for Hekaton is worse than the best value for our proposed HFA approach using HFA-2, HFA-4 and HFA-6. In Fig. 7(b), the best CPU time value for Hekaton is worse than the best value for our proposed HFA approach using HFA-3, HFA-7 and HFA-11 hot columns.

As shown in Fig. 8(a), results show that our HFA outperforms Hekaton. The HFA approach has CPU time improvement on average 31–97% and Hekaton approach on average 12–62% compared to the original ORDERS table. In Fig. 8(b), the HFA approach has CPU time improvement on average 60–96% and Hekaton approach on average 41–83% compared to the original LINEITEM table.

6. Conclusion

Due to the budgetary challenges of storing vast amounts of data in the cloud, identifying hot/cold storage is emerging as a significant trend. To contribute to this research, we investigated the optimization of the storage space requirements with the aim of reducing the cost in main-memory cloud databases.

We conducted a comprehensive analysis of existing main-memory databases that focus on hot/cold data management. We proposed a novel Hybrid Filtering Approach (HFA) that filters the tables in the main-memory both horizontally by rows and then vertically by columns. We demonstrated its workflow through a detailed case study. We evaluated the effectiveness of HFA using the standard TPC-H benchmark.

Experimental evaluation proved that the proposed HFA approach is superior to Hekaton in terms of all performance metrics in main-memory row store database. The proposed HFA reduces the storage space by average of 44–96%, reduces the query elapsed time by average of 25–93% and reduces the CPU time by average of 31–97% compared to the traditional database approach.

References

[1] Gupta M, Verma V, Verma M. In-memory database systems-a paradigm shift. arXiv preprint arXiv, vol. 6, no. 6; December 2013.

[2] Arora I, Gupta A. Improving performance of cloud based transactional applications using in-memory data grid. Int J Comput Appl 2014;107(13).

[3] Boissier M. Optimizing main memory utilization of columnar in-memory databases using data eviction. VLDB PhD Workshop; 2014.

[4] Mark P. Buffington J, Keane M. Cloud storage: the next Frontier for tape. White paper of Enterprise Strategy Group, April 2013.

[5] Song Y. Storing big data—the rise of the storage cloud. Advanced Micro Devices, Inc., AMD; December 2012.

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

[7] 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 2013;36(2):14–20.

[8] Stonebraker M, Weisberg A. The VoltDB main memory DBMS. IEEE Data Eng Bull 2013;36(2):21–7.

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

[10] 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 2008;1(2):1496–9.

[11] 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; 2011. p. 195–206.

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

[13] Färber F, Cha SK, Primsch J, Bornhövd C, Sigg S, Lehner W. SAP HANA database – data management for modern business applications. ACM Sigmod Record 2011;40(4):45–51.

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

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

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

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

[18] DeBrabant J, Pavlo A, Tu S, Stonebraker M, Zdonik S. Anti-caching: a new approach to database management system architecture. Proc VLDB Endow 2013;6(14):1942–53.

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

[20] Delaney K. SQL server in-memory OLTP internals overview for CTP2. SQL Server Technical Article; 2013.

[21] Weiner M, Levin A. In-memory OLTP – common workload patterns and migration considerations. SQL Server Technical Article; 2014.

[22] TPC BENCHMARK™ H Standard Specification Revision 2.17.1. Transaction processing performance council; 2014. Information available at <http://www.tpc.org/tpch/>.