Naif Arab University for Security Sciences
Journal of Information Security & Cybercrimes Research
مجلة بحوث أمن المعلومات والجرائم السيبرانية
https://journals.nauss.edu.sa/index.php/JISCR
JISCR

Vertical Fragmentation for Database Using FPClose Algorithm

Arwa S. Al-Shannaq [1*], Sultan Almotairi [2]
[1] *Computer Science Department, Faculty of Computing and Information Technology, King Abdul-Aziz University, Jeddah, Saudi Arabia.*
[2] *Department of Natural and Applied Sciences, Community College, Majmaah University, Al-Majmaah, 11952, Saudi Arabia.*

Received 04 Feb. 2019 ; Accepted 06 Apr. 2019; Available Online 10 May 2019

Abstract

Vertical fragmentation technique is used to enhance the performance of database system and reduce the number of access to irrelevant instances by splitting a table or relation into different fragments vertically. The partitioning design can be derived using FPClose algorithm, which is a data mining algorithm used to extract the frequent closed itemsets in a dataset. A new design approach is implemented to perform fragmentation. A benchmark with different minimum support levels is tested. The obtained results from FPClose algorithm are compared with the Apriori algorithm.

I. Introduction

Data fragmentation is related to a process to divide a table or database into many partitions. Each partition is saved in a distributed site and many queries can be executed in different fragments. Fragmentation is useful to minimize very large database or table so it becomes easy to manage and backup, data fragmentation enhances also the performance of database systems.

Vertical fragmentation refers to a technique in which a table or relation is vertically broken into different fragments depending on the attributes. These new partitions should not lose any information from the original relation, so the reconstruction of the original relation is still possible.

Data mining is related to the methods that are used to discover the information which are implicitly exists from big tables and convert them to meaningful data. One of the fastest algorithms for extracting closed frequent items in a dataset is FPClose algorithm [13], which was implemented using FP-trees and FP-arrays structure to make it more effective. Furthermore, the generated closed itemsets number is almost smaller than the frequent items that are generated using Apriori algorithm [8].

By using FPClose technique to extract the closed frequent item sets of attributes, a vertical fragmentation can be done by grouping those closed frequent items.

The aim of this study is to perform the vertical fragmentation on the database depending on the frequent items and specifically the closed itemesets. Hence the performance of the database system will improve:

II. Related Work

Vertical fragmentation in data warehouse views and it's utility was experimentally confirmed in terms of cost reduction for the expected workload execution [1]. Elhoussaine [2], developed another approach for vertical and horizontal fragmentation based on genetic algorithm. The author aimed to reduce the cost of query execution by applying the data partitioning on star schema for relational DW. Marir et al. [3]. proposed a simple and easy algo-

Keywords: *data mining*, *partitioning*, *vertical fragmentation*.

Production and hosting by NAUSS

* Corresponding Author: Arwa S. Al-Shannaq
Email: almotairi@mu.edu.sa

doi: 10.26735/16587790.2019.005

rithm based on the affinity matrix, which started from the values of affinity between the different attributes to the step of generating the initial groups. The final fragments were produced by merging those initial groups A comparative summary was performed between various fragmentation algorithms for DW such as hill climbing and genetic algorithms [4]. Another comparison study was conducted between two methods for horizontal fragmentation with adaptation to XML context, these methods are affinity-based fragmentation [5]. Yeruva et al. provided a methodology for distributed warehouse design and vertical fragmentation for large schema based on attribute affinity matrix and bond energy algorithm [6]. Benkrid et al. proposed another fragmentation approach based on genetic algorithm for data warehouse that was modeled using a star schema and the allocation of these different partitions at several nodes [7]. Gorla and Betty designed a new methodology for vertical fragmentation in relational database based on data-mining technique by adapting the Apriori algorithm [8]. Another usage of data-mining techniques for vertical fragmentation was performed using FP-max algorithm [9] and tested on large database and warehouse [10], the performance was compared with approach based on Apriori algorithm [8]. Rakesh and A. Bhanu used a clustering algorithm called k means to perform partitioning for a dataset after horizontal aggregations [11]. Fung et al. developed cost model for query processing on object-oriented database which was implemented used vertical partitioning [12].

III. Methodology

Our approach consists of three main steps: generating the closed itemsets using the FPClose algorithm [13], then deriving the vertical partitions, and finally finding the optimal partitioning scheme.

The approach will be tested using sample database and transaction set [8] (see Appendix A)

A. Generating the closed itemsets

Closed frequent itemsets are called closed when both items have a support that is equal to -or larger than min-support, which is previously defined. An itemset is supposed to be closed in a data set unless a superset exists, which has the same support as the original itemset. The closed itemset is extracted by using the FPClose algorithm. The frequency of queries is represented by the support. At this step, the inputs are database transaction sets with transaction frequencies and a predetermined support level. The outputs will be a set of closed frequent items.

B. Deriving the potentials of vertical partitions

Once the large itemsets was generated, the partitions scheme is determined by tracing the large itemset from the k-itemset. The first partition is picked from k-itemset then the next partition is picked from (k-1) itemset to reach first set considering that the partitions are disjoint. This process should be repeated until deriving all the possible partitioning schemes. The input for this step use the list of large itemsets extracted from the previous step. The output will be the potential list of all partitioning schemes.

C. Finding the optimal portioning scheme

To find the optimal scheme, the cost formula is applied to compute the operating cost for each transaction. Equation 2 is used for (Select) retrieval transaction and equation 3 is used to update transaction such as (insert/delete). The lowest cost is related to the optimal scheme. For this module, the inputs are the list of all partitioning schemes, the set of transactions and the optimal scheme will be the output of this step.

D. Cost formula

The selecting of the optimal partitioning scheme will depend on the lowest cost reductions. The cost is calculated using the number of blocks as shown in equation1 which are accessed by a query as used in [8]. A formula for each segment will be applied which represents the cost of query processing (retrieval) as shown in equation 2. In our approach, we include the cost of blocks estimate to calculate the cost without considering the cost of storage access and index access.

$$\#\ of\ blocks = n\left(1-\left(1-{}^{1}/_{n}\right)^{k}\right) \qquad (1)$$

Where:

$Fq=$	Query q frequency
$Segqj=$	# of partitions j satisfying a query q
$Ni=$	$[T*Li/BS]$
$Ni=$	# of accessed blocks in partition i
$Kq=$	# of tuples satisfying a query
$T=$	total tuples in the dataset
$Li=$	size of partition i in bytes
$BS=$	The block size

In case of insert/delete queries, the cost will be multiplied by 2 while it takes two I/O times, which is estimated as used in [8]:

$$2 * fq * \left(\sum_{i=1}^{segqj} \left\lceil Ni \left(1 - \left(1 - {}^{1}/_{Ni}\right)^{kq}\right) \right\rceil \right) \quad (2)$$

IV. Implementation

In this approach, a real-life database from (UCI) called Teaching Assistant Evaluation (TAE) dataset with 6 attributes, 151 tuples, and a block size of 100 bytes [14] was used. The description of the database is shown in Table I.

(TAE Dataset) Experiment

The TAE dataset attributes description is given in Table I. It is assumed that the block size of this dataset to be 100 bytes. The generated transaction set as shown in Appendix A, contains 10 SELECT, one INSERT and one DELETE SQL Query with the related frequency and number of tuples satisfied the query which was used in prior research [8].

The partitions are derived using predefined support levels (20%,30%,40%,50%,60%). The extracted closed itemsets are listed in five groups with min-support of 20% as shown in Table II. The partitioning schemes then grouped from these different disjoint itemsets.

It can be noticed the better performance of FPclose algorithm over Apriori algorithm. FPclose has an improvement in cost by 15.74% whereas Apriori algorithm has 13.62% as shown in Fig. 1.

An open source library for data mining algorithms (SPMF) [15], which was developed in java used to conduct the FPclose algorithm to extract the closed items from the set of transactions. The generating time for extracting data for 20% support is 15ms, while it is zero for the other support levels.

$$fq * \left(\left(\sum_{i=1}^{segqj} \left\lceil Ni \left(1 - \left(1 - {}^{1}/_{Ni}\right)^{kq}\right) \right\rceil \right) * \left(1 + \left(0.1 * (segqj - 1)\right)\right) \right) \quad (3)$$

As shown in Table III the best partitioning scheme is obtained when the minimum support level is 20% or 30% which gives a result of 15.74% cost reductions compared with unpartitioned design.

The optimal design is:

{Speaker, Course, Semester}

{Course_instructor, Class_attribute}

{Class_size}

TABLE I
TAE Database Description

	Attributes	Types	Size (bytes)
A	speaker	(binary) English speaker=1, non-English speaker=2	19
B	course_instructor	(25 categories)	2
C	course	(26 categories)	2
D	semester	(binary) Summer=1, Regular=2	7
E	class_size	(numerical)	2
F	class_attribute	categorical) Low=1, Medium=2, High=3	6

TABLE II
Closed Itemsets for TAE With MIN_SUP=20%

Itemset L1	
Itemsets	**Support**
{ A }	57
{ B }	62
{ C }	56
{ D }	54
{ E }	39
{ F }	88
Itemset L2	
Itemsets	**Support**
{FB}	50
{FA}	55
{AC}	46
{AD}	44
{BD}	47
{CD}	43
{DE}	39

TABLE II
CLOSED ITEMSETS FOR TAE WITH MIN_SUP=20% *(Continued.)*

Itemset L3	
Itemsets	**Support**
{CDE}	28
{ADE}	29
{BDE}	32
{ACD}	33
{BCD}	36
{BAD}	37
{FAD}	42
{FAC}	44
Itemset L4	
Itemsets	**Support**
{BCDE}	21
{BADE}	22
{FADE}	27
{BACD}	26
{FACD}	31
{FBAD}	35
Itemset L5	
Itemsets	**Support**
{ABDEF}	20
{ABCDF}	24

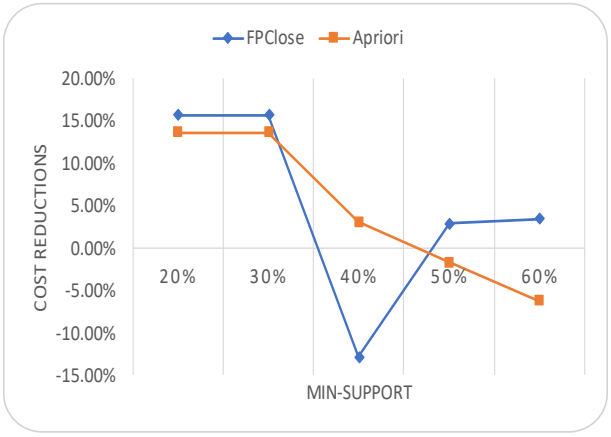

Fig. 1. Cost reduction for TAE database.

TABLE IV
COST REDUCTIONS COMPARISONS

Min-Support	FPClose	Apriori
20%	15.74%	13.62%
30%	15.74%	13.62%
40%	-12.88%	3.04%
50%	2.94%	-1.67%
60%	3.43%	-6.21%

TABLE III
RESULTS OF TAE WITH DIFFERENT SUPPORT LEVEL

Min-Support	Best Partition Schemes	Cost	Cost Reductions	Number of closed items
20%	ACD FB E	2720.30	15.74%	29
30%	ACD FB E	2720.30	15.74%	20
40%	ACF BD E	3643.80	-12.88%	13
50%	A BF C D E	3133.00	2.94%	7
60%	A B C D E F	3117.20	3.43%	2
Optimal Scheme	**ACD FB E**	**2720.30**	**15.74%**	-
Without Partitioning	**ABCDEF**	**3228.00**	-	-

V. Discussion

FPclose algorithm is trying to discover the frequent and closed itemset in the database. And if the related database is splitting regarding these kind of itemsets, the cost of query processing will be decreased. The reduction in cost will also depend on the queries and their frequencies. Moreover, the characteristics of the database have an effect while finding the optimal solutions. If there are more inserts/delete queries, more fragmented solution will be produced, because of the high cost for these queries in partitioned scheme. On the other hand, the minimum support with high level will give more fragments which will increase the cost.

Our results show a better performance against Apriori algorithm because of the less number of fragments are produced using FPclose. The cost reductions in our approach is 15.74% while using Apriori algorithm gave 13.62%.

In the real world, the proposed method in our research can be used by the database designers to obtain the efficient partitioning design.

VI. Conclusion

A new approach has been presented in this study to make a vertical fragmentation for database using FPclose data mining algorithm, which is used to extract the closed itemsets in any relation. The results show a better performance against Apriori algorithm because of the less number of fragments are produced using FPclose. The future work will use the proposed method in mix fragmentation and other large databases such as data warehouse.

References

[1] M. Golfarelli, D. Maio and S. Rizzi, “Vertical Fragmentation of Views in Relational Data Warehouses,” in *Atti del Settimo Convegno Nazionale Sistemi Evoluti per Basi di Dati (7th SEBD)*, 1999, pp. 19-33.

[2] Z. Elhoussaine, D. Aboutajdine and E. Abderrahim, “Complete algorithm for fragmentation in data warehouse,” in *Proc. 7th WSEAS Int. Conf. Artif. Intell. Knowl. Eng. Data Bases*, Feb. 2008, pp. 537-540.

[3] F. Marir, Y. Najjar, M. Y. AlFaress and H. I. Abdalla, "An enhanced grouping algorithm for vertical partitioning problem in DDBs," *2007 22nd Int. Sympo. Comput. Inf. Sci.*, Ankara, 2007, pp. 1-6, doi: 10.1109/ISCIS.2007.4456833.

[4] M. Thenmozhi and K. Vivekanandan, "A comparative analysis of fragmentation selection algorithms for data warehouse partitioning," *2014 Int. Conf. Adv. Eng. Technol. Res. (ICAETR - 2014)*, Unnao, 2014, pp. 1-5, doi: 10.1109/ICAETR.2014.7012866.

[5] H. Mahboubi and J. Darmont, “Enhancing XML Data Warehouse Query Performance by Fragmentation,” in *Proc. 2009 ACM Symp. Appl. Comput.*, Mar. 2009, pp. 1555–1562, doi: 10.1145/1529282.1529630.

[6] S. Yeruva, P. V. Kumar and P. Padmanabham, "Design of distributed warehouse-a vertical fragmentation approach," in *2015 1st Int. Conf. Next Gener. Comput. Technol. (NGCT)*, Dehradun, 2015, pp. 616-621, doi: 10.1109/NGCT.2015.7375195.

[7] S. Benkrid, L. Bellatreche and H. Drias, "A Combined Selection of Fragmentation and Allocation Schemes in Parallel Data Warehouses," *2008 19th Int. Workshop Database Expert Syst. Appl.*, Turin, 2008, pp. 370-374, doi: 10.1109/DEXA.2008.63.

[8] N. Gorla and P. W. Betty, “Vertical Fragmentation in Databases Using Data Mining Technique,” *Int. J. Data Warehous. Min.*, vol. 4, no. 3, pp. 35-53, July 2008, doi: 10.4018/jdwm.2008070103.

[9] G. Grahne and J. Zhu, “High performance mining of maximal frequent itemsets,” in *6th Int. Workshop High Perform. Data Min.*, 2003, p. 34.

[10] M. Bouakkaz, Y. Ouinten and B. Ziani, "Vertical fragmentation of data warehouses using the FP-Max algorithm," *2012 Int. Conf. Innov. Inf. Technol. (IIT)*, Abu Dhabi, 2012, pp. 273-276, doi: 10.1109/INNOVATIONS.2012.6207746.

[11] R. R. Kumar and A. B. Prasad, “K Means Clustering Algorithm for Partitioning Data Sets Evaluated From Horizontal Aggregations,” *IOSR J. Comput. Eng. (IOSR-JCE)*, vol. 12, no. 5, pp. 45-48, Jul.-Aug. 2013, doi: 10.9790/0661-1254548.

[12] Chi-wai Fung, K. Karlapalem and Qing Li, "An evaluation of vertical class partitioning for query processing in object-oriented databases," in *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 5, pp. 1095-1118, Sept.-Oct. 2002, doi: 10.1109/TKDE.2002.1033777.

[13] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," in *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 10, pp. 1347-1362, Oct. 2005, doi: 10.1109/TKDE.2005.166.

[14] M. Lichman, “{UCI} Machine Learning Repository,” 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[15] P. Fournier-Viger, “The SPMF Open-Source Data Mining Library Version 2,” in *Jt. Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, in Machine Learning and Knowledge Discovery in Databases, in Lecture Noyes in Computer Science, vol. 9853, pp. 32-35, Sept. 2016.

Appendix

Appendix A

Database Transactions

Transaction	Attributes	Frequency	Kq
SELECT	A, B, C, D, E	2	1
SELECT	A, B, D, E, F	3	137
SELECT	A, C, F	13	55
SELECT	A, B, C, D, F	15	50
SELECT	A, B, D, E, F	2	66
SELECT	A, C, D, E, F	7	0
SELECT	B, F	15	115
SELECT	A, B, D, E, F	6	21
SELECT	F	18	149
SELECT	B, C, D, E	10	17
INSERT	A, B, C, D, E, F	6	1
DELETE	A, B, C, D, E, F	3	4