ISSN 2347 - 3983
Volume 8. No. 9, September 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter166892020.pdf
https://doi.org/10.30534/ijeter/2020/166892020

An Adaptive Data Management Policy For Heterogeneous Mobile Databases

Kottilingam kottursamy[1], Hrishikesh Rajesh Deshmukh[2]
[1]Associate Professor, Department of IT, SRM Institute of Science and Technology, Chengalpattu, India
[2]Undergraduate Student, Department of IT, SRM Institute of Science and Technology, Chengalpattu, India
[1]k.kottilik@srmist.edu.in, [2]hd7716@srmist.edu.in

ABSTRACT

Mobile databases have spread into today's communication and computing, with numerous applications relying on mobile technology. It is an ever growing field and very much in demand, thus proving to be an excellent area to carry out research. With a large amount of data, management of such data is the key issue. Existing policies consist of Message Digest algorithms used along with cache replacement algorithms. However, the Message Digest algorithms have a major drawback of data replication, leading to a large overhead on the memory constraints. This report discuses a data management policy for heterogeneous mobile databases. This policy involves a Synchronization Server, which contains only the most critical of data, instead of all the data as in case of a normal server. The synchronization server data is efficiently managed by classifying it under various states based on the frequency of data access, the time of data access, i.e., which data has been actively used over a period of time and which has not been used so. This aims to facilitate synchronization of data between heterogeneous databases using cache replacement and to improve the rate of synchronization between heterogeneous mobile databases and the database servers. This policy has shown better performance than Least Recently Used and Most Frequently Used algorithms in terms of the execution time.

Key words : Message digest algorithms(MDA), Mobile databases, synchronization and data replacement.

1. INTRODUCTION

The ubiquity of the Mobile Database is expanding step by step as individuals need data even progressing in the quick evolving world. This information base innovation grants representatives utilizing cell phones to associate with their corporate organizations, crowd the required information, work in the detached mode and reconnect to the organization to synchronize with the corporate information base [1] . In this situation, the information is being drawn nearer to the applications so as to improve the exhibition and self-rule. This prompts many fascinating issues with regards to portable information base exploration and Mobile Database has become a fruitful land for some analysts. The utilization of portable information base has expanded quickly in light of the ceaseless development of the equipment gadgets with more noteworthy stockpiling limit and more controlled CPU. The manner by which portable applications access the information and oversee them is changed totally. Information are drawn nearer to them to improve the effectiveness and self-sufficiency as opposed to putting away them in a focal data set. This new style makes many inspiring issues in portable information base examination.

There are different definitions found in the writing on Mobile Databases. A few creators characterized Mobile Database as an information base that is put away on the cell phones, for example, PCs, PDAs and Cell telephones. Hardly any different creators expressed that it is a conveyed information base in which the getting to mode is portable. Some different creators portrayed Mobile information base as the association of dispersed data set, detached information base, Ad-hoc data set and broadcast plate [11]. The conveyed information base is treated as the home for portable information base and the others manage the entrance of versatile clients [16].A Mobile data set incorporates a Database Server which oversees and stores information, and gives applications, a far off information base/DBMS which oversees and stores portable information, and gives versatile applications, a Mobile Database Platform which incorporates Laptop, PDA, cellphones and so on., lastly, a two way correspondence connect between the workers and Mobile Database Servers. Essentially, it is a framework with basic and useful properties, for example, Distributed framework with versatile availability, Full information base framework capacity, Complete spatial portability, Wireless and wired correspondence ability.

There are different definitions found in the writing on Mobile Databases. A few creators characterized Mobile Database as an information base that is put away on the cell phones, for example, PCs, PDAs and Cell telephones[21]. Barely any different creators expressed that it is a circulated information base in which the getting to mode is portable. Some different creators portrayed Mobile information base as the association

of circulated data set, detached data set, Ad-hoc information base and broadcast circle. The dispersed information base is treated as the home for versatile data set and the others manage the entrance of portable clients. A Mobile information base incorporates a Database Server which oversees and stores information, and gives applications, a far off data set/DBMS which oversees and stores versatile information, and gives portable applications, a Mobile Database Platform which incorporates Laptop, PDA, cellphones and so on., lastly, a two way correspondence connect between the workers and Mobile Database Servers. Essentially, it is a framework with basic and utilitarian properties, for example, Distributed framework with versatile network, Full information base framework ability, Complete spatial portability, Wireless and wired correspondence capacity.

2. RELATED WORK

Barbara (1999) provides us a basic overview of mobile databases [3]. This research has produced interesting results in areas such as data dissemination over limited bandwidth channels, location-dependent querying of data, and advanced interfaces for mobile computers. Caching is one of the primary mechanisms to have been used in the process of synchronization and in particular mobile databases. Many caching algorithms have been proposed [2]. The more popular ones are Belady's algorithm which always discard the information that will not be needed for the longest time in the future, Least Recently Used (LRU) which discards the least recently used items first and Most Recently Used (MRU), which in contrast to the LRU, discards the most recently used items first. Further improvements were made with respect to the response time of the caching mechanisms and Fawaz (2013) provides a comprehensive analysis [4]. Mi-Young Choi (2010) provided an efficient method for synchronization of mobile databases, incorporating caching mechanisms [1]. This was based on message digest in order to facilitate data synchronization between a server-side database and a mobile database. McCormick and Schmidt (2012) focused on data synchronization patterns in mobile design. Two types of data synchronization patters were put forth, namely Asynchronous Data Synchronization and Synchronization and Synchronous Data Synchronization [8]. Competitive Analysis of Caching was the main work of Wolfson (1998) [5].This paper revolved around two objectives. First, a model for evaluating the performance of data allocation and replication algorithms in distributed databases.Secondly, an algorithm for automatic dynamic allocation of replicas to processors.Fanelli (2014) has presented a solution for efficient context data distribution, by stressing our principal design guidelines, and highlighting how the usage of different wireless modes and distributed context caching can deeply improve Context Data Management efficiency [6]. Gomaa, Messier, Williamson and Davies (2013) introduced an analytical model for estimating the cache hit ratio as a function of time [7]. A Markov chain analysis was used for estimating the instantaneous hit ratio of Infinite Cache. The proposed analysis considered a single Web cache with infinite or finite capacity. For a cache with finite capacity, two replacement policies were considered: Least Recently Used (LRU) and First-In–First-Out (FIFO). LRU ejects the least recently requested object, while FIFO ejects the object that was brought into the cache earliest. An improved version of FIFO, called FB-FIFO (Frequency based FIFO) was introduced. The cache was split into two segments, namely Protected segment and Unprotected Segment. When an object is requested for the first time, it is brought into the Unprotected Cache Segment. The Protected Segment contains those objects, which have been requested more than once, i.e., their popularity or frequency is more. When the cache size is full, the object that was brought into the cache, the earliest, would be ejected from the cache. Assuming a fixed number of objects stored at the Web server, the results show that the hit ratio reaches steady state within a period that depends on cache capacity, object expiry rate and request rate. FB-FIFO outperforms LRU and FIFO in the steady state, especially for small cache capacities. On the other hand, these results change when the Web server generates new popular objects periodically.

Hua et al., proposed MobiDNA, an adaptive scheme for dynamic Web content in a mobile computing environment. This scheme improved dynamic browsing of mobile content by lowering browsing latency and bandwidth consumption [9]. Lu et al., focused on web databases, addressing the problem of data annotation and automatic data alignment. A probabilistic method was proposed to combine the annotators. A clustering based method was used for the alignment problem [10]. Scotney et al., provided a new method for integrating aggregate views of distributed databases by using a dynamic shared ontology [11]. Parker et al., developed an approach for scaling spatial probabilistic and temporal databases. Also, algorithms for consistency checking, optimistic selection and caution selection were proposed [12]. Chen et al., proposed an approach to rank SQL queries results using users' navigational behaviour [13]. Wang et al., provided algorithms for shared data access in clouds. Dynamic programming techniques were used to present optimal solutions [15]. Gitzenis et al., developed a framework for transmitter power control and cache management. Dynamic programming and controlled markov chains were used that help in obtaining the essential performance trade-offs. The problem of data pre-fetching with power control in wireless networks was addressed [20]. Holliday et al., proposed epidemic algorithms for replicated databases. This helped in maintaining atomicity of transactions, eliminating global deadlocks achieving consistency and serializability [17].

3. PROPOSED SYSTEM

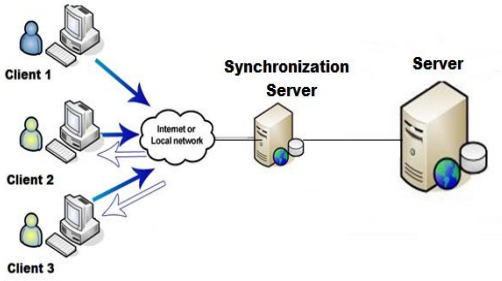

Figure 1: Overall system architecture

3.1 SYNCHRONIZATION SERVER

The primary function of the Synchronization Server is to satisfy the user's requests, either by servicing it itself or with the help of the Server. Another important function is the data management. Efficient management of data will lead to faster retrieval of data, hence leading to quicker times in servicing the user's requests. The synchronization Server classifies the data in its table with respect to certain parameters such as Frequency of data usage, Time of data access and so on. By analyzing such parameters, the Synchronization Server performs cache replacement[22-24], thereby keeping only the data, that is very essential, which is indicated by the frequency of its usage, leading to better organization and management of data. The architecture of the Synchronization can be seen clearly in Figure 2.

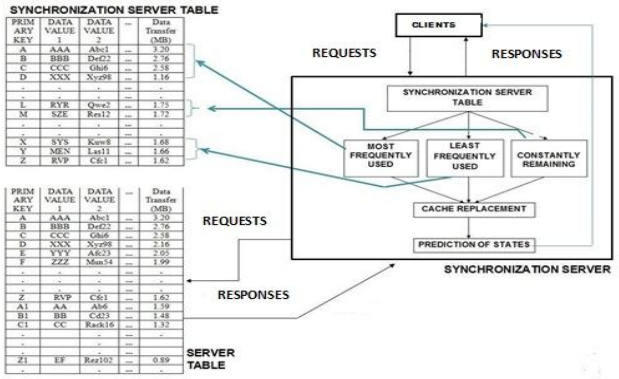

Figure 2 : Synchronization server architecture

As can be seen in Figure 2, some of the states of the data in the Synchronization Server table are Most Frequently Used Data, Least Frequently Data and other data, which are neither most frequently nor least frequently used, but they stay in the Synchronization Server more often than not. The data can move from one of those states to the other and this is done as Cache Replacement.

The Server has a large amount of data and hence accessing it every time to fetch a resource would not be the most feasible choice. This is where the synchronization server comes into play. Since it has only a portion of the data in the Server, which are the most important data, it results in quicker access, faster retrieval and efficient servicing of requests.

3.2 DATA CLASSIFICATION IN SYNCHRONIZATION SERVER

Each row in the synchronization server is classified under any of four states namely, Protected High, Protected Low, Unprotected High and Unprotected Low. The row id of each row is checked, based on which it is classified under any of the four mentioned states.

Algorithm 1: Data classification in Synchronization Server

```
1 Integer var  //# rowid of each row
2 If var less than 30
3     Classify under "PROTECTED HIGH"
4 else if var between 30 and 50
5     Classify under "PROTECTED LOW"
6 else if var between 50 and 70
7     Classify under "UNPROTECTED HIGH"
8 else
9     Classify under "UNPROTECTED LOW"
```

3.2.1 Unprotected Low state

Algorithm 2: Management of Unprotected Low data

```
1 var rowsync    //no of rows in sync server
2 if rowsync ++
3 compare (new row hit value, min_hitvalue(unprotected low))
4     if new row hit value is greater
5         replace(last row in unprotected low, new row)
6     else
7         no replacement
8 if rowsync --
9     delete row from sync table
```

If a new row is to enter the synchronization server table from the server table, the hit value of the corresponding row is compared with that of the last row in the Unprotected Low state. If it is greater, then it replaces the last row in the Unprotected Low state. Otherwise, there is no replacement. That row remains in the server. If a row in the Unprotected Low state is to be deleted, it is directly deleted from the synchronization server table and is replaced by that row which has the next best hit value.

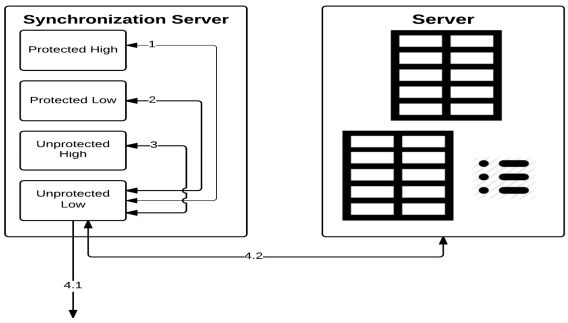

Figure 3: State transitions of data from unprotected low state

Reduction of frequency of a row from Unprotected Low which in turns moves the row to Protected High and causes corresponding replacement of the respective row from Protected Low. Reduction of frequency of a row from Unprotected Low which in turns moves the row to Protected Low and causes corresponding replacement of the respective row from Protected Low. Reduction of frequency of a row from Unprotected Low which in turns moves the row to Unprotected High and causes corresponding replacement of the respective row from Unprotected High.

Reduction of frequency of a row from Unprotected Low resulting in removal of the row from the Synchronization Server. Replacement of the row deleted from Synchronization Server with the corresponding row from Server table.

Figure 3 depicts clearly the above transitions.

3.2.2 Unprotected High state

Algorithm 3: Management of Unprotected High data

```
1  var uph        // No of rows in unprotected high
2  if uph ++
3      compare (new row hit value, min_hitvalue(Unprotected High))
4      if new row hit value is greater
5          replace(last row in Unprotected High, new row)
6      else
7          no replacement
8  if uph --
9      move row (Unprotected
```

Increase of frequency of a row from Unprotected High which in turns moves the row to Protected High and causes corresponding replacement of the respective row from Protected Low. Increase of frequency of a row from Unprotected High which in turns moves the row to Protected Low and causes corresponding replacement of the respective row from Protected Low.

Reduction of frequency of a row from Unprotected High which in turns moves the row to Unprotected Low and causes corresponding replacement of the respective row from Unprotected Low. Figure 4 depicts clearly the above transitions.

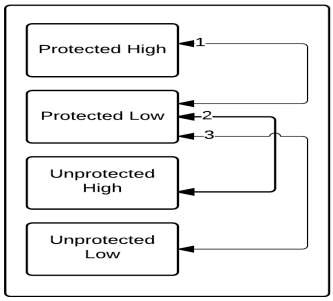

Figure 4: State transitions of data from unprotected high state

3.2.3 Protected Low state

Algorithm 4: Management of Protected Low data

```
1  var pl          // No of rows in Protected Low
2  if pl ++
3      compare (new row hit value, min_hitvalue(Protected Low))
4      if new row hit value is greater
5move (new row,corresponding position in Protected Low)for var i  new row to last row, row[i]-> row[i]+1
else no replacement     if pl - move(row,corresponding position in Unprotected High).
```

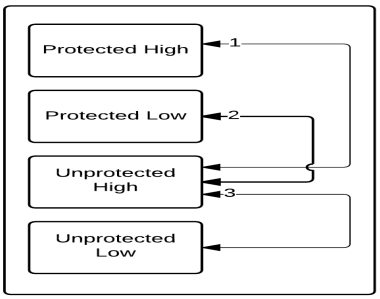

Figure 5 : State transitions of data from protected low state

1) Increase of frequency of a row from Protected Low which in turns moves the row to Protected High and causes corresponding replacement of the respective row from Protected Low.

2) Reduction of frequency of a row from Protected Low which in turns moves the row to Unprotected High and causes corresponding replacement of the respective row from Unprotected High.

3) Reduction of frequency of a row from Protected Low which in turns moves the row to Unprotected Low and causes corresponding replacement of the respective row from Unprotected Low.

3.2.4 Protected High state

Algorithm 5: Management of Protected High data

```
1  var ph      // No of rows in Protected High
2  if ph ++
3      compare (new row hit value, min_hitvalue(Protected High))
4      if new row hit value is greater
5          move (new row,corresponding position in Protected High)
6          for var i  new row to last row
7              row[i]-> row[i]+1
8      else
9          no replacement
10     if ph --
11         move(row,corresponding position in Protected Low).
```

If a new row is to enter the Protected High state from the Protected Low state, the hit value of the corresponding row is compared with that of the last row in the Protected High state. If it is greater, then it occupies the corresponding position in Protected High state, pushing the remaining rows down.

Otherwise, there is no replacement. That row remains in the Protected Low state. If a row in the Protected High state is to be deleted, it is moved to its corresponding position in Protected Low state.

1)Reduction of frequency of a row from Protected High which in turns moves the row to Protected Low and causes corresponding replacement of the respective row from Protected Low.

2) Reduction of frequency of a row from Protected High which in turns moves the row to Unprotected High and causes corresponding replacement of the respective row from Unprotected High.

3) Reduction of frequency of a row from Protected High which in turns moves the row to Unprotected Low and causes corresponding replacement of the respective row from Unprotected Low.

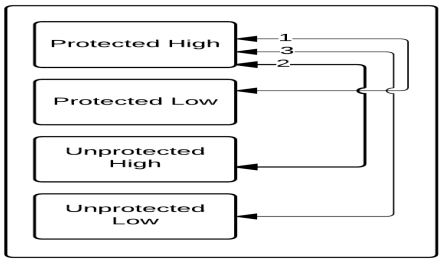

Figure 6: State transitions of data from protected high state

3.3 DATA MANAGEMENT IN SYNCHRONIZATION SERVER

3.2.1 Based On Frequency

Algorithm 6: Data Management based on Frequency

```
1  Integer var1
2  for each row i inserted into Sync table
3      var1[i] ---> 1
4  for each row i updated in Sync table
5      Increment var1[i] by 1
6  if new row enters Sync table
7      Compare(var1[new row], avg[var1])
8      Classify under one of the four
mentioned categories.
9  sort sync table based on var1
10     Display cache
```

Here, the frequency of updation of data is of primary concern. The synchronization server holds only the most critical of data, which is indicated by its frequency. So, every time a row is updated or deleted, its frequency is updated. The average frequency of the rows in the synchronization server is calculated, and it is compared with the frequency of individual rows. In case a new row is being updated more frequently, it will be put into the Synchronization server. But, as the capacity of the synchronization server is fixed, any incoming row will replace the last row of the synchronization server, i.e, the Least Frequently Used row.

3.2.2 Based On Time

Here, the time of updation of data is the main criterion. Every time a row is updated, the time at which the update took place is recorded. Also, the time of the update immediately before the most recent update is recorded. The time difference between these two updated is found for each row in the synchronization server. Also, the average time difference of the updated for the rows in the synchronization server is calculated. Any row, whose update time difference is greater than that of the average time difference is removed and replaced with a row that has a shorter update time difference than the row that is to be replaced. Hence, the Most Recently used row has the highest priority.

Algorithm 7: Data Management based on Time

```
1  time v1,v2
2  for each row i inserted into Sync table
3      store v1[i]
4  for each row i updated in Sync table
5      store v2[i]
6  for each successive update of row i
7      v2[i]----> Most recent update time
8      v1[i]----> Last but one update time
9      v3---> Calculate Difference(v2[i],v1[i])
10         compare(v3[i], avg[v3])
11             sort sync table based on var3
12     classify under one of the four categories.
13     Display cache
```

This method combines the features of the two methods discussed above. The frequency of the updation of rows is calculated as said in 1) and the synchronization server is maintained. Then the update time difference of the rows is calculated and the Synchronization server is maintained as said in 2). A new criterion called 'Popularity' is introduced here. This measures the number of different users using a particular row. So, each time a different user does some manipulation with a row, its popularity increases. In case one or some users delete a row from their respective table, the popularity of the corresponding row is decreased and the synchronization server is rearranged. So, despite a row being very frequently accessed, the reduced popularity pushes it down the synchronization table. In this way, the synchronization server is managed more effectively with respect to the three criteria discussed above.

4. RESULTS AND ANALYSIS

4.1 Comparative Analysis

A comparison is done between three different data management policies, namely, Synchronization Server, Least Recently Used replacement policy and First-In-First-Out replacement policy. Figure 15 shows the time taken to perform synchronization of 100 rows of data using the synchronization server.

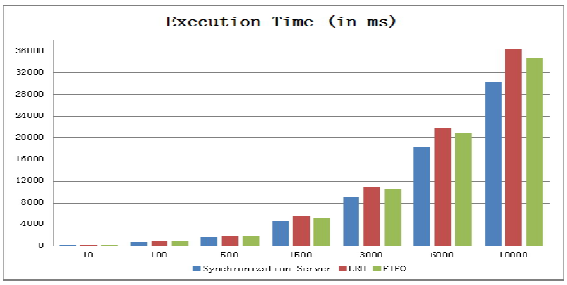

Figure 7 : Comparison of performance of various data management policies

It takes 711 ms to perform the above operation. The same operation is done using an LRU cache replacement policy. So, it can be seen that the synchronization server performs much better than LRU cache replacement. The analysis can be extended to more rows of data. Figure 15 depicts the performance of various data management policies, with respect to the execution time.

For smaller data, FIFO performs the best. It takes only 80ms for updation of 10 rows compared to 112ms for Synchronization Server and 150 ms for LRU. FIFO is the simplest of all policies, hence works very well with small data.

However, when it is increased to 100 rows, it can be seen that the synchronization server outperforms both the LRU and FIFO policies. It takes only 711ms for the Synchronization server, whereas it takes 912 and 980 ms for LRU and FIFO respectively. Further, on scaling up the number of rows to 1000 we can observe that boththe LRU and FIFO take comparatively longer time to the Synchronization Server. This is the case on increasing the rows to 1500, 3000 and

Table 1: Represents the time taken by various synchronization algorithm

1. Sync Algorithm 2. No. of 3. Rows	SYNCHRONIZATION SERVER (ms)	4. LRU 5. (ms)	6. FIFO 7. (ms)
10	112	150	80
100	711	912	980
500	1512	1810	1736
1500	4536	5430	5208
3000	9072	10,860	10,416
6000	18,144	21,720	20,832
10,000	30,240	36,200	34,720

6000. Ultimately, on increasing the row numbers to 10,000 we can observe that the time difference between the LRU and FIFO to the Synchronization server is 5960 ms and 4480ms respectively. This is due to the fact that the synchronization server has better data management by organizing the data into various states and operating only on the data essential rather than FIFO and LRU, which involve the entire set of data. The proposed data management policy performs better than both LRU and FIFO. It works well especially with large amount of data, due to its efficient management.

5. CONCLUSION

A data management policy for mobile databases is proposed using a Synchronization server. This has made organization of data efficient and replacement of cache better, thereby improving the retrieval rate of data. The proposed policy has proven to be better than existing cache replacement policies such as LRU and FIFO terms of the response time. This makes it very viable for applications that give most importance to data management.This algorithm can be applied to mobile devices such as PDAs, Tablets and so on, to speed up the synchronization of data between them, while taking into account the most important aspect, 'Mobility'.

REFERENCES

1. Mi-Young Choi, Eun-Ae Cho, Dae-Ha Park, Chang-Joo Moon, Doo-Kwon Baik, "**A Database Synchronization Algorithm for Mobile Devices",** in IEEE Transactions on Consumer Electronics, Vol. 56, No. 2, May 2010, pp 392-398.

2. K. Kedzierski, M. Moreto, F.J. Cazorla, M. Valero **"Adapting cache partitioning algorithms to pseudo-LRU replacement policies",** Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium, 19-23 Apr. 2010, pp 1-12.

3. Daniel Barbara, **"Mobile Computing and Databases- A survey",** in IEEE Transactions on Knowledge and Data Engineering, Vol 11, No 1, Jan. 1999, pp 108-117.

4. KassemFawaz, Hassan Artail, **"DCIM: Distributed Cache Invalidation Method for Maintaining Cache Consistency in Wireless Mobile Networks",** in IEEE Transactions On Mobile Computing, Vol. 12, No. 4, Apr. 2013, pp 680-693.

5. Ouri Wolfson, Yixiu Huang, **"Competitive Analysis of Caching in Distributed Databases "**, in IEEE Transactions On Parallel And Distributed Systems, Vol. 9, No. 4, Apr. 1998, pp 391-409.

6. M. Fanelli, Foschini, L. ,Corradi, A., Boukerche, A. **, " Self-Adaptive Context Data Management in Large-Scale Mobile Systems",** in IEEE Transactions on computers, Vol. 63, No 10 , Oct. 2014, pp 2549 – 2562.

7. HazemGomaa, Geoffrey G. Messier, Carey Williamson and Robert Davies **"Estimating Instantaneous Cache Hit Ratio Using Markov Chain Analysis",** in IEEE/ACM transactions on Networking, Vol. 21, no. 5, Oct. 2013 , pp 1472 – 1483.

8. Zach McCormick and Douglas C. Schmidt**, "Data Synchronization Patterns in Mobile Application Design"** , in proceedings of the Pattern Languages of Programs (PLoP) 2012 conference, Oct. 19-21.

9. Zhigang Hua, Xing Xie,Hao Liu,Hanqing Lu,Wei-Ying Ma, "**Design and Performance Studies of an Adaptive Scheme for Serving Dynamic Web Content in a Mobile Computing Environment" ,** in IEEE transactions on Mobile Computing, Vol. 5, No. 12, Dec. 2006, pp 1650-1662.

10. Yiyao Lu, Hai He, Hongkun Zhao, WeiyiMeng, Yu, C., **"Annotating Search Results from Web Databases" ,** in IEEE Transactions on Knowledge And Data Engineering, Vol.25, No 3, Mar. 2013, pp 514-527.

11. McClean, S., Scotney, B., Greer, K., **"A scalable approach to integrating heterogeneous aggregate views of distributed databases",** in IEEE Transactions on Knowledge And Data Engineering, Vol.15, No 1, Jan. 2003, pp 232-236.

12. Parker, A., Infantes, G., Grant, J., Subrahmanian, V.S., **"SPOT Databases: Efficient Consistency Checking and Optimistic Selection in Probabilistic Spatial Databases" ,** in IEEE Transactions on Knowledge And Data Engineering, Vol.21, No 1, Jan. 2009, pp 92-107.

13. Zhiyuan Chen, Tao Li, Yanan Sun, **"A Learning Approach to SQL Query Results Ranking Using Skyline and Users' Current Navigational Behavior",** IEEE Transactions on Knowledge And Data Engineering, Vol.25, No 12, Dec. 2013, pp 2683-2693.

14. Weigang Wu, Jiannong Cao, Xiaopeng Fan**, "Design and Performance Evaluation of Overhearing-Aided Data Caching in Wireless Ad Hoc Networks",** in IEEE Transactions on Parallel And Distributed Systems, Vol.24, No 3, Mar. 2013, pp 450-463.

15. Yang Wang, Veeravalli, B., Chen-KhongTham "On Data Staging Algorithms for Shared Data Accesses in Clouds", in IEEE Transactions on Parallel And Distributed Systems, Vol.24, No 4, Apr. 2013, pp 825-838.

16. Bassiouni, M.A., **"Single-site and distributed optimistic protocols for concurrency control",** in IEEE Transactions on Software Engineering, Vol.14, No 8, Aug. 1998, pp 1071-1080.

17. Holliday, J., Steinke, R., Agrawal, D., El Abbadi, A., **"Epidemic algorithms for replicated databases"**, in IEEE Transactions on Knowledge And Data Engineering, Vol.15, No 5, Sept. 2003, pp 1218-1238.

18. Hui Chen, Yang Xiao, Shen, Xuemin, **"Update-Based Cache Access and Replacement in Wireless Data Access",** in IEEE Transactions On Mobile Computing, Vol. 5, No. 12, Dec. 2006, pp 1734-1748.

19. Xin Yu, **"Distributed cache updating for the dynamic source routing protocol",** in IEEE Transactions On Mobile Computing, Vol. 5, No. 6, Jun. 2006, pp 609-626.

20. Gitzenis, S., Bambos, N., **"Joint Transmitter Power Control and Mobile Cache Management in Wireless Computing"**, in IEEE Transactions On Mobile Computing, Vol. 7, No. 4, Apr. 2008, pp 498-512.

21.V.Nallarasan,Dr.K.Kottilinagam, **"Enhanced security in IoT Networks using ensemble learning methods-A Cognitive Radio Approach"**, International Journal of Emerging Trends in Engineering Research, Volume 8. No. 8, August 2020.

22. Kottursamy, Kottilingam, Gunasekaran Raja, and K. Saranya. "**A data activity-based server-side cache replacement for mobile devices**." Artificial Intelligence and Evolutionary Computations in Engineering Systems. Springer, New Delhi, 2016. 579-589.

23. Kottursamy, Kottilingam, Gunasekaran Raja, Jayashree Padmanabhan, and Vaishnavi Srinivasan. "**An improved database synchronization mechanism for mobile data using software-defined networking control.**" Computers & Electrical Engineering 57 (2017): 93-103.

24. Raja, Gunasekaran, Kottilingam Kottursamy, Sajjad Hussain Chaudhary, Ali Hassan, and Mohammed Alqarni. "**SDN assisted middlebox synchronization mechanism for next generation mobile data management system**." In 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 1-7. IEEE, 2017.