Hot and Cold Data Identification using Query Aware Hybrid Partitioning

by

Jai Jai Kanwar
201311038

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

June, 2015

DA-IICT

Declaration

I hereby declare that

(i) The thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at DA-IICT and has not been submitted elsewhere for a degree,

(ii) Due acknowledgement has been made in the text to all the reference material used.

Signature of Student
Jai Jai Kanwar

Certificate

This is to certify that the thesis work entitled **Hot and Cold Data Identification using Query Aware Hybrid Partitioning** has been carried out by **Jai Jai Kanwar** (**201311038**) for the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology under my supervision.

Thesis Supervisor
Dr. Minal Bhise

Acknowledgements

I would like to express my deep gratitude towards my thesis advisor Dr. Minal Bhise for her guidance, advice and immense support in the thesis work.
I would like to thank Dr. P M Jat for his kind support. I would also like to thank Ms. Trupti Padiya for helping me with experiments during my thesis work.

Last but not the least I am grateful to my parents for their constant support and encouragement which has helped in the completion of my thesis work.

Jai Jai Kanwar
(201311038)

Contents

Abstract

The price of main memory is reducing with time, which helps to store huge amount of data in the memory. OLTP applications have large database size. It is observed that some applications exhibit skewed access pattern i.e. not all the records are accessed every time. Older data is less likely to be accessed as compared to the recent data. The objective is to store data in such a way so that it makes optimal utilization of memory and helps in faster query execution. To identify this hot and cold data we have proposed a Query Aware approach using Hybrid Partitioning (QAA-HP) approach. For given query workload, QAA-HP identifies the hot schema and the hot data corresponding to it. The hot data and the cold data can be configured differently so that their directed queries are accelerated. Different configuration techniques like vertically partitioned table or binary tables, n-ary tables, horizontally partitioned tables are presented for this purpose. We have used TPC-C benchmark for our experiments, which is an OLTP workload. Initially tables are vertically partitioned for hot schema and then further partitioned horizontally for hot data. Metrics for performance analysis are designed based on Query Analysis and Query Execution Time. The results show that when taking 9% of the TPC-C data in clusters, 79% of the hottest query workload QW is answered. The percentage of time gain $TG\%$ for hottest queries when run on hot clusters is observed to be 37% for cold runs and 31% for hot runs.

Keywords: Cold data, Data Partitioning, Hot data, Main Memory Databases, Query Aware Approach

List of Principal Symbols and Acronyms

f_i	Frequency for i^{th} Query
QL	Query Load
QW	Query Workload
$(QET)_{HotCluster}$	Query Execution Time for Clusters
$(QET)_{original}$	Query Execution Time for Original Data
TG	Time Gain
TTS	Total Time Saved

Other minor symbols are defined at first occurrence; wherever necessary some symbols are redefined in the text.

AUM	Attribute Usage Matrix
AAM	Attribute Affinity Matrix
BEA	Bond Energy Algorithm
CV	Confidence Value
CAM	Clustered Affinity Matrix
DRDB	Disk Resident Databases
MMDB	Main Memory Databases
OLTP	Online Transaction Processing
QET	Query Execution Time
QAA-BEA	Query Aware Approach using Bond Energy Algorithm
QAA-ARM	Query Aware Approach using Association Rule Mining
QAA-HP	Query Aware Approach using Hybrid Partitioning
SV	Support Value
TPC	Transaction Processing Performance Council
VP	Vertical Partition

List of Tables

List of Figures

Chapter 1

Introduction

With the increase in memory and storage size it has become easier for data warehouse to maintain vast historical data. The data should be stored efficiently so that it can reduce the query processing time. The main memory databases (MMDB) are also becoming popular these days. With the increase in size of main memory it is possible to store entire data in memory. MMDB can provide faster access so storing data in main memory can be helpful in providing faster execution. There are applications which have either large database size or exhibit skewed access patterns. One of the examples is of OLTP workloads which has a skewed access pattern where some data is accessed very frequently known as hot data and the other part of data is not so frequently used i.e. cold data. If the data can be divided into two parts viz. hot data and cold data, it will help in managing the data in memory and also improving the query execution time. We can also store the hot data in main memory providing faster access to data and optimally utilizing the memory at the same time. There are different ways to identify hot and cold data.

The Query aware method is the approach we have proposed for identifying hot and cold data. The approach makes use of the query workload information. Each query subjected to the application is observed and analysed. The query information helps in analysing those attributes which are frequently queried. The frequent attributes are the basis for identification of hot data. We need to observe the query frequency and using the query frequency, find out the attribute frequencies. The most frequently occurring queries indicate that the data processed by them is hot. The attribute frequency helps in identifying hot schema and cold schema where hot schema implies the set of attributes which are frequently occurring together. Once the hot schema is identified using query information the hot data needs to be identified and mapped to the hot schema. Similarly, cold data is mapped to the cold schema. Identifying hot data corresponding to hot schema further optimize the size of hot data set.

1.1 Hot and Cold Data

The data has been categorized into two parts, hot data and cold data. The data which is frequently accessed and useful is hot data whereas the data which is rarely or never accessed is termed as cold data. The identification of hot and cold data helps in applications which exhibit skewed access patterns as discussed above. The example of such OLTP application having natural skew pattern is an e-commerce website where some items are more popular than others. The popularity changes but not very rapidly. So the popular items can be stored together as hot data so that complete data need not to be processed by the query and it will also help in improving query processing time. The other types of applications are those which exhibit time-correlated skew. In such type of applications a part of data is frequent only for fixed amount of time and once the task is completed those records are rarely accessed. The example for such application can be a courier service agency. It accesses the data of customer when some product is to be delivered to the customer i.e. it will require to check and update the status of courier until reached. This application show time related skew as the courier agency will process the data of customer only till the courier is in transit, once the product reaches the customer it may be checked for some time and then rarely or never accessed.

1.2 Objective

The objectives of the thesis are:

- To identify hot and cold data
- Configuring hot data and cold data using storage techniques
- Executing queries on hot data as well as actual data and comparing the query execution time (QET) for both.

The work has been done for disk based systems which can be further extended to main memory systems.

Chapter 2 discusses the literature survey done for the thesis work and research issue. The details of available methods that can be used to solve our objective are discussed in chapter 3. Chapter 4 contain details of proposed solution, experimental setup, dataset and query sets. The results and analysis have been discussed in chapter 5.

Chapter 2

Literature Survey

The working of any database application depends on how the data is stored and managed. The applications having large and voluminous data are difficult to handle. One of the examples of such data is RDF (Resource Description Framework) data or semantic web data [1]. RDF data is stored in the form of triples <subject, property, object>. This section discusses about different partitioning techniques available and can be useful in storing hot and cold data. It also discuss about main memory databases and its impact.

2.1 Partitioning Techniques

The RDF triples can be stored as three columns relational tables as shown in figure 2.1. The data is stored in a single table so it involves more number of self joins, which result is high query execution time.

2.1.1 Property Table Approach (n-ary table) [3]

The property table can be categorized into two types, one is a clustered property table and the other is property-class table. In the clustered property table, all the properties that are defined together or queried together are clustered and kept in one single table. It reduces the need for join as all the properties can be accessed from one single table.

Subject	Property	Object
ID1	Type	Book
ID1	Title	“xyz”
ID1	Author	“Rox, John”
ID1	Copyright	“2001”
ID2	Type	CD
ID2	Title	“ABCD”
ID2	Artist	“Orr, Tim”
ID2	Copyright	“1985”
ID2	Language	“French”
ID3	Type	Book
ID3	Title	“MNOP”
ID3	Language	“English”
ID4	Type	DVD
ID4	Title	“MNO”
ID5	Type	CD
ID5	Title	“ghi”

Table 2.1: Example Triple Store [3]

In case of Property-class table, the property which is common to many subjects is classified and all those subjects are kept together in one table.

Given the triple store example in Table 2.1, the property table will look like the one in Table 2.2. There are chances of null values in property tables as it is not necessary that corresponding to every property the subject must have a value so it will store null values for those properties.

Subject	Type	Title	Copyright
ID1	Book	“XYZ”	“2001”
ID2	CD	“ABCD”	“1985”
ID3	Book	“MNOP”	NULL
ID4	DVD	“DEF”	NULL
ID5	CD	“GHI”	“1995”
ID6	Book	NULL	“2004”

Table 2.2: Clustered Property Table [3]

2.1.2 Vertically Partitioning Approach [3]

It is also called binary table where triples are stored in the form of two column table. There are n tables for n unique properties. It is decomposed storage model where data for a particular property is kept in one table. If a query wants to access data for a specific property there will be no need for joins and query can be answered from single table only.

Table: Type

ID1	Book
ID2	CD
ID3	Book
ID4	DVD
ID5	CD

Table: Title

ID1	“XYZ”
ID2	“ABCD”
ID3	“MNOP”
ID4	“DEF”
ID5	“GHI”

Figure 2.1: Vertical Partitioned Tables [3]

While accessing more than one property there will be simple merge joins. This approach is simpler than property table approach as clustering algorithms used for property table need to be carefully designed whereas vertical partitioning (VP) approach is straightforward. The results show that VP works 6-7 times better than triple store [3].

To enhance the performance further in vertical partitioning approach, materialized views have been used on top of it [4]. In triples if we have some inference queries it will require subject-object joins. To avoid self-joins materialized view is used which store the result beforehand. The main reduction in time by using materialized view based query is because a fewer number of tuples needs to be traversed and less join operations are needed to be performed at runtime. As views have fewer tuples than the base table because they are formed by imposing condition(s) on the base table. Also in the views we can store beforehand the join outputs in the form of an attribute. The results show that there is a gain of 15% (for cold runs) and 26% (for hot runs) in query execution time when materialized view is used with vertical partitioning [4].

2.1.3 Horizontal Partitioning Approach [5]

Horizontal partitioning divides the database table into multiple tables. Each table consist of same number of columns but fewer rows. A given row in a table is mapped to a partition number, so all the rows mapped to the same partition number are stored in same partition. The table can be partitioned on the basis of range. Considering the example of triple store in Figure 2.1 the horizontal partitioning can be done on the basis of subject column containing IDs. The ID from range 1 to 2 can be part of one table and so on. So the database table is divided into multiple tables having same number of columns but fewer rows.

Subject	Property	Object
ID1	Type	Book
ID1	Title	“xyz”
ID1	Author	“Rox, John”
ID1	Copyright	“2001”
ID2	Type	CD
ID2	Title	“ABCD”
ID2	Artist	“Orr, Tim”
ID2	Copyright	“1985”
ID2	Language	“French”

Subject	Property	Object
ID3	Type	Book
ID3	Title	“MNOP”
ID3	Language	“English”
ID4	Type	DVD
ID4	Title	“MNO”
ID5	Type	CD
ID5	Title	“ghi”

Figure 2.2: Horizontal Partition Tables

2.1.4 Structure Index Partitioning Approach [5]

Structure index partitioning divides the RDF triples on the basis of characteristics of data. RDF data is analysed based on the unique properties it has i.e. subjects which are associated with same properties are grouped together. This grouping of triples on the basis of subjects and properties is referred as structure. The lookup table is used to keep the record of different structures and it consists of property list and index to the structures.

2.1.5 Hybrid Data Aware Approach

The analysis done in [5] shows that different types of queries show different behaviour with partitioning approaches. Some queries give a better result with property table approach while some give better results with vertical partitioning approach. A hybrid approach proposed in [6] uses both binary table and property tables instead of partitioning whole data into binary tables. This hybrid approach can be useful for arranging the identified hot data and cold data in memory. The data once analysed on the basis of queries can be stored in memory in the form of either property table (also called n-ary table) or binary table.

These different partitioning techniques can be useful in storing the identified hot and cold data more effectively. For databases which are large in size, it is not necessary that all the data will be accessed. If data needs to be stored in main memory than keeping only the part of data which is important helps in the optimal utilization of main memory. The query aware approach helps in identifying such data and the data can then be stored using appropriate storage technique.

2.2 Clustering Techniques

The data partitioning techniques show better performance for RDF data as compared to triples store. There are different types of queries for RDF data like single triple pattern, path queries, star queries, entity queries [5]. With the types of queries, the partitioning techniques may also vary. It is understood that queries which are user oriented and involve no join, execute faster in horizontal partitioning [9]. Similarly, there are various clustering techniques which cluster properties on the basis of query types and data information. In this section, we will discuss two clustering techniques viz. Query Aware Approach using Bond Energy Algorithm (QAA-BEA) and Data Aware Approach using Data Centric Method (DAA-DCM).

2.2.1 Query Aware Approach using BEA (QAA-BEA) [2]

In this technique, the initial query workload is analyzed to find the group of frequently-queried and related attributes that can be kept in n-ary relation. The method uses a two phase algorithm to find clusters of attributes. Phase 1 is a vertical partitioning phase which aims to reduce joins in a query by clustering the queried attributes together. Phase 2 is an adjustment phase which analyzes the query workload in a dynamic environment and applies required changes to the schema.

Phase 1: Vertical Partitioning Phase

From the available query set, the set of attributes is subdivided into clusters. Given a set of relations $R = \{R_1, R_2, \ldots, R_n\}$ and a set of queries $Q = \{Q_1, Q_2, \ldots,$

$Q_m\}$, we need to identify the set of fragments F = $\{F_1, F_2, \ldots, F_x\}$ such that every fragment represents a subset of the attributes of a relation and each attribute should appear exactly in one cluster. The approach exploits the affinity measure (how often a group of attributes are accessed together in the queries of a representative workload) within a set of attributes.

Step 1: The first step is to build an Attribute Usage Matrix (AUM). AUM is a matrix where each row represents a query q in the access workload, each column represents a unique attribute A of the dataset. Each 0/1 entry in the matrix reflects whether a given attribute is accessed by a given query or not. Figure 2.3 shows the example matrix for AUM.

	A1	A2	A3	A4	A5
Q1	1	1	0	0	0
Q2	0	1	1	0	1
Q3	1	0	0	1	1

Figure 2.3: Attribute Usage Matrix (AUM)

In Figure 2.3 it is given that query Q1 access attributes A1, A2. The size of the matrix depends on number of queries (m) and number of attributes (n) i.e. m x n.

Step 2: Using AUM, derive an Attribute Affinity Matrix (AAM). It is a symmetric square matrix where each row/column represent a unique attribute of the RDF dataset and it stores the affinity between a pair of attributes (ai, aj) as aff (i , j).

	A1	A2	A3	A4	A5
A1	30	0	10	20	0
A2	0	60	20	0	30
A3	10	20	40	12	25
A4	20	0	12	20	0
A5	0	30	25	0	90

Figure 2.4: Attribute Affinity Matrix (AAM)

Affinity gives the measure of how often a group of attributes are accessed together. The high value of affinity for attribute pair indicates that the two attributes should belong to the same fragment. The value of aff (i , i) represent the total access number for each attribute ai in the whole query workload. Figure 2.4 gives an example for AAM.

The diagonal in AAM gives affinity measure for the individual attribute, for example, attribute A1 has total occurrence frequency of 30 and it appears 20 times with attribute A2.

Step 3: Clustering Phase
The AAM is used in clustering phase to build Clustered Affinity Matrix (CAM). The Bond Energy Algorithm (BEA) is used for clustering. BEA takes as input the attribute affinity matrix, permutes its rows and columns and generates a clustered affinity matrix (CAM). It follows the following steps:

Initialization: in the initialization step it place and fix one of the columns of AAM arbitrarily into CAM.
Iteration: Once the initialization is done, the remaining n-i columns are picked and placed at the remaining i+1 positions in the CA matrix. The algorithm chooses the place for columns where it makes greatest contribution to the global affinity measure. The aim of maximizing the affinity measure (AM) it to cluster large values of the matrix together as these attributes are queried more. The formula of AM is given as:

$$\boldsymbol{AM} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{aff(A_i, A_j)}\ [\boldsymbol{aff(A_i, A_{j-1})} + \boldsymbol{aff(A_i, A_{j+})} + \boldsymbol{aff(A_{i-1}, A_j)} + \boldsymbol{aff(A_{i+1}, A_j)}] \qquad \textit{\textbf{Eq. (2.1)}}$$

Row ordering: after the column ordering is done; also change the row ordering so that their relative positions match the relative positions of the columns.

Figure 2.5 gives an example for clustered affinity matrix.

	A5	A2	A3	A1	A4
A5	90	30	25	0	0
A2	30	60	20	0	0
A3	25	20	40	10	12
A1	0	0	10	20	12
A4	0	0	12	20	20

Figure 2.5: Clustered Affinity Matrix

The CAM obtained from clustering phase groups attributes on the basis of their affinity measures. The example given in Figure 2.5 reflects that A5, A2, A3 are accessed by most of the queries and have the possibility of being stored together.

Step 4: Partitioning Phase:

The Partitioning phase uses clustered affinity matrix to produce two non-overlapping clusters. It selects one point P along the diagonal of the CAM. The point splits the CAM into two blocks i.e. an upper one (U) and a lower one (L), each block gives the set of attributes to be kept in property table. The partition point is decided by selecting the point that maximizes the Z value

$$\boldsymbol{Z = (QU * QL) - QI^2} \qquad \textit{Eq. (2.2)}$$

Here, QU is the total number of queries that need to only access the attributes of the upper fragment, QL is the total number of queries that need to only access the attribute of the lower fragment and QI is the total number of queries. If we consider the example of CAM given in Figure 2.5 then the partition point can be selected between attribute A3 and A1 as it divides the matrix into two clusters, upper and lower. The selection of a splitting point is shown below with the help of a figure.

	A5	A2	A3	A1	A4
A5	90	30	25	0	0
A2	30	60	20	0	0
A3	25	20	40	10	12
A1	0	0	10	20	12
A4	0	0	12	20	20

Figure 2.6: Example CAM with sample splitting point

Phase 2: Pivoting / Unpivoting Phase:

The query workload characteristics change continuously in the dynamic environment. An adjustment mechanism is triggered periodically that re-analyze the workload information and re-apply the steps of the vertical partitioning phase. The pivot operator is used to move data from the triple store to the property tables and the unpivot operator move back the data from the property tables to the triple store.

This approach is used for clustering attributes on the basis of their affinity measure. At the end of the algorithm, we will get clusters of attributes which are mostly queried together.

2.2.1.1 Challenges in BE Algorithm

The size of attribute affinity matrix can be very large in the case of datasets having large number of attributes. To manage such a matrix will require high space complexity. Also in the partitioning phase, if m-way partitioning is used instead of 2-way than the time complexity rises to O (2^m). This is so because in m-way partitioning it is necessary to try 1, 2… m-1 split points and for each of these, it is necessary to check which place maximizes 'z'.

2.2.2 Data Aware Approach using Data Centric Method [8]

In the Data Centric Approach, the information about the data is used to generate a relational schema which helps in improving query performance. The data centric approach achieves its goal by reducing the need to join tables in a query by storing more number of properties together, and to reduce null values, a null threshold is set. The method involves two phase algorithm viz. clustering and partitioning. The clustering phase is used to cluster properties which are defined for common subjects. The properties forming one cluster are stored as property tables and properties which are not defined with other properties should be kept as binary tables.

2.2.2.1 Data Structures

The algorithm uses *Support threshold ($Thresh_{sup}$)* and *Null threshold ($Thresh_{null}$)*. Support threshold measures the strength of correlation between properties in the RDF data. The set of properties satisfying the support threshold value are candidates for same property tables. The null threshold gives the percentage of null values allowed in an n-ary table.

This algorithm needs two data structures, *Property usage list (PU)* and *Subject-Property basket (B)*. The property usage list is a list structure which stores all the unique properties and corresponding to them stored the count of subjects that have that property defined. For example if a property P_1 is defined for 4 subjects then it entry in property usage list will be ($P_1 \rightarrow 4$). The subject-property basket is list of all subjects mapped to their associated properties denoted as subjectID $\rightarrow$ $\{P_1,\ldots,P_n\}$ where P_1 , P_n are unique RDF properties.

2.2.2.2 Algorithm

The algorithm consists of two phases (1) Clustering and (2) Partitioning. These are described below.

Phase 1 Clustering

The clustering phase creates clusters of properties using subject-property basket and support threshold. It scans all the subject-property basket and find group of properties which occur often in the basket. The group of properties are considered clusters only if they satisfy support threshold value. If the support value for a group of properties is less than support threshold then they are not stored as one cluster. Figure 2.7 gives the pseudo code for clustering algorithm.

```
1: Function Cluster(Baskets B,Usage PU,Thresh_sup,Thresh_null)
2: Clusters ← GetClusters(B, Thresh_sup)
3: Tables ← properties not in PC /* Binary tables */
4: for all c1 ∈ PC do
5:   OK ← false
6:   if Null%(c1, PU) ≤ Thresh_null then
7:     OK ← true
8:     for all c2 ∈ PC if c1 ∩ c2 ≠ φ then OK ← false
9:   end if
10:  if OK then Tables ← Tables ∪ c1; Clusters ← Clusters − c1
11: end for
12: return Tables,Clusters
```

Figure 2.7: Pseudo Code for Clustering Phase [8]

The properties which are not appearing in any cluster are to be kept as binary table and added to tables list in step 3. The clusters obtained from step 2 are further checked for null values in step 6. If the cluster satisfies the null threshold and is a disjoint set then it is added to final table list else it is sent to partitioning phase.

Phase 2 Partitioning

The clusters which do not satisfy null threshold or have overlapping properties are sent to partitioning phase in sorted order of their support values. The algorithm checks each cluster for null threshold and overlapping property starting from the cluster with highest support value. The pseudo code for partitioning algorithm is given in Figure 2.8. It checks for the null threshold in step 5. The formula used for null threshold is given as

$$\boldsymbol{Null}\%(\boldsymbol{c}) = \frac{\sum_{\forall i \in c}(\boldsymbol{PU.maxcount}(\boldsymbol{c}) - \boldsymbol{PU.count}(\boldsymbol{ci}))}{(|\boldsymbol{c}| + \boldsymbol{1}) \times \boldsymbol{PU.maxcount}(\boldsymbol{c})} \qquad \textit{Eq. (2.3)}$$

The null percentage for a cluster is calculated using property usage list (PU). In *Eq. (2.3)*, $PU.maxcount(c)$ gives the maximum property usage count for cluster c.

Case 1: If the cluster satisfies the null threshold, then it looks for overlapping clusters and prunes the overlapping property from low support cluster. The pruned low support clusters are merged and the high support cluster is added to final table list.

Case 2: If the cluster does not satisfy null threshold it is partitioned. As given in step 7 of pseudo code, find the property p causing maximum null values referring to property usage list. The property is removed from the cluster. The cluster is partitioned until it satisfies null threshold. If the property p appears in low support cluster then continue iteration (step 9) else it should be added to final table list as a binary table.

```
1: Function Partition(PropClust C,PropUsage PU,Thresh_null)
2: Tables ← φ
3: for all clust_1 ∈ C do
4:   C ← (C − clust_1)
5:   if Null%(clust_1, PU) > NullThresh then
6:     repeat
7:       p ← property causing most null storage
8:       clust_1 ← (clust_1 − p)
9:       if p exists in lower-support cluster do continue
10:      else Tables ← Tables ∪ p /* Binary table */
11:    until Null%(clust_1, PU) ≤ NullThresh
12:  end if
13:  Tables ← Tables ∪ clust_1
14:  forall clust_2 ∈ C do clust_2 ← clust_2 − (clust_2 ∩ clust_1)
15:  Merge cluster fragments
16: end for
17: return Tables
```

Figure 2.8: Pseudo Code for Partitioning Algorithm [8]

At the end of partitioning phase, we get the clusters of properties that should be stored as n-ary tables and few properties that need to be kept as binary tables.

2.3 Main Memory Databases [10]

Main memory database (MMDB) systems are those in which data resides in the main physical memory permanently. On the other hand, in conventional database systems data is disk resident so they are called disk resident database (DRDB) systems.

2.3.1 Difference between DRDB and MMDB

In MMDB, the primary copy is present permanently in main memory and there is a backup copy on disk. In DRDB, data from disk is cached into memory for access.

- The access time for main memory is orders of magnitude less than for disk storage.
- Main memories were volatile and disk storage was not, so storing data permanently was not advisable. Now non-volatile memories are also available which solves the issue for volatility, still a backup copy needs to be kept on secondary storage.

- The cost of accessing data from disk is high as compared from main memory. The disk supports sequential access whereas random access of data can be done in MMDB.

The size of main memory and disk are different so can it be assumed that the entire database can fit in main memory? The answer to this question is yes for some applications. The applications which are of limited size and static in nature can fit completely. There are applications which will never fit in main memory, in that case the data need to be divided into hot and cold data where hot data is the data which is frequently accessed and low in volume whereas cold data is the rarely accessed data and voluminous in size. In such scenario, there is a possibility to partition the data and store the hottest one in main memory.

2.3.2 Impact of Memory Resident Data

In this section, we are going to discuss the impact of memory residency on some of the functional components of database management systems. Concurrency control, commit processing, access methods and recovery are some of the major components of database system.

2.3.2.1 Concurrency Control

Access to main memory is faster than disk access so transactions can complete faster. In this scenario, locks will not be held for long in the lock based concurrency control system. In disk based systems, small lock granules are chosen to reduce contention. In MMDB systems the level of interleaving will be small so instead of choosing small lock granules we can choose lock granule to be the entire database. This will perform serial execution of transactions, which in turn reduces the cost of concurrency control.

2.3.2.2 Commit Processing

A backup copy is needed to be maintained as there can be chances of media failure in the system. The log of transaction activity is required to be kept to protect against such failures. The logs must be kept in stable storage and transaction should write its activity records in the log before committing. To reduce the log bottleneck group commits can be used. Instead of performing individual commit operations for each transaction, group commits force all the transactions together on disk. Precommiting is used when stable memory is not available i.e. locks are released as soon as log records of transactions are sent to log files.

2.3.2.3 Access Methods

B-trees which are designed for block-oriented storage is not required for main memory systems. T-tree has been designed for MMDB. T-tree is a balanced index tree data structure for cases where both the index and the actual data are fully kept in memory. T-tree seeks to gain the performance benefits of in-memory tree structures such as AVL trees. For unordered data, modified linear hashing is best.

2.3.2.4 Recovery

The recovery is required to be done if a media failure affects the operations of database systems. Backups of memory resident data are always maintained on disk. The three components of recovery are: commit processing, check pointing and reload. In commit processing, log is forced to disk at commit but sometimes log I/O becomes a bottleneck. So the question arises that how long we need to keep the log, in general cases until the next check point log is maintained. Other possible solutions can be of using stable storage memory for log tails or using group commits. Precommitting can also be used in which the lock is released the as soon as the log is sent to log buffer instead of waiting for the information to reach the disk. One of the components of recovery is checkpointing which make a copy of the data on disks and then truncate the logs. In case of system failure, the system needs to restore to the previous check point. In main memory systems, the data must be restored from the backups maintained in disk.

2.4 Research Issues in MMDB

Main memory is becoming very popular these days. The architecture and algorithms used in DRDB differ from that of MMDB. Many algorithms needs to be optimized in case of main memory which includes the algorithms required in transaction management. To identify hot and cold data is another issue where hot data is the data frequently accessed and should be kept in main memory. The two main research areas in MMDB are discussed in below.

2.4.1 Data management of MMDB

There are two kinds of applications, one that can completely fit in main memory and the other which has large database size and cannot fit completely in main memory. The identification of hot and cold data is useful for both the types of applications. There are different storage techniques that can be used to manage the data in main memory which were discussed in section 2.1. It helps to improve the query performance. The problem of data management of hot and cold data is divided into two parts. Firstly, identify the hot data and cold data and secondly, store hot data in main memory and cold data in secondary storage.

2.4.1.1 Identification of Hot and Cold data for MMDB

Some applications can fit completely in main memory but more free memory can be lead to better performance. It is observed that OLTP workload exhibit skewed access patterns. The skew can be natural skew or time correlated skew. The natural skewedness works on the principle of working set model [11]. It says that in an application only some part of the data is very frequently accessed (hot data). The concept of data aging defines the time correlated skew [11]. According to data aging, recent data is accessed more frequently than older data. In the skewed access pattern, the data can be divided into two parts i.e. hot data and cold data on the basis of data aging or working set model. There is no need to store the complete database in main memory. The frequently accessed hot data can be kept in main memory and cold data in the secondary storage. This will help in optimal utilization of memory. For OLTP applications consisting frequent insert, update and delete operations, identifying hot and cold data will help in faster access.

2.4.1.2 Structuring Data in MMDB

The query execution can be made faster if data is arranged properly in main memory. The experiments for RDF data [9] show that partitioning techniques perform an order of magnitude better as compared to all other data stores (triple store). With the types of queries the partitioning techniques may also vary. As discussed in section 2.1 user oriented queries which involves no joins, execute faster in horizontal partitioning. The data can be stored in the main memory using the appropriate partitioning techniques.

2.4.2 Transaction Management in MMDB

OLTP is characterized by a large no. of short on-line transactions (insert, update and delete). The OLTP system focus on fast query processing and data can now be kept in main memory. Some optimizations need to be done in the transaction processing algorithms. The overheads of OLTP were measured and it was found that 10 % of the time is spent on doing useful work and the remaining 90% is consumed by four sources of overhead viz. buffer pool overhead, logging, locking and latching [12].

Buffer manager is used to keep data in RAM, the database access pages through buffer pool. The buffer pool stores pages currently used by the database system and if a page is not found it uses the relevant page replacement algorithm. In case of main memory database, since the data can be kept in main memory the buffer management can create an overhead and should be optimized according to it.

Locking and latching are used in multi-threaded databases, where data structures should be locked before being accessed. They are used to prevent more than one process from executing the same piece of code at a given time. The main memory has faster access so they can also perform better in single-threaded databases which neglects the use of locking and latching. The locking algorithm used in disk based system is not suitable for MMDB system. To optimally utilize the main memory databases we need light weight locking algorithms as given in [13] which aims at reducing the overheads associated with traditional lock manager operations.

The experiment shows different optimizations that can be done for main memory systems [12]. They have developed a system named SHORE (Scalable Heterogeneous Object Repository) for experimenting. The experiment was performed by removing shore components one by one. It removes system components starting with logging, followed by locking and latching and then buffer manager in the end. Shore with logging enabled and database cached in main memory provide about 640 transactions per second. Shore without log flushing and running in main memory provides about 1700 transactions per second. So it is observed that there is a factor of 20 improvement in overall throughput. The experiment also reflects that buffer manager use 34.6% of instruction count which is maximum percentage of instruction count among other system components. Optimizing all the transaction management algorithms for main memory is a research issue in the area of MMDB.

2.5 Log Based Identification of Hot and Cold Data [7]

The log based method has been developed to identify hot and cold data. This method is also targeted for OLTP workloads having a skewed access pattern. They have maintained a log file which logs each record access with its time of access. The offline analysis of log file is done to find records which are accessed most frequently. The basic concept of the algorithm is to calculate record access frequencies of all the logged records and then find the most frequent records among them. A set of four classification algorithms are used; Forward Algorithm, Backward Algorithm and parallelization of both. The algorithms scans the log and calculate record access frequency. The access frequency of records helps in identifying data into two sets which is hot set and cold set.

The record is stored in the log file as a pair of (record id, time slice). Each time a record is accessed its entry is done in log file. The record access frequency is measured using an exponential smoothing method. It performs the offline analysis of log to calculate access frequencies. The offline approach writes record access data to log for later analysis. It is used to reduce system overhead as caching techniques are expensive and logging does not impose much overhead. The analysis can also be done on a separate machine which helps in reducing the overhead on the system running the transactional workloads. The algorithms have been discussed in detail in chapter 3.

2.6 Open Research Issue: Identifying Hot and Cold Data

The faster query processing is a basic requirement for interactive web applications. OLTP databases have data size varying from few gigabytes to terabytes and they exhibit skewed access patterns. So it is a good idea to separate frequently queried data from the one which is not. In case of MMDB systems, storing only the frequently queried data can help in optimal utilization of memory. We can partition the data according to its usage or the way it is queried. The frequently accessed hot data can be kept in main memory while the cold data should be stored on disk. The log based method performs offline analysis of logs and takes the complete record as hot data. It works for static environment. It does not provide any way to handle hot data for which the access pattern may drastically change over a period of time. The identification of hot data and cold data is a research issue which can be worked upon.

This thesis provides a mechanism to identify hot and cold data using a query workload aware approach. Knowing the query set and query frequency, the hot and cold data can be identified.

Chapter 3

Identification of Hot and Cold Data

With the increased size of main memory, allows most of the databases to be stored in it. The OLTP workloads show skewed access pattern, one is due to natural skewedness and the other is due to data aging. The natural skewedness is defined by the working set model which says that only a part of data is actually accessed. The working set is the part of data very frequently accessed (also called hot data). The other part of data which is rarely or never accessed is termed as cold data. For example, in an e-commerce website only some products are popular and frequently accessed. Data aging is another factor for skewed access. It is also known as time correlated skew. In data aging, the older the data is, the less frequent it will be accessed. The example can be a courier service agency, where the details of current packages are important and useful and once the package is delivered the details of it will be rarely accessed. So it is useful if the data can be classified into hot and cold data. The purpose of identification of hot and cold data is to divide the data in such a way so that most of the queries targeted to the system can be answered by the hot data set. It can be useful in handling huge data more effectively and to optimally utilize the memory size.

Log based method has been developed to identify hot data for main memory databases [7]. The log based method logs all the records accesses and calculates their access frequencies using classification algorithms. The top K records with highest access frequencies are taken as hot data set where K is a user defined value for number of records to be taken as hot set. The value of K depends on the size of memory and dataset used. This method performs offline analysis of log so there is a need to run the classification algorithms after every fixed interval of time. It stores complete record of a table in the hot set. In our proposed approach, we will first identify the hot schema. The hot schema will be the subset of attributes from original database schema. Instead of storing the full record only hot data corresponding to hot schema is stored. The clustering algorithms like BEA, Association Rule Mining (ARM) and Hybrid Partitioning (HP) can be used to identify the hot schema. Dummy example has been worked for the algorithms in each section.

3.1 Log Based Method [7]

This section describes log based method for identifying hot and cold data for OLTP workloads. The algorithms are based on logging records and then analyzing those log files. The log files contain (record ID, Time Slice) pair for every record access. This method consists of four algorithms; Forward Algorithm, Backward Algorithm, Parallel Forward Algorithm, and Parallel Backward Algorithm. The input to these algorithms is a log file L where record accesses have been stored and a parameter k which represents the number of records to be classified as hot. The exponential smoothing equation is used for calculating record access frequency of record r at time slice t_n. The exponential smoothing method is used for time series data. It calculates the access frequency for a record r as given in *Eq. (3.1)*.

$$\boldsymbol{est_r(t_n) = \alpha * x_{t_n} + (1 - \alpha) * est_r(\,t_{n-1})} \qquad \textbf{\textit{Eq. (3.1)}}$$

In *Eq. (3.1)*, t_n represents the current time slice, x_{t_n} represents the observation value at time t_n. If an access for r is observed at t_n then the value of x_{t_n} is taken as 1 else 0.

3.1.1 Forward Algorithm

It scans the log in forward direction from the beginning time slice. Whenever a record is encountered its current access frequency is updated. The access frequency estimate is calculated using exponential smoothing. The equation is given as:

$$\boldsymbol{est_r(t_n) = \alpha + est_r(t_{prev}) * (1 - \alpha)^{(t_n - t_{prev})}} \qquad \textbf{\textit{Eq. (3.2)}}$$

where t_{prev} is the time slice at which r was last observed and α is a decay factor. When the scan is completed, the algorithm ranks the records in descending order on the basis of their access frequencies. It then selects the top k records as hot and returns only those records.

The drawback of forward algorithm is that it scans the entire log and the storage required is proportional to the number of unique record ids in the access log.

3.1.2 Backward Algorithm

It avoids scanning the log completely. It scans the log from end to beginning in reverse order and derives tighter upper and lower bounds on record access estimates.

Input: Access log file L, hot data size k

Output: top k records as hot data set.

Data Structures Used: Hash table is used for storing running estimate for each record it processes. The record ids from H are mapped to a structure containing 3 fields, *backEst* the running backward access frequency. *loEst* a record's lower-bound access frequency estimate. *upEst,* a record's upper-bound access frequency estimate.

Algorithm using dummy example

The log is scanned in reverse and the running backward estimate estb is calculated as:

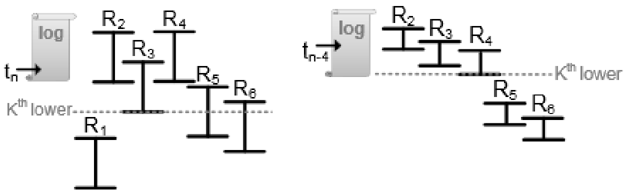
$$estb_r(t_n) = \alpha(1-\alpha)^{(t_e-t_n)} + estb_r(t_{last}) \qquad \textit{Eq. (3.1)}$$

In *Eq. (3.3)*, $estb_r(t_{last})$ is the backward estimate calculated when record r was last encountered in the log at time slice t_{last}. The backward estimate is used to compute upper bound and lower bound. The upper bound $(upEst_r(t_n))$ is the maximum access frequency record *r* can have assuming it is encountered at every record access while scanning backwards. The lower bound $(loEst_r(t_n))$ is the minimum estimate value *r* can have. The equations are given as:

$$upEst_r(t_n) = estb_r(t_n) + (1-\alpha)^{t_e-t_n+1} \qquad \textit{Eq. (3.4)}$$

$$loEst_r(t_n) = estb_r(t_n) + (1-\alpha)^{t_e-t_b+1} \qquad \textit{Eq. (3.5)}$$

In *Eq. (3.4)*, t_e gives the end time slice in the log. The Backward algorithm scans the log and fills hash table H with initial k records. The upper bound, lower bound, and backward estimate are stored in structure. The lower bound of these k records is used to find kth lower bound. The records having upper bound less than kth lower bound are discarded as they can never be a part of the hot data set. The kth lower bound defines the accept threshold (*acceptThresh*). The Figure 3.1 gives an example for backward classification.



(a) At time slice t_n (b) At time slice t_{n-4}

Figure 3.1: Backward Classification Example [7]

Figure 3.1 (a) represents the state of six records R_1 to R_6 at time slice t_n. It is assumed that k = 3. The records having upper bound less than kth lower bound are dropped so record R1 can be removed as it cannot be part of the hot set. After moving back to four time slice represented in Figure 3.1 (b), one can observe only three records in contention for hot set. Since the value of k was assumed 3 so the algorithm stops at this point. The top k hot records which are obtained are R_2, R_3, and R_4.

The advantage of backward algorithm is that we don't need to scan the complete log. The storage space required is equivalent to number of hot records instead of all the records. The parallel version of both the classification algorithms is available which speed up the estimation time.

3.2 Query Aware Approach

In Query Aware approach, it is assumed that query set and the corresponding frequencies are known. The objective of this approach is to identify hot and cold schema and data. There are methods like QAA-BEA and query aware approach using association rule mining (QAA-ARM) which can be used for identification of hot and cold data. The thesis work proposes query aware approach using hybrid partitioning (QAA-HP). It looks at the problem of identification of hot and cold data in two phases, one is identifying hot schema and the other is to identify the corresponding hot data. It involves a combination of vertical partitioning and horizontal partitioning which is the reason it is called hybrid partitioning.

3.2.1 QAA-BEA

The BEA algorithm was discussed in section 2.2.1. The clustering algorithm can be used to identify clusters of hot attributes. The algorithm has been explained with the help of dummy example.
Suppose there is query set { Q1, Q2, Q3, Q4 }. The attribute set for each query and the respective query frequencies are given in table below:

QID	**Attribute Set**	**Frequency**
Q1	A1, A2, A3	20
Q2	A3, A4	25
Q3	A2, A4, A5	90
Q4	A3, A4	10

Step 1: Construct the Attribute Usage Matrix (AUM)

	A1	A2	A3	A4	A5
Q1	1	1	1	0	0
Q2	0	0	1	1	0
Q3	0	1	0	1	1
Q4	0	0	1	0	1

Step 2: Construct Attribute Affinity Matrix (AAM) for given example

	A1	A2	A3	A4	A5
A1	20	20	20	0	0
A2	20	110	20	90	90
A3	20	20	55	25	10
A4	0	90	25	115	90
A5	0	90	10	90	100

Step 3 Clustering: Using Bond Energy Algorithm (BEA), the clustered affinity matrix (CAM) is built. Let us initialize the CAM with attribute A1 and A2. The attributes are placed according to the contribution global affinity measure (AM). The AM can also be estimated using bond formula [17].

$$\boldsymbol{bond}(\boldsymbol{A_x},\boldsymbol{A_y}) = \sum_{\boldsymbol{z=1}}^{\boldsymbol{n}} \boldsymbol{aff}(\boldsymbol{A_z},\boldsymbol{A_x})\boldsymbol{aff}(\boldsymbol{A_z},\boldsymbol{A_y}) \qquad \textit{\textbf{Eq. 3.6}}$$

The AM can be given as

$$\boldsymbol{AM} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} [\boldsymbol{bond}(\boldsymbol{A_j},\boldsymbol{A_{j-1}}) + \boldsymbol{bond}(\boldsymbol{A_j},\boldsymbol{A_{j+1}})] \qquad \textit{\textbf{Eq. 3.7}}$$

The reordering of attributes in affinity matrix is done on the basis of maximum contribution value. To place attribute A_k between attributes A_i and A_j the net contribution [17] is calculated as

$$\begin{aligned}\boldsymbol{Cont}\ (\boldsymbol{A_i},\boldsymbol{A_k},\boldsymbol{A_j}) \\ = \boldsymbol{2}\ \boldsymbol{bond}\ (\boldsymbol{A_i},\boldsymbol{A_k}) + \boldsymbol{2}\ \boldsymbol{bond}\ (\boldsymbol{A_k},\boldsymbol{A_j}) - \boldsymbol{2}\ \boldsymbol{bond}\ (\boldsymbol{A_i},\boldsymbol{A_j})\end{aligned} \qquad \textit{\textbf{Eq. 3.8}}$$

In the example, A1 and A2 are placed in CAM and check for A3, the contribution will be calculated for combinations (1-2-3) and (1-3-2). The combination giving maximum contribution value is considered for ordering.

For attribute A3:

$cont(1-3-2) = \ 2\ bond(1,3) + \ 2\ bond(3,2) - \ 2\ bond(1,2)$
$bond(1,3)$ = (20 * 20) + (20 * 20) + (20 * 55) + (0 * 25) + (0 * 10) = 1180
$bond(3,2)$ = (20 * 20) + (20 * 110) + (55 * 20) + (25 * 90) + (90 * 10) = 6850
$bond(1,2)$ = (20 * 20) + (20 * 110) + (20 * 20) + (0 * 90) + (0 * 90) = 3000
Therefore, $cont(1-3-2)$ = 2 (1180 + 6850 – 3000) = 10,060
Similarly, $cont(1-2-3)$ = 17,340
The attributes are ordered as {A1, A2, A3}

For attribute A4:
To place attribute A4, the contribution of (1-4-2), (1-2-4), (2-3-4) and (2-4-3) is calculated. The contribution is calculated same as given for attribute A3.

$cont(1-4-2) = 2\,bond(1{,}4) + 2\,bond(4{,}2) - 2\,bond(1{,}2)$ = 56,300
$cont(1-2-4)$ = - 47,100
$cont(2-3-4)$ = - 30,100
$cont(2-4-3)$ = 57,900
Since the contribution for attribute set {A2, A4, A3} is maximum; A4 will be placed between A2 and A3. The ordering we get after taking A4 is {A1, A2, A4, A3}

For attribute A5:
To place attribute A5, the contribution of (1-5-2), (2-5-4), (4-5-3) and (4-3-5).

$cont(1-5-2)$ = 39,800
$cont(2-5-4)$ = 39,500
$cont(4-5-3)$ = 52,700
$cont(4-3-5)$ = - 30,300
The final ordering of the attributes is obtained as {A1, A2, A4, A5, A3}. The clustered affinity matrix (CAM) formed after the permuting all the rows and columns is given below

	A1	A2	A4	A5	A3
A1	20	20	0	0	20
A2	20	110	90	90	20
A4	0	90	115	90	25
A5	0	90	90	100	10
A3	20	20	25	10	55

Step 4 Partitioning: find the split point P maximizing value of Z given in *Eq. 2.2*. The point is taken across the diagonal, to divide the CAM into two fragments.
Split at {A1 | A2, A4, A5, A3}:
Number of queries accessing only upper fragment (QU) = 0
Number of queries accessing only lower fragment (QL) = 125
Number of queries need to access both the fragments (QI) = 20
Z = (0 * 125) – (20) ^2 = (-) 400
Split at {A1, A2 | A4, A5, A3}: the value of Z is estimated to be (-) 12100
Split at {A1, A2, A4 |A5, A3}: the value of Z is estimated to be (-) 18,225
Split at {A1, A2, A4, A5 | A3}: the value of Z is estimated to be (-) 3,025

The value of Z is maximum at {A1 | A2, A4, A5, A3}. The two fragments obtained are {A1} and {A2, A4, A5, A3}.
The partitioning (2-way) phase takes O (N) time in terms of number of attributes. If m-way partitioning is used instead of 2-way than the time complexity rises to O (2^m). This is so because in m-way partitioning it is necessary to try 1, 2… m-1 split points and for each of these, it is necessary to check which place maximizes 'z'.

3.2.2 Query Aware Approach using Association Rule Mining (QAA-ARM)

The algorithm has been proposed to identify hot schema and cold schema which can be further mapped to hot data and cold data respectively. Given a query set and their frequencies we need to find out the attribute clusters. The QAA-ARM algorithm takes as input the query set and their frequency. It consists of two phases, clustering phase and partitioning phase. The clustering phase provides the set of hot and cold schema and the partitioning phase takes care of the possible overlapping in the hot schema.

3.2.2.1 Data Structures

The algorithm uses *Support value (SV)* it is the frequency of occurrence of an attribute. *Support threshold* is the minimum support value allowed. *Confidence value (CV)* between attribute X and Y determines how frequently attribute Y appear in query that contain X [14]. *Null threshold* it is the percentage of null values allowed in a cluster.

This algorithm needs four data structures, *Query Attribute Basket*: Each query is mapped to its associated attributes. It is represented as Query ID → {A1, A2,…An}. *Query Frequency list*: it maps queries with their occurrence frequency count. *Attribute Frequency List (AF List):* Prepared using query attribute basket and query frequency list. It keeps the frequency count of each attribute. *Query Frequency Attribute List (QFA basket):* store the query frequency (QF) and the corresponding attribute set together. Denoted as QF → {A1,A2…}

3.2.2.2 Algorithms

This algorithm consists of two phases, clustering phase followed by partitioning phase. The clustering phase generates clusters of attributes that can be stored together as n-ary tables. The attributes which are mostly accessed together are candidates to appear in same cluster. In case the clusters have overlapping attributes, partitioning needs to be done.

Phase 1 Clustering:
Input: Query Frequency Attribute List (QFA basket), Attribute Frequency List (AF list)

Output: Clusters of attributes (Cluster List) and set of attributes which will be part of cold schema (Cold Schema List)

L_k = Set of large k-attribute set, C_k = candidate set of size k containing Attribute-set and its SV, C-1 = QFA basket, K = 2

Step 1: Initialize L_{K-1} with AF list and Final Table List as empty.

Step 2: Calculate min_supp (minimum support) using function min_sup (L_{K-1})

- Sort L_{K-1} in descending order of support value.
- Take median and put min_supp = median support value.

Step 3: Check if SV > min_sup

- if yes than keep it in L_{K-1}
- Else remove attribute entry from L_{K-1} and check for confidence value (CV) where CV (XY) = Support (XY)/Support(X). Here minimum confidence value (min_conf) is assumed to be 0.5.
- If CV > 0.5 then send the attributes set in Cluster List else send it to Cold Schema List.

Step 4: Generate C_K using Apriori_Gen (L_{K-1}) [14]

Apriori_Gen (L_{K-1}) works as given below:

- For k=1, C1 = all 1-itemsets.
- For k > 1, generate Ck from Lk-1 as follows:

  Step 4.1: join step
  C_K = k-2 way join of L_{K-1} with itself
- If both $\{A_1, \ldots, A_{k-2}, A_{k-1}\}$ & $\{A_1, \ldots, A_{k-2}, A_k\}$ are in L_{K-1}, then add $\{A_1, \ldots, A_{K-2}, A_{K-1}, A_K\}$ to C_K
- Step 4.2 After join we have a Prune Step which remove $\{A_1, \ldots, A_{K-2}, A_{K-1}, A_K\}$ if it contains a (k-1) subset which is not present in L_{K-1}

Step 5: Calculate support value for candidates in C_K by iterating over C-1. Repeat step 2 to 5 until C_K turns out to be empty.

Step 6: Store attribute sets from L_{K-1} in the cluster list.

Step 7: Keep disjoint clusters in Final Table List and clusters with overlapping attributes are sent to partitioning phase.

Phase 2 Partitioning:

This is used for clusters having overlapping attributes. Input to this phase is a Cluster List obtained from phase 1.

Step 1: Arrange clusters in descending order of SV.

Step 2: Take cluster 1 with maximum support value. Check for null threshold (we are assuming 10% of the size of the table as null threshold). If it satisfies the null threshold then prune the overlapping attribute from lower support cluster and store cluster 1 in final table list. Else check for each attribute which create

maximum null values and remove that attribute from the cluster. The attribute if not part of any other cluster than store it as binary table.
Step 3: Repeat step 2 for all clusters and update the final table list.
Step 4: The attribute clusters obtained is our hot schema and will be stored as n-ary tables or binary table form.

The algorithm QAA-ARM runs till the candidate set C_k turns out to be empty. Let the number of iterations be *j*. Suppose there are total M number of queries, N number of unique attributes. Let the large k-attribute set (L_k) be sorted in each of the *j* iterations. The time taken to sort the list can be given as O (N logN) in worst case. The candidate set formation takes O (N^2) time in each iteration and to calculate the support value it scans the M entries in the QFA basket. Thus the computational complexity becomes high for the QAA-ARM algorithm, estimated to be O ($N^2 * M$).

3.2.2.3 Dummy Example

The algorithm is explained here with the help of dummy example of the library database. Our assumption is that we know the query set and frequency for each query.
Library database has the following schema:
Book (<u>BookID</u>, Bname, AuthorID), ***Author*** (<u>AuthorID</u>, Aname), ***Student*** (<u>SID</u>, Sname), ***BookIssue*** (<u>BookID</u>, <u>SID</u>)
The query set consist of following three queries
Q1 List books authored by 'XYZ'
Q2 No. of books issued by a particular student
Q3 Print name of students who have issued a book authored by 'XYZ'
Following four data Structures have been defined:
Query-Attribute Basket

Q1 -> { BookID, Bname, AuthorID, Aname }
Q2 -> { SID, Sname, BookID }
Q3 -> { Sname, SID, BookID, AuthorID, Aname}

Query Frequency List

QID	Frequency
Q1	180
Q2	150
Q3	50

Attribute Frequency List (AF List)

Attribute	Frequency
BookID	350
Bname	180
AuthorID	200
Aname	200
SID	200
Sname	200

Phase 1: Clustering

Considering the QFA basket

C-1

QF	Set-of-attributes
180	{ { BookID}, {Bname}, {AuthorID}, {Aname} }
150	{ {Sid}, {Sname}, {BookID} }
50	{ {Sname}, {SID}, {BookID}, {AuthorID}, {Aname} }

L_1

Attributes	Support
BookID	350
AuthorID	200
Aname	200
SID	200
Sname	200

Iteration 1: using L_1 find attribute-set of C_2 and calculate SV using C-1

Attribute-Set	Support
{BookID, AuthorID}	230
{BookID, Aname}	230
{BookID, SID}	200
{BookID, Sname}	200
{AuthorID, Aname}	230
{AuthorID, SID}	50
{AuthorID, Sname}	50
{Aname, Sid}	50
{Aname, Sname}	50
{SID, Sname}	200

Iteration 2: Sort the frequencies in descending order and take its median. Here it is 250

Attribute-Set	Support
{BookID, AuthorID}	230
{BookID, Aname}	230
{AuthorID, Aname}	230
{BookID, Sid}	200
{BookID, Sname}	200
{SID, Sname}	200
{AuthorID, Sname}	50
{Aname, Sid}	50
{Aname, Sname}	50
{AuthorID, SID}	50

L_2

Attribute-Set	Support
{BookID, AuthorID}	230
{BookID, Aname}	230
{AuthorID, Aname}	230
{BookID, SID}	200
{BookID, Sname}	200
{SID, Sname}	200

Calculate confidence value for {AuthorID, SID}, { AuthorID, Sname}, {Aname, Sname}, {Aname, Sname}.

CV (AuthorID, Sname} = Support(AuthorID, Sname)/ Support(AuthorID)

= 50/200 = 0.25 < 0.5

So this set is discarded and sent to cold schema list. Similarly, calculate CV for remaining attribute set, here all the remaining attributes sets have confidence value less than the min_conf which is 0.5.

Generate C3 from L2

Attribute-Set	Support
{ BookID, AuthorID, Aname }	230
{ BookID, SID, Sname }	200

L_3

Attribute-Set	Support
{BookID, AuthorID, Aname}	230

Check CV of attribute set {BookID, SID, Sname} = 200/200 = 1. So send cluster {BookID, SID, Sname} to Cluster_list.

Since C4 is generated empty. The final cluster we get is L3 {BookID, AuthorID, Aname}.

The Cluster List now contains 3 clusters of attributes as shown below:

Clusters	Support
{BookID, AuthorID, Aname}	230
{BookID, SID, Sname}	200
{Bname}	180

Since cluster 1 and 2 contain overlapping properties so they are not final clusters and sent to partitioning phase but Bname is part of final table list.

3.2.2.4 Challenges in the Algorithm

- To set the null threshold. Null threshold is the percentage of null values allowed in a relation. If a cluster has number of null values exceeding the null threshold that clusters need to be partitioned. So initially it is assumed to be 10% of the total values in the relation.
- To set the support threshold for making clusters. Support threshold is calculated to check the affinity of attributes i.e. how often they are used by a query. It is assumed to be 80% of the median value from all sorted support counts. If support value of an attribute is less than support threshold than it is not in contention for hot set.
- To set Confidence value (CV). Confidence value of X and Y denote that how frequently attribute Y appear in queries which also use attribute X. It is calculated using the support values of X, Y. Confidence value should be more than minimum confidence (min_conf) value. For experimental purpose, min_conf is assumed to be 0.5.

3.2.3 Query Aware Approach using Hybrid Partitioning (QAA-HP)

The QAA-HP can be used for identification of hot and cold data. The algorithm works on the assumptions that query set and the frequency for each query is known. It calculates the attribute frequencies occurring in the query set. With the help of attribute frequency, we try to find hot and cold schema. The hot schema is the subset of original schema given for a database. To find out schema subset which is hot can be achieved from vertical partitioning. The vertical partitioning means we divide the relational tables into subsets. On the basis of attributes usage, the attributes which are frequently accessed are clustered together. The OLTP workload queries contain rare joins, so this approach considers all the queries without joins. The attribute clusters formed are a subset of the original schema as no join query is there. The maximum number of clusters obtained from the algorithm will be equivalent to number of relations in the given OLTP database. Once the hot schema is identified, we find the hot data corresponding to the hot schema. The hot data can be obtained with the help of horizontal partitioning. The concept of horizontal partitioning was discussed in section 2.1. It partitions a single table into multiple tables but with a fewer number of rows in it as compared to original table. In the identification of hot and cold data, if the hot schema is identified we need to find data corresponding to only those attributes and data that are accessed frequently by the queries. The algorithm use vertical partitioning for schema identification. When it comes to data identification it uses horizontal partitioning. Since we are using a combination of vertical and horizontal partitioning this approach has been named as query aware approach using hybrid partitioning.

3.2.3.1 Data Structures

The algorithm needs three data structures. *Query Frequency list*: it maps queries with their occurrence frequency count. *Query Frequency Attribute List (QFA basket):* store the query frequency (QF) and the corresponding attribute set together. Denoted as QF → {A1,A2…}. *Attribute Frequency List (AF List):* Prepared using query frequency attribute basket. It keeps the frequency count of each attribute. It can be represented as $A_j \rightarrow f_j$, where j denotes j^{th} attribute.
The algorithm uses *frequency threshold,* it is the minimum frequency count for an attribute to be a part of the hot schema.

3.2.3.2 Algorithms

The algorithm uses hybrid partitioning i.e. vertical partitioning and horizontal partitioning. Firstly it identifies hot and cold schema using vertical partitioning. The corresponding hot and cold data can be identified using horizontal partitioning. If there are *r* number of tables in a database we can get maximum *r* clusters from the algorithm. The two algorithms used are *QAA-HP1* and *QAA-HP2*. The QAA-HP1 returns the hot schema in the form of cluster list. Algorithm QAA-HP2 returns cluster tables corresponding to hot schema with relevant data. The pseudo code for algorithms QAA-HP1 and QAA-HP2 are given in Algorithm 1 and Algorithm 2, respectively.

The Algorithm 1 QAA-HP1 takes as input AF List and returns clusters of attributes which are called hot schema. The AF List consists of all attributes which are part of the query set with their respective frequencies. Considering all the attributes in AF List, find the mean frequency value of the frequency counts. We have set the frequency threshold as 80% of the mean frequency value as given in step 5. Step 6 checks the frequency count for each attribute and attributes having frequency more than the frequency threshold are considered to be the part of hot schema. The attributes are added in the clusters according to the database relation they belong to. The clusters for all the *r* tables in the database are defined. The attribute is added to the corresponding table cluster. The algorithm returns the non-empty clusters in the end. After identifying the hot schema, the hot data corresponding to it is to be identified.

Algorithm 1: QAA-HP1 (AF list)

1. Set total_frequency = 0, attribute_count = 0
2. For all A_i in AF List **do**
a. total_frequency = total_frequency + f_{A_i}
b. attribute_count ++;
3. end for
4. mean frequency = total_frequency / attribute_count
5. frequency threshold = mean * 0.8
6. for all A_i in AF List **do**
a. if f_{A_i} > Frequency Threshold **then**
For each table T_r **do**
If A_i exist in T_r **then** add A_i to Hot Schema of T_r
End For
b. else
Add A_i to cold schema list
7. End For
8. Return Clusters

The Algorithm 2 QAA-HP2 is used for identifying hot data. In step 1 it creates temporary tables and stores the query results in them. It creates tables for all the clusters obtained from algorithm QAA-HP1 in step 2.a. The cluster tables will have only a subset of attributes from the original database table. The attributes type will be same in cluster table as it is given in the corresponding original database table. The cluster tables formed using hot schema is vertical partitioning of the original tables. The cluster tables are populated with distinct records coming from temporary tables. The data obtained in cluster table is our hot data.

Algorithm 2: QAA-HP2 (Cluster list C)
1. Store all query results in temporary tables **2. For** all C_i in cluster list **do** a. Create table C_i b. Select and store distinct tuples from temporary tables **3. End For**

3.2.3.3 Dummy Example

The Algorithm 1 QAA-HP1 is explained here with the help of a dummy example. Our assumption is that we know the query set and the frequency for each query. Let us consider the example of a library database. The schema is given as:
Book (<u>BookID</u>, Bname, B_AuthorID, Publisher), ***Author*** (<u>AuthorID</u>, Aname), ***Student*** (<u>SID</u>, Sname, S_BookID, S_Stream, S_Year)

The query set consists of following four queries
Q1 Give names of students who have issued any book
Q2 Give author id of the book 'xyz'
Q3 What is the book id of book 'abc'
Q4 Give author id of author 'pqr'

The data structures are given as

Query Frequency List

QID	Frequency
Q1	200
Q2	125
Q3	50
Q4	20

Query Frequency Attribute Basket

QF	Set-of-Attributes
200	{SID, Sname, S_BookID}
125	{BookID, Bname, B_AuthorID}
50	{BookID, Bname}
20	{AuthorID, Aname}

Attribute Frequency List (AF List)

Attribute	Frequency
SID	200
Sname	200
S_BookID	200
BookID	175
B_AuthorID	125
Bname	175
AuthorID	20
Aname	20

The QAA-HP algorithm calculates mean frequency value, here mean frequency value is 139 (average of frequency counts given in AF List). The frequency threshold is estimated to be 112 (80% of mean frequency). All the attributes having frequency count more than frequency threshold are considered as hot attributes. The hot attributes obtained in the example are {SID, Sname, S_BookID, BookID, Bname, B_AuthorID}. The algorithm works for queries for no joins. The attributes of same table are kept in the respective hot schema of their tables. The hot schema obtained for the example is
HotSchema_Student (SID, Sname, S_BookID)
HotSchema_Books (BookID, Bname, B_AuthorID)

3.2.3.4 Time Complexity of QAA-HP Algorithm

The QAA-HP algorithm requires data structures QFA Basket and AF List. Let there be total *M* queries and *N* attributes.
To prepare QFA Basket, we need to scan the query list and query frequency list for each query. There are total *M* queries, so the time required to prepare QFA Basket is O (M).

To prepare AF List, we need to scan the QFA Basket. It is given that there are total *N* unique attributes. To calculate frequency of each attribute, we need to scan the QFA Basket M times. The time taken to construct AF List is estimated to be O(MxN).

In the algorithm QAA-HP1, the calculation of total frequency will require O (N) time. Similarly checking for hot attributes and making the hot schema will require O (N) time.

3.2.3.5 Limitations of the Algorithm

The frequencies have been assigned on general understanding of the data. To set the frequency threshold is one of the limitations. At the moment it has been taken as 80% of the mean frequency value of total frequency count but it needs to be addressed in the future. In order to get to the query frequency number, log based method may be required. QAA-HP algorithm considers read-only static data. It will work for queries having no joins.

Chapter 4

Research Methodology and Experimental Setup

The research methodology followed for implementation of QAA-HP is discussed in this chapter. It also gives details about the experimental setup, the benchmark dataset, and the query set. Metrics have been devised to analyze the technique. In section 3.2.2 the algorithms of QAA-HP has been discussed. QAA-HP is used for identification of hot and cold schema as well as data. To find out schema subset which is hot, can be achieved by vertical partitioning the original schema. When it comes to hot data, the algorithm uses horizontal partitioning
It uses both vertical and horizontal partitioning so it is termed as hybrid partitioning approach. The QAA-HP works under certain constraints and assumptions. The approach is currently applicable for static environment. We have assumed the data to be read only and the queries have been designed based on our understanding of the dataset. It is observed that the OLTP queries contain either no joins or very few joins. In our approach the query set designed is for queries without joins. The query frequencies have been assigned manually using information of the dataset available in our experiment. We may require to analyze the log files to estimate exact query frequencies for all the queries.

4.1 Research Methodology

The QAA-HP technique divides the problem of identification of hot and cold data into two parts. The first part is to identify the hot schema for our database on the basis of query workload. The algorithm QAA-HP1 is used to find the hot schema. To identify the data for corresponding hot schema, it use algorithm QAA-HP2. The pseudo codes for both the algorithms are given in section 3.2.3. The research methodology for QAA-HP has been represented using a flow diagram given in Figure 4.1.

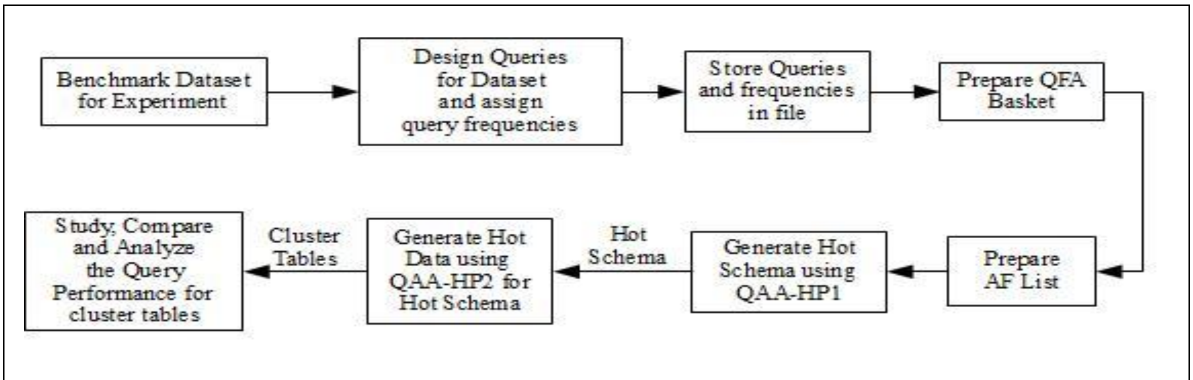


Figure 4.1: Research Methodology for QAA-HP

Step-wise procedure to implement QAA-HP represented in Figure 4.1 is described below:

- The queries are designed for available benchmark dataset and the frequencies are assigned manually. The queries with their query id and frequency count are stored in a text file. Let us assume there are 'M' number of queries and 'N' attributes used in the queries.
- *Prepare QFA basket* The QFA basket is prepared using queries and their frequency count. It scans the queries written in file and for each query its frequency is checked. Each entry of the QFA basket will contain query frequency and the attribute set used in the respective query.
- *Prepare AF List* the attribute frequency list is prepared using QFA basket. It scans all the entries of QFA basket every time for each unique attribute. The frequency of the query containing that attribute is added to attribute frequency count.
- The hot clusters are generated using algorithms QAA-HP1 and QAA-HP2 which are discussed in section 3.2.3.2.

The clusters performance will be analyzed on the basis of performance parameters devised for our algorithm.

4.2 Performance Parameters

The hot clusters are used to store hot data, so the size of clusters will be less than original database tables. After the identification of hot schema and hot data, we will run all the queries for hot clusters and original tables. The different performance parameters are devised for analyzing the query load and the query execution time on original data and hot clusters data. The analysis is done for three different cases of query types.

Case 1 All Queries: In this case we are considering all the queries given for a dataset. The queries are arranged in descending order of their frequency counts (f_i), where i is the query id. Given a set of N queries, run all the queries on hot clusters and observe the number of queries that can be answered by hot clusters.

*Case 2 Hot Queries***:** instead of taking all the queries consider only the hot or frequent queries. The hot queries are selected based on the frequency count (f_i) of the query. Given a query set arrange the queries in descending order of their frequencies. Take the median value and record the corresponding query frequency count. Take all the queries which have frequency more than the median frequency count. So almost top 50% of the queries are considered to be hot.

Case 3 Hottest Queries: In this case consider only the most frequent queries. The top 10% of total queries has been considered the hottest queries. Given a query set arrange the queries in descending order of their frequencies. Consider only top 10% of the queries from the sorted list of queries.

4.2.1 Query Analysis

Using QAA-HP we can identify the hot and cold data. The hot data is stored in hot clusters. We need to analyze the clusters on the basis of queries. This analysis is needed as we are dividing the data into two sets. The hot data is to be stored in hot clusters. We have identified two factors, Query Load QL and Query Workload QW, which will help in analyzing the cluster performance on the basis of queries being answered by them from the given query set.

4.2.1.1 Query Load QL

The queries are executed on hot clusters. The Query Load analysis is done on the basis of number of queries that can run on clusters. Let the total number of queries be $\boldsymbol{N}$. We need to check for the percentage of number of queries that can be answered by hot clusters. The query load analysis for the number of queries that can be answered by clusters is denoted as $(QL)_{QueryType}$ where the values of $QueryType$ is varying for all the three cases we have considered.

Case 1 All Queries: Given a set of $\boldsymbol{N}$ queries, run all the queries on hot clusters and observe the number of queries that can be answered by hot clusters. Let the clusters can answer $\boldsymbol{M}$ queries. So the percentage of queries that can be answered by hot clusters is given as:

$$(\boldsymbol{QL})_{\boldsymbol{AllQuery}} = \left(\frac{\boldsymbol{M}}{\boldsymbol{N}}\right) \times \mathbf{100} \qquad \textit{Eq. (4.1)}$$

Case 2 Hot Queries: If $\boldsymbol{H}$ queries are taken as hot and it is observed that out of , $\boldsymbol{Q}$ queries can run on clusters, than the query load analysis will be given as:

$$(\boldsymbol{QL})_{\boldsymbol{HotQueries}} = \left(\frac{\boldsymbol{Q}}{\boldsymbol{H}}\right) \times \mathbf{100} \qquad \textit{Eq. (4.2)}$$

Case 3 Hottest Queries: This analysis is done to check the percentage of hottest queries running on clusters compared to the total number of hottest queries. Let total number of hottest queries be $\boldsymbol{P}$ and $\boldsymbol{R}$ be the number of hottest queries that can be answered by clusters. The query load analysis is given as:

$$(\boldsymbol{QL})_{\boldsymbol{HottestQueries}} = \left(\frac{\boldsymbol{R}}{\boldsymbol{P}}\right) \times \mathbf{100} \qquad \textit{Eq. (4.3)}$$

4.2.1.2 Query Workload $\boldsymbol{QW}$

The Query Workload analysis is done for the amount of workload that can be answered by the clusters. The QL gives the percentage of number of queries that can be answered by clusters. In a system, different queries are having different occurrence frequencies$f$. The overall workload of the system is based on the summation of frequencies of all the queries targeted for the system.

Case 1 All Queries: For all the queries running on clusters, we need to calculate the percentage of amount of workload answered by them as given in *Eq. 4.4.* The value of i varies for $\boldsymbol{M}$ queries

$$(\boldsymbol{QW})_{\boldsymbol{AllQueries}} = \mathbf{100} \times \frac{\sum_{\forall \boldsymbol{M}} \boldsymbol{f_i}}{\sum_{\forall \boldsymbol{N}} \boldsymbol{f_i}} \qquad \textit{Eq. (4.4)}$$

Case 2 Hot Queries: Considering only the hot queries, the percentage of workload answered by hot queries is given by:

$$(\boldsymbol{QW})_{\boldsymbol{HotQueries}} = \mathbf{100} \times \frac{\sum_{\forall \boldsymbol{Q}} \boldsymbol{f_i}}{\sum_{\forall \boldsymbol{H}} \boldsymbol{f_i}} \qquad \textit{Eq. (4.5)}$$

Case 3 Hottest Queries: The percentage of workload answered by the hottest queries out of total hottest queries is given as

$$(\boldsymbol{QW})_{\boldsymbol{HottestQueries}} = \mathbf{100} \times \frac{\sum_{\forall \boldsymbol{R}} \boldsymbol{f_i}}{\sum_{\forall \boldsymbol{P}} \boldsymbol{f_i}} \qquad \textit{Eq. (4.6)}$$

So the workload analysis will give the idea of the quantity of workload that can be handled by the clusters.

4.2.2 Query Execution Time *QET*

The queries are executed on hot clusters and original data. It is observed that query execution time QET on hot clusters ($(QET)_{HotCluster}$) is less than the time taken to execute on original tables ($(QET)_{original}$), termed as time gain for hot clusters TG. The time gain factor can be analyzed to check improvement in accessing data over clusters as compared to the original tables. We are devising two parameters for the analysis of QET, Time gain TG and total time saved TTS, where time saved gives the amount of time saved w.r.t. the workload of queries.

4.2.2.1 Time Gain $\boldsymbol{TG}$

The time gain will be the difference between ($(QET)_{original}$) and ($(QET)_{HotCluster}$). To calculate time gain for i^{th} query, the QET for i^{th} query for original table will be denoted as $((\boldsymbol{QET})_i)_{\boldsymbol{original}}$ and the QET for i^{th} query for cluster table will be given as $((\boldsymbol{QET})_i)_{\boldsymbol{HotCluster}}$.

The gain in time for i^{th} query is given as:

$$\boldsymbol{TG} = ((\boldsymbol{QET})_i)_{\boldsymbol{original}} - ((\boldsymbol{QET})_i)_{\boldsymbol{HotCluster}} \qquad \textit{Eq. (4.7)}$$

The time gain in terms of percentage $TG\%$ can be calculates as given in *Eq. (4.8)*

$$\boldsymbol{TG}(\%) = \frac{((\boldsymbol{QET})_i)_{\boldsymbol{original}} - ((\boldsymbol{QET})_i)_{\boldsymbol{HotCluster}}}{((\boldsymbol{QET})_i)_{\boldsymbol{original}}} \times \mathbf{100} \qquad \textit{Eq. (4.8)}$$

4.2.2.2 Total Time Saved ($\boldsymbol{TTS}$)

The time gain for each query can be used to estimate the overall time saved by queries running on hot clusters. Here also we are considering three different cases for query runs which are discussed above.

Case 1 *All Queries*: Given initial set of $\boldsymbol{N}$ queries, run all the N queries and observe the number of queries which can run on hot clusters. It is expected that only a subset of the query set can run on hot clusters, gain for such hot cluster queries can be calculated using *Eq. (4.7)*. Suppose we get $\boldsymbol{M}$ queries out of total queries that can be executed on hot clusters. The $(QET)_{original}$ and $(QET)_{HotCluster}$) for these M queries is observed and the time gain for each query can be calculated using *Eq. (4.7)*. The total time saved (TTS) by running queries on hot clusters can be given by

$$(\boldsymbol{TTS})_{\boldsymbol{no.of\ hot\ cluster\ queries}} = \sum_{\boldsymbol{QueriesOnHotCluster}} (\boldsymbol{TG} * \boldsymbol{f_i}) \qquad \textit{Eq. (4.9)}$$

The value of i varies M times, where M is the number of queries which can run on hot clusters.

Case 2 *Hot Queries***:** instead of taking all the queries considering only the frequent queries. The frequent queries are selected based on the frequency count (f_i) of a query. The steps are given below:

1. Run only the hot queries on hot clusters and calculate the QET for each query on original data as well as hot cluster table.
2. Calculate time gain for each query that runs on hot clusters using *Eq. (4.7)* and in the end calculate the total time saved for these queries using *Eq. (4.9)*.

Case 3 *Hottest Queries***:** In this case consider only the most frequent queries. The top 10% of total queries has been considered the most frequent queries here.

1. Run only the hottest queries on hot clusters and calculate the QET for each query on original data as well as hot clusters.

2. Calculate time gain for each query that runs on hot clusters using *Eq. (4.7)* and in the end calculate the total time saved for these queries using *Eq. (4.9)*.

4.3 Experimental Setup

The algorithm has been implemented for disk based systems. Experiments are carried out to construct clusters for hot data and execute queries on it. Eclipse 4.4.1, Java 1.7 and postgres 9.3.6 are used as software tools in system with 8 GB RAM and Ubuntu 14.04 operating system.

OS	Ubuntu 14.04
Hard Disk Size	500 GB
RAM Size	8 GB
Database Tools	Postgresql 9.3.6
Software Tools	Eclipse IDE 4.4.1, Java 1.7

Table 4.1: Experimental Setup

The details of experimental setup required are summarized in Table 4.1. The data structures used and the algorithm to generate hot schema have been implemented in java.

4.4 Dataset

The experiment is using TPC-C benchmark dataset. This benchmark has been designed by TPC (Transaction Processing Performance Council) [15]. TPC-C is an OLTP workload. It is centered on the activity of processing orders and provides a logical database design. The TPC-C benchmark represents the activity of industry which manages, sells, or distributes a product or service [16]. It is designed for wholesale supplier with a number of geographically distributed sales districts and associated warehouses. The details of TPC-C benchmark are given in Table 4.2.

- 10 districts are covered by each regional warehouse.
- Each district is serving 3,000 customers. So the customer table will have total W*30,000 entries where W is number of warehouses.
- History is maintained for transactions made by each customer. So it has W*30k entries.
- All warehouses are maintaining stocks for the 100,000 items sold.

- The stock level for each item in each warehouse is maintained by stock table.
- Order table is permanent record of each order whereas in order_line table an entry is made for each item ordered.
- There are on an average 10 items per order.

Relation Name	No. of Rows
Item	100,000 rows
Warehouse	1 row for each warehouse
District	10 rows for each warehouse
Customer	30,000 rows for each warehouse
History	30,000 rows for each warehouse
Order	30,000 rows for each warehouse
Order_line	A mean value of 300,000 rows for each warehouse
New_order	9,000 rows for each warehouse
Stock	100,000 rows for each warehouse

Table 4.2: Overview of TPC-C Benchmark [16]

4.4.1 Size of TPC-C

The size of TPC-C dataset can be increased by adding number of warehouses. Our experiment uses TPC-C with 70 warehouses. The total data size of TPC-C for 70 warehouses in postgresql is 6591 MB. In the identification of hot and cold data for MMDB, the percentage of data that should be taken as hot also depends on available memory size. We are assuming that the size of data that can be kept in main memory should not be more than 10% of the memory size. It is so because; the remaining memory size is used for processing. We have system with 8 GB RAM size, so only up to 0.8 GB hot data can be kept in main memory. Since the size of TPC-C with 70 warehouses is 6.6 GB approx. and is nearer to main memory size, so it can be used for experimental purpose.

4.5 Query Set

The TPC-C is an OLTP workload, so we have considered queries without joins. Total 100 queries are designed for TPC-C dataset. Each query has been assigned a unique query id. All the 100 queries are given in Appendix A. Table 4.3 gives the number of queries corresponding to each TPC-C relation. All the query frequencies with respect to query ids are given in Appendix B.

TPC-C Relation Name	Total No. of queries (100)
Warehouse	7
District	16
Customer	24
History	5
Order	16
Item	9
Stock	8
Order_line	15

Table 4.3: Query Distribution for TPC-C Relations

4.5.1 Query Frequencies

The query frequencies have been assigned randomly to all the queries. On the basis of general understanding of dataset the frequency values are decided. In the real life scenario, these values can vary for the queries. In order to get to the exact frequency value of the queries, some method can be required. One such method can be log file analysis. All the query runs are logged by the database system. These log files can be used to estimate the query frequencies.

4.6 Implementation

The experimental flow of the QAA-HP technique is shown in Figure 4.2. It also gives the details of software tools used for the particular block of process.

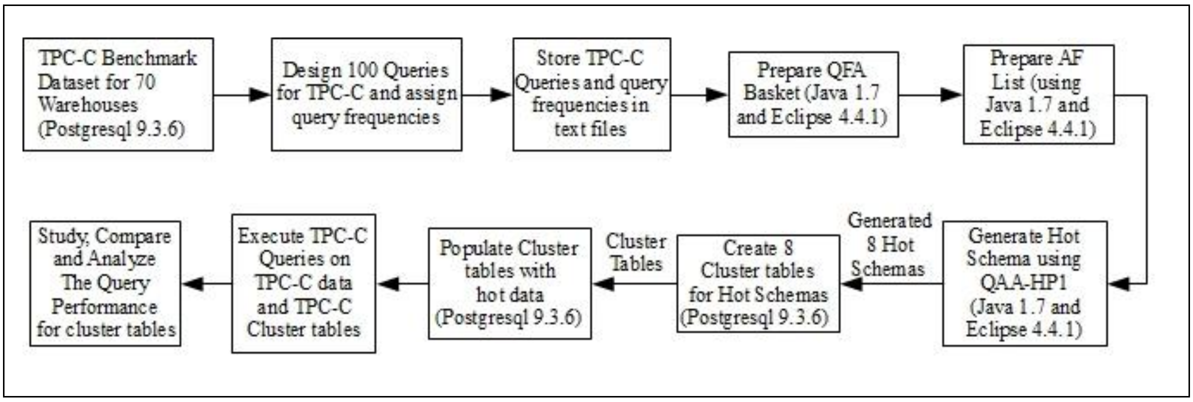


Figure 4.2: Experimental flow of QAA-HP

We are using TPC-C benchmark dataset for our experiment [16]. The data is populated using postgresql database tool. We have designed 100 queries for TPC-C and the query frequencies are assigned manually in the experiment. The code to prepare QFA basket is implemented in Java using eclipse IDE. The code for QFA basket scans the text files containing queries and query frequencies and stores the resulting attribute set and frequency of query in another text file. QFA basket is input to the code for preparing attribute frequency list (AF List). It scans each query to calculate the occurrence frequency for an attribute. The hot schema is generated using algorithm QAA-HP1 which has been implemented in Java. We are getting 8 clusters for the TPC-C schema. The hot clusters are populated in postgresql for query analysis. The performance parameters devised by us are used for analyzing the clusters performance.

Chapter 5

Results and Discussions

Experiments were performed for identification of hot and cold data. This chapter includes results of the experiment. It analyzes the hot cluster tables on the basis of query execution time and the percentage query workload answered by them. The performance parameters discussed in section 4.2, are part of this analysis.

The QAA-HP technique is used for identification of hot and cold data. The technique uses a pair of algorithms for identifying hot and cold data. The first algorithm is used to find the hot schema. We have defined the hot schema as a set of frequent attributes. It is the subset of the original schema achieved using vertical partitioning. The purpose of identifying hot schema is that not every attribute of a relation is part of a query. It is not meaningful to store the data of these attributes in main memory and process the data. Once the hot schema is identified, the data corresponding to hot schema is identified using algorithm QAA-HP2. The algorithms have been implemented for disk based system only.
The QAA-HP technique for identification of hot and cold data is demonstrated using TPC-C benchmark. The details of TPC-C are discussed in section 4.4. The size of TPC-C dataset can be increased by adding warehouses. Our experiments use TPC-C with 70 warehouses. The size of TPC-C dataset with 70 warehouses is 6591 MB. There are 9 tables in TPC-C dataset originally. We are getting 8 hot cluster tables from the algorithms QAA-HP.

5.1 Hot Clusters using QAA-HP

The hot clusters are constructed for hot schema obtained from algorithm QAA-HP1. The hot clusters contain subset of attributes from the original tables. It will possibly contain less number of tuples as compared to the original tables this is because; it only stores the data which the queries require. We are getting 8 hot clusters out of 9 tables for TPC-C dataset. The reason being, clusters are made based on frequent queries. The attributes occurring in frequent queries have higher affinity and more chances to get in clusters. The tables which are not accessed or have low frequency value have the possibility of not being in hot clusters.

5.1.1 Clustered Schema

The algorithm QAA-HP1 is run on the 100 queries designed for TPC-C. The clustered schema obtained from the algorithm is given in Table 5.1. After obtaining the clusters, they were populated with the hot data.

TPC-C Relations	Clusters	Query Ids for Queries Running on Clusters
Warehouse (w_id, w_name, w_street_1, w_street_2, w_city, w_state, w_zip, w_tax, w_ytd)	**ClusterWarehouse** (w_id, w_name)	
District (d_id, d_w_id, d_name, d_street_1, d_street_2, d_city, d_state, d_zip, d_tax, d_ytd, d_next_o_id)	**ClusterDistrict** (d_id, d_w_id, d_name, d_ytd)	Q3, Q14, Q26, Q29, Q41, Q49, Q68
Customer (c_id, c_d_id, c_w_id, c_first, c_middle, c_last, c_street_1, c_street_2, c_city, c_state, c_zip, c_phone, c_since, c_credit, c_credit_lim, c_discount, c_ytd_payment, c_payment_cnt, c_delivery_cnt, c_data	**ClusterCustomer** (c_id, c_d_id, c_w_id, c_first, c_credit, c_credit_lim)	Q1, Q2, Q5, Q19, Q20, Q30, Q39, Q78
History (h_c_id, h_c_d_id, h_c_w_id, h_d_id, h_w_id, h_data, h_amount, h_date)	**ClutserHistory** (h_c_id, h_c_d_id, h_c_w_id, h_amount)	Q12, Q25, Q73
Order (o_id, o_d_id, o_w_id, o_c_id, o_entry_d, o_carrier_d, o_ol_cnt, o_all_local)	**ClusterOrder** (o_id, o_w_id, o_d_id, o_c_id, o_entry_d, o_ol_cnt)	Q7, Q22, Q32, Q35, Q42, Q44, Q52, Q61, Q70, Q71, Q85, Q87
Order_line (ol_o_id, ol_d_id, ol_w_id, ol_number, ol_i_id, ol_supply_w_id, ol_delivery_d, ol_quantity, ol_amount, ol_dist_info)	**ClusterOrder_line** (ol_o_id, ol_w_id, ol_d_id, ol_number, ol_i_id, ol_supply_w_id)	Q10, Q17, Q23, Q40, Q48, Q53, Q65, Q95, Q99
Item (i_id, i_im_id, i_name, i_price, i_data)	**ClusterItem** (i_id, i_name, i_price)	Q4, Q50, Q81, Q90
Stock (s_id_id, s_w_id, s_quantity, s_dist_01, s_dist_02, s_dist_03, s_dist_04, s_dist_05, s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10, s_ytd, s_order_cnt, s_remote_cnt, s_data)	**ClusterStock** (s_i_id, s_w_id, s_quantity)	Q11, Q33, Q84, Q89

Table 5.1: TPC-C Schema and TPC-C Clustered Schema

The attribute clusters are made using frequent queries, so it can be observed that all the clustered schema have less number of attributes as compared to the original schema. Only the hot attributes are considered for hot clusters and query analysis is done for these hot clusters. We observed that 47% of the total queries are getting answered by the clusters. The query ids of the respective queries that can be answered by the clusters are provided in Table 5.1.

5.1.2 Percentage of Clustered Schema

The clusters are obtained by vertically partitioning the database schema so the hot schema has less number of attributes than original. The percentage of attributes from TPC-C schema used in clusters is given in Table 5.2. The percentage of queries out of total targeted queries for the relation that can be answered by the corresponding clusters is also given in Table 5.2.

TPC-C Relation	**No. of attributes in TPC-C benchmark schema**	**Percentage of attributes in Clusters (%)**	**Percentage of queries answered by Clusters out of total queries subjected to the relation (%)**
Warehouse	9	22.2	0
District	11	36.4	43.75
Customer	21	28.57	33.33
History	8	50	60
Order	8	75	75
Order_line	10	60	60
Item	5	60	44.44
Stock	17	17.65	50

Table 5.2: Percentage of Attributes in Clusters

The cluster for Order relation can answer 75% of the Order relation queries. Similarly, History and Order_line clusters can answer 60% of the total queries targeted to the corresponding TPC-C relation. The cluster for warehouse relation is not able to answer any query. The reason for this can be the queries for warehouse are possibly not part of the hot queries.

5.1.3 Clustered Data

The clusters are populated with data which is required by query workload. The clusters will mostly contain data less than the original tables. The percentage of data we are getting in the 8 clusters is given in Table 5.3.

TPC-C Relations	**No. of Tuples in TPC-C Relation**	**No. of Tuples in Clusters**	**Percentage of data in Clusters**
Warehouse	70	65	92.85
District	700	700	100
Customer	2100000	2100000	100
History	2100000	1400	0.066
Order	2100000	2100000	100
Item	100000	97892	97.89
Stock	7000000	100138	1.43
Order_line	19946000	6441382	32.29

Table 5.3: Percentage of Data in Clusters

The size of TPC-C data for 70 warehouses is 6591 MB. The size of 8 clusters is found to be 567.944 MB. It is observed that clusters contain only 9% of the total TPC-C data. Only 9% of data can answer 47% of the queries. The query performance and analysis for clusters is discussed in next section.

5.2 Performance Analysis for Original Data & Hot Clusters Data

There are total 100 TPC-C queries identified for the experiment. All the 100 queries with their query id are given in Appendix A and corresponding query frequencies in Appendix B. The algorithm for identification of hot and cold data, stores a part of TPC-C data as hot cluster data. The queries were executed for original TPC-C data as well as for hot clusters data.

The Performance analysis is done using performance parameters *Query Analysis* and *Query Execution time* discussed in section 4.2. It is observed that out of 100 queries, 47 queries can be answered by hot clusters.

From the 47 queries we check for number of hot queries that can be answered by the hot cluster tables. The top 50% of the total given queries are considered as *hot queries*. The hot queries are considered on the basis of query frequencies. The threshold for hot queries is set to be the median frequency value. All the queries having query frequency above median frequency value are considered as hot queries. The hot queries are the frequent queries of the system. It is observed that 52 queries are hot queries among the 100 TPC-C queries. From these 52 hot queries, it is found that only 31 hot queries can run on hot clusters. It can be said that out of 47 queries running on clusters, 31 are hot queries.

We can further classify the hot queries as the *hottest queries*. Given all the sorted queries in descending order of their frequency count, the top 10% of the queries are considered to be the hottest queries. In our query set of 100 queries, we are getting 13 hottest queries. It is observed that out of these 13 queries, 9 queries can be answered by the hot cluster tables.

We will be considering these 9 hottest queries for our analysis. The query set {Q1, Q2, Q3, Q4, Q5, Q7, Q9, Q10, Q11} is used for analysis of the thesis work. These 9 queries are also denoted as *most frequent queries*.

5.2.1 Query Analysis

The Query Analysis is done based on two factors, Query Load *QL* and Query Workload *QW*. *QL* estimates the percentage of number of queries that can be answered by clusters. *QW* estimates the percentage of amount of query workload that can be answered by the clusters. The details of number of queries running on cluster are given in Table 5.4. We have calculated the percentage of number of queries that can be answered by hot clusters.

	No. of Queries	No. of Queries answered by Clusters	Percentage of queries answered by clusters (QL)
Total Queries$((QL)_{AllQuery}$	100	47	47%
Hot Queries$((QL)_{HotQueries})$	52	31	60%
Hottest Queries$((QL)_{HottestQueries})$	13	9	70%

Table 5.4: Query Statistics for Hot Clusters

From Table 5.4, it is observed that the clusters can answer 47% of the total TPC-C queries. If considering the hot queries, 62% of the hot queries are getting answered by the hot clusters. The percentage of hottest queries getting answered by clusters is observed to be 70%.

If we consider the total workload for queries, which is the summation of frequencies of all the queries. It is estimated to be 6013 in our case. We need to check the amount of workload that can be handled by the clusters. The workload for each type of query set is given in Table 5.5.

	No. of Queries	Total Workload for each query type
Total TPC-C Queries	100	6013
Total Queries Answered by Clusters	47	3540
Total Hot Queries	52	5295
Hot Queries Answered by Clusters	31	3325
Total Hottest Queries	13	2045
Hottest Queries Answered by Clusters	9	1495

Table 5.5: Workload Information for each Query Type

The workload for 100 queries denotes our total query workload. The 47 queries represent the workload handled by clusters and 31 queries give the amount of workload handled by hot queries through clusters. The hottest 13 queries give information for the hot workload. The percentage of amount of workload handled by clusters is given in Table 5.6. The amount of workload handled by clusters is 3540, which is 59% of the total workload. The amount of workload handled by hot queries is given 3325 in Table 5.6. It is 63% of the total workload handled by hot queries. Similarly, the hot workload is given by 13 queries having value 2045. It can be observed from the information given in Table 5.6 that 73% of the hot workload can be answered by clusters.

	Total Workload	Amount of Workload answered by Clusters	Percentage of Workload Answered by Clusters (QW)
All Queries	6013	3540	59%
Hot Queries	5295	3325	63%
Hottest Queries	2045	1495	73%

Table 5.6: Percentage Amount of Workload Handled by Clusters

5.2.2 Query Execution Time QET

The query execution time for original TPC-C data ($(QET)_{original}$) and query execution time for hot clusters data ($(QET)_{HotCluster}$) is calculated both for cold runs and hot runs. The average of three query runs is taken for each query. We are considering cold runs to be the query execution when cache is empty. The query execution when the query result is already cached is considered as hot runs. The QET for cold runs and hot runs for all the 47 queries is provided in Appendix C.

In case of 47 queries, the average ($(QET)_{original}$) for hot runs is estimated to be 7037 ms and ($(QET)_{HotCluster}$) is estimated 6245 ms. All the queries running on clusters provide an average percentage time gain of 22%.

There are 31 hot queries running on clusters. The top 31 queries given in Appendix C are the hot queries. The QET for original data and clusters and the time gain for hot runs is given in Appendix C. The average hot runs time for hot queries when run on original data is observed to be 10187 ms and 9176 ms on clusters. The hot queries when run on clusters provide a time gain of 19%. Similarly, for cold runs the average time taken by original data is 12026.63806 ms and average time taken by clusters is 9404 ms. It provides a time gain of 2623 ms.The total time saved is estimated to be 2989879 ms.

5.3 Performance Analysis for Hottest Queries Data

The definition for hottest queries is discussed in section 5.2. We got top 13 queries to be the hottest, out of which 9 queries can run on clusters. The query execution time for original TPC-C data ($(QET)_{original}$) and query execution time for hot clusters ($(QET)_{HotCluster}$) is given in the Table 5.7. All the QET are given in milliseconds. The average of three cold runs and three hot runs are taken as QET of a query. The average time for cold runs for original TPC-C data is estimated to be 19153 ms whereas for cluster tables it is estimated to be 14660 ms. There is a time gain of 4493 ms for cold runs when queries are running on clusters.

5.3.1 Query Analysis

As discussed in section 5.2.1. The hottest queries when run for clusters, the value of $((QL)_{HottestQueries}$) is estimated to be 70%. The query workload QW for all the hottest queries is 2045. The query workload for hottest queries running on clusters is estimated to be 1495 i.e. hottest queries can answer 73% of the amount of hot workload

5.3.2 Query Execution Time

This section gives the details about the hottest query runs and there execution time when run on original TPC-C data and hot clusters. Graphs are plotted against hottest queries and the QET for hot and cold runs. The graph represents the comparison between $(QET)_{Original}$ **and** $(QET)_{HotCluster}$ for queries when run on original TPC-C data and hot cluster data respectively. The graph for hot runs for queries is represented in Figure 5.1. The QET for cold runs for queries are represented in graph given in Figure 5.2. The time gain TG analysis for hottest queries is also performed and is discussed later in this section.

QID	Average $(QET)_{Original}$ for Cold Runs(in ms)	Average $(QET)_{HotCluster}$ for Cold Runs(in ms)	Average $(QET)_{Original}$ for Hot Runs(in ms)	Average $(QET)_{HotCluster}$ for Hot Runs(in ms)
Q1	97306.67	97296.34	97521.34	97161.34
Q2	11678.34	1021	495.67	253
Q3	41	54	11.46666667	10.6
Q4	4266.67	4305.67	4283.666667	4280
Q5	12181.67	1615.67	522.67	261.34
Q7	26940.67	26943.34	26820	26796.67
Q9	655.34	49.34	651	11.34
Q10	19260.34	608.67	521.34	154.34
Q11	45.67	45.67	11.67	11
Average	**19152.93**	**14659.96667**	**14537.64704**	**14326.62556**

Table 5.7: QET for Hottest Queries

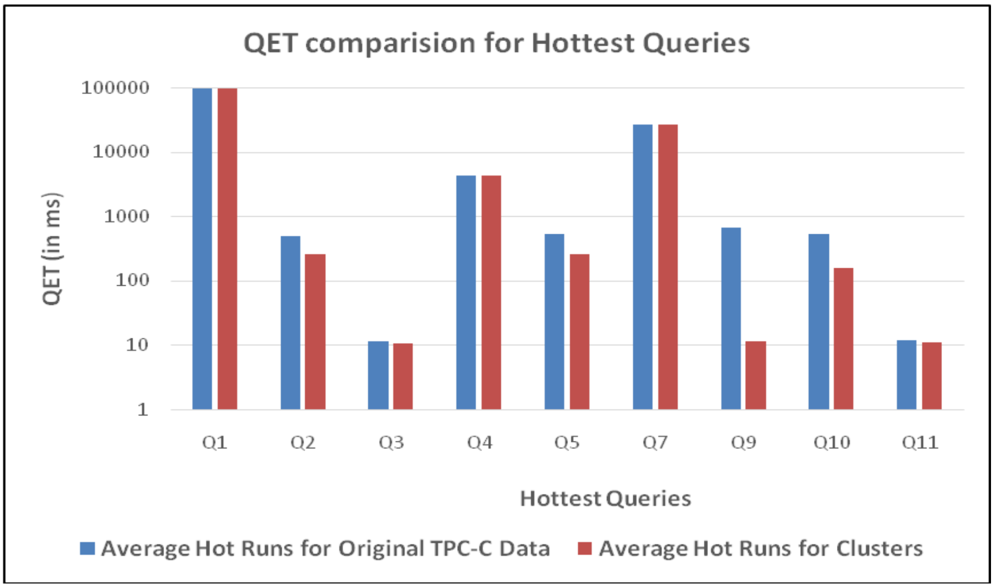


Figure 5.1: Graph for QET for Hottest Queries (Hot Runs)

From the graph given in Figure 5.1, it is observed that there is an improvement in QET for queries when run on clusters as compared to original TPC-C data. Query Q9 provide 98% time gain *TG%* when run for hot cluster table. Query 10 observes 70% of time gain. Similarly, Q2 and Q5 are having gain of 49% and 50% respectively.

When considering the cold runs, given in Figure 5.2, query 10 gives the best performance for clusters. It is providing a time gain of 96.8%. The readings for time gain are given in Table 5.8.

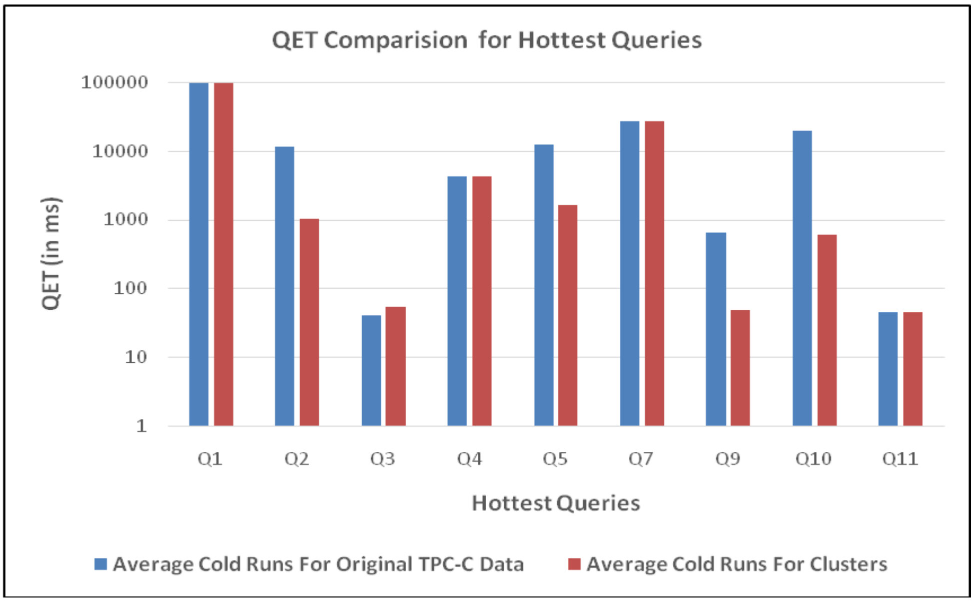


Figure 5.2: Graph for QET for Hottest Queries (Cold Runs)

QID	Average $(QET)_{Original}$ for Hot Runs(in ms)	Average $(QET)_{HotCluster}$ for Hot Runs(in ms)	Time Gain (TG)	Gain in percentage $(TG\%)$	Time Saved (TS)in ms
Q1	97521.34	97161.34	360	0.369149973	79200
Q2	495.67	253	242.67	48.95797607	48534
Q3	11.46666667	10.6	0.866666667	7.558139535	173.3333333
Q4	4283.666667	4280	3.666666667	0.085596452	733.3333333
Q5	522.67	261.34	261.33	49.99904337	39199.5
Q7	26820	26796.67	23.33	0.086987323	3499.5
Q9	651	11.34	639.66	98.25806452	79957.5
Q10	521.34	154.34	367	70.39551924	45875
Q11	11.67	11	0.67	5.741216795	83.75
Average	**14537.64704**	**14326.62556**	**211.0214815**	**31.27241036**	**33028.43519**

Table 5.8: QET Analysis for Hottest Queries

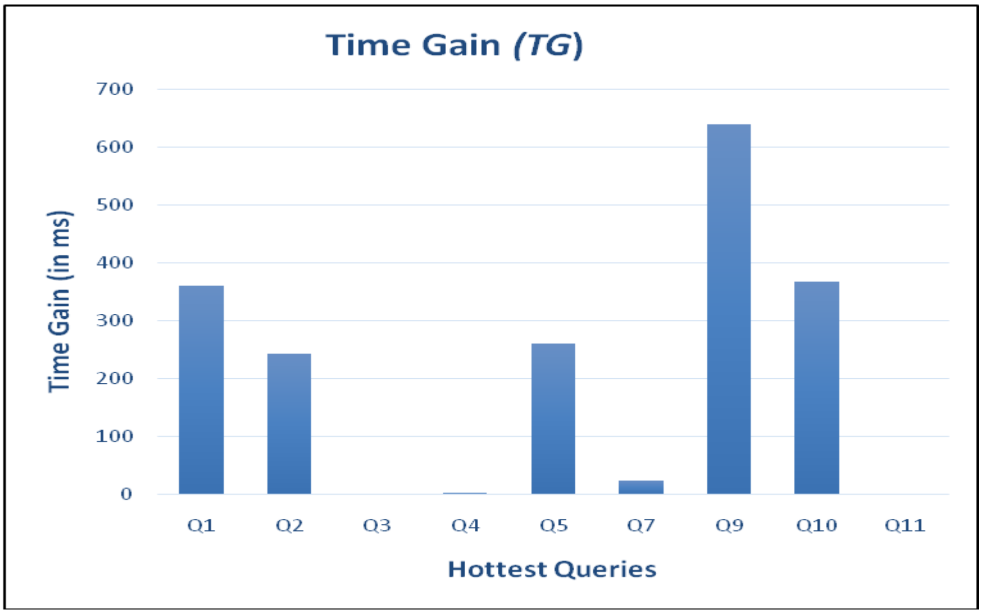


Figure 5.3: Time Gain for Hottest Queries

The time gain is a part of QET analysis. The queries were run on clusters, we have calculated the time gain TG and total time saved TTS using *Eq. (4.7)* and *Eq. (4.9)* .The QET analysis is done using TG and $TG\%$. TS for hottest queries is given in Table 5.8. The overall estimated time gain for hottest queries is 211 ms. The difference in time gain for all the queries is shown in Figure 5.3 with the help of a graph. The graph is plotted between hottest queries and their respective TG for hot runs. As shown in Figure 5.3, query Q9 provides time gain of 640 ms. The total time saved by hottest queries is estimated to be 297256 ms.

Chapter 6

Conclusion and Future work

With the skewed access patterns in database applications, the data can be divided into hot and cold data. The hot data can be stored separately so that the applications need to process only the useful data. Hot data can be stored in main memory and cold data can be stored in secondary storage. Different storage techniques which are appropriate for specific kind of data i.e. hot or cold can be used, which would further accelerate query execution.

6.1 Conclusion

When main memory database are to be used, for optimal utilization of the memory it is advisable not to keep the entire dataset in the memory. The thesis work presents a technique named Query Aware Approach using Hybrid Partitioning (QAA-HP) for identification of hot and cold data. The approach use query workload information for identifying hot schema and hot data. The experiment was performed using TPC-C benchmark and the queries were designed for it. The size of original data is 6591 MB and the clusters are estimated to be 568 MB in size. This implies that clusters store only 9% of the original data. We demonstrated that hot clusters formed using hot schema and hot data can answer 47% of the query load $((QL)_{AllQuery})$ and 59% of the query workload $((QW)_{AllQueries})$. It can be said that 9% of data can answer 59% of the total query workload (QW). When considering only hot queries $((QL)_{HotQueries})$ we observed that approximately 60% of the hot queries can be answered by hot clusters. Analyzing cluster performance only for the hottest queries, we get that 9% of hot data can answer 73% of the hottest query workload $((QW)_{HottestQueries})$. The clusters were also analyzed on the basis of query execution time (QET). For the hottest queries the percentage of time gain $(TG(\%))$ when run on hot clusters is observed to be 37 % for cold runs and 31 % for hot runs. The hot queries of the system show a time gain of 19% when run on hot clusters. Similarly, for all the queries that are running on clusters, the time gain was estimated to be 22%. We have observed that clusters can improve the query performance. Only 9% of the data is accessed by most of the queries so there is no need to store the remaining 90% of the data in memory. It can be kept in secondary storage.

We are considering the data to be static and read only in our experiment. The frequency of the queries was set manually for the experiment. The actual frequencies need to be estimated. One possible way can be the log based method. The queries executed for the system can be logged and the frequencies can be estimated by scanning the log files. The query workload (QW) may change with time, so the algorithm should check the logs regularly to cope up with the dynamic environment.

6.2 Future Work

The QAA-HP technique for identification of hot and cold data has been implemented for disk based system and can be extended to main memory database systems. It works for static environment and needs to be enhanced for dynamic environment. The hot data can be kept in main memory and the cold data is to be kept on disk. The decision of memory allocation for hot data can be made dynamically and it may vary with the available memory size. In dynamic environment, the hot data may change over a period of time, so there will be a need to rerun the algorithm for identification of hot and cold data at certain frequency. The frequency of algorithm rerun also depends on the type of application. In some applications data may change very frequently where it is required to rerun the algorithm with higher frequency. On the other hand, for some applications the data changes happen over a longer period of time. In such cases the algorithm need not have to be run frequently. The data should be moved back to disk when it is not a part of hot set. There is need for algorithms which can move data from memory to disk seamlessly without affecting the application performance.

References

1. Tim Berners-Lee, James Hendler and Ora Lassila: The Semantic Web. *Scientific American. May 17,* 2001
2. Hooran Mahmoudi Nasab, Sherif Sakr. Efficient and Adaptable Query Workload-Aware Management for RDF Data. *In proc. of 11th International Conference, on web information system engineering, pp. 390-399, Dec., 2010*
3. D. J. Abadi, Adam Marcus, S.R. Madden, Kate Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. *In proceedings of VLDB* 2007.
4. Sandeep Vasani, Mohit Pandey, Minal Bhise and Trupti Padiya. Faster Query Execution For Partitioned RDF Data. *In proc. ICDCIT, vol. 7753, pp. 547-560,* 2013
5. Bhavik Shah. Structure Indexed Vertical Partition and Query Execution for RDF data. *M.Tech thesis, T00470, Resource Centre, DA-IICT, July* 2014
6. Prashant Rajkotiya. Data Aware Hybrid Storage Technique and Query Execution for RDF data. *M.Tech thesis T00452 Resource Centre, DA-IICT. October* 2014
7. J. Levandoski, P Larson, R. Stoica. Identifying Hot and Cold Data in Main-Memory Databases. *In proc. of International conference on Data Engineering*. 2013
8. Justin J. Levandoski, M.F. Mokbel. RDF Data-Centric Storage*. In proc. Of ICWS 2009: pp. 911-918, Jan.* 2009
9. Trupti Padiya, Minal Bhise, Sandeep Vasani., Mohit Pandey. Query Execution for RDF Data on Row and column Store. *In proc. ICDCIT ,vol. 8956, pp 403-408,* 2015
10. H. Gracia-Molina, K. Salem: Main Memory Database Systems: An Overview. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL.4, NO. 6, December,* 1992
11. M.Boissier, H. Plattner. Optimizing Main Memory Utilization of Columnar In-Memory Databases Using Data Eviction. *In proceedings of VLDB, Phd Workshop,* 2014
12. S. Harizopoulos, D. Abadi, S. Madden, M. Stonebraker. OLTP Through the Looking Glass, and What We Found There. *SIGMOD'08, June 9-12,*2008
13. K.Ren, A. Thomson, D. Abadi. Lightweight Locking for Main Memory Database Systems. *In proceedings of VLDB,* 2013.
14. R. Agrawal, R. Srikant. "Fast Algorithm for Mining Association Rules", *in VLDB,* 1994.
15. Details of TPC-C available on: http://www.tpc.org/tpcc/detail.asp (last accessed on: 24 March 2015)
16. Document for TPC-C http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5-11.pdf
17. M.Tamer Ozsu, P.Valduriez, *Principles of Distributed Database Systems (2Nd Ed.).* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

Appendix A

TPC-C 100 Queries

Query#	Description	Query
1	List customers whose credit is good	SELECT c_id,c_d_id,c_w_id,c_first FROM customer WHERE c_credit = 'GC'
2	What is id of customer 'xyz'	SELECT c_id, c_first FROM customer WHERE c_first = '?<<z[Fo{b.xAU'
3	Give sum of amount of tax paid by all the districts.	SELECT sum(d_ytd), d_id FROM district GROUP BY d_id
4	Give the name of items having price > 12	SELECT i_id,i_name FROM item WHERE i_price > 12
5	What is the current balance and credit limit of customer 'JfT`L|5we'	SELECT c_first,c_credit_lim FROM customer WHERE c_first = 'wWdM&#D!Z'
6	List warehouse with sale tax > 0.012	SELECT w_id,w_name FROM warehouse WHERE w_tax > 0.012
7	Which order contain more than 10 items	SELECT o_id,o_d_id,o_w_id FROM order WHERE o_ol_cnt > 10
8	Give the quantity of ordered items for item id 12	SELECT ol_quantity FROM order_line WHERE ol_i_id = 12
9	Give those customer id who paid maximum amount.	SELECT max(h_amount),h_c_id FROM history GROUP BY h_c_id
10	List items for order id '12'	SELECT ol_i_id FROM order_line WHERE ol_o_id = 12
11	What is the quantity of stock for item no. 1 coming from warehouse 1	SELECT s_quantity FROM stock WHERE s_i_id = 1 AND s_w_id = 1
12	Print customer name whose balance is 50 (more than some amount).	SELECT c_first FROM customer WHERE c_balance = 50
13	Give payment date for customer between 12 to 30	SELECT h_c_id,h_date FROM history WHERE h_c_id BETWEEN 12 AND 30
14	Give name and id of the district belonging to warehouse 1	SELECT d_id,d_name FROM district WHERE d_w_id = 1
15	give carrier id for orders by customer 23	SELECT o_id,o_c_id,o_carrier_id FROM order WHERE o_c_id = 23
16	The item xyz belong to which brand?	SELECT i_id,i_data FROM item WHERE i_name = '<S=_iJ#(!@hjjksl/_'
17	Give supplier warehouse for item id 1 to 10	SELECT ol_supply_w_id FROM order_line WHERE ol_i_id BETWEEN 1 AND 10
18	List customers from city 'xyz'	SELECT c_first FROM customer WHERE c_city = 'BU4Fr4X8il^_$9'
19	Print name of customers between c_id = 1 to 4	SELECT c_first FROM customer WHERE c_id BETWEEN 1 AND 4
20	Give details of customer who have credit limit more than 5000	SELECT c_id,c_d_id,c_first FROM customer WHERE c_credit_lim > 5000
21	Name of the customer who has payment count more than 50000	SELECT c_first FROM customer WHERE c_payment_cnt > 50000
22	No. of orders placed by customer id 23	SELECT count(o_id),o_ol_cnt FROM order WHERE o_c_id = 23 AND o_d_id = 1 AND o_w_id = 1 GROUP BY o_ol_cnt
23	Give the supplier warehouse's id for order having id 2	SELECT ol_supply_w_id FROM order_line WHERE ol_o_id = 2

24	Give those district name and id having sales tax value < 10	SELECT d_id,d_name FROM district WHERE d_tax < 10
25	What is the amount paid by customer 2	SELECT h_c_id,h_c_d_id,h_c_w_id, h_amount FROM history WHERE h_c_id = 2
26	Give id of the district with their maximum balance.	SELECT distinct d_id,max(d_ytd) FROM district GROUP BY d_id
27	Give balance for item id 1	SELECT s_ytd FROM stock WHERE s_i_id = 1
28	Give item image id for item having price less than 10	SELECT i_im_id FROM item WHERE i_id BETWEEN 1 AND 10
29	Print year to date balance of all districts belonging to warehouse 1	SELECT d_id,d_ytd FROM district WHERE d_w_id = 1
30	List customers whose credit is bad	SELECT c_id,c_first FROM customer WHERE c_credit = 'BC'
31	What is the balance available at warehouse 2	SELECT w_ytd FROM warehouse WHERE w_id = 2
32	Number of orders for a given warehouse.	SELECT o_w_id, sum(o_ol_cnt) FROM order WHERE o_w_id = 1 GROUP BY o_w_id
33	Give item id and its average quantity. (using group by)	SELECT s_i_id, avg(s_quantity) FROM stock GROUP BY s_i_id
34	Calculate the quantity of items ordered FROM warehouse 1	SELECT sum(s_order_cnt),s_w_id FROM stock GROUP BY s_w_id
35	give customer id and order entry date for order-line count > 10	SELECT o_c_id,o_entry_d FROM order WHERE o_ol_cnt > 10
36	What is the amount paid for order id 134	SELECT ol_amount FROM order_line WHERE ol_i_id = 134
37	Give delivery date of order id 1 from district 1	SELECT ol_delivery_d FROM order_line WHERE ol_o_id = 1 AND ol_d_id = 1
38	Print warehouse names and there balance till date	SELECT w_name,w_ytd FROM warehouse
39	Give id and credit limit of customer whose first name is ‘wzxy’	SELECT c_first,c_credit_lim FROM customer WHERE c_first = 'FHubBypiD'
40	Give those order ids and item id who have warehouse 1 as supplier	SELECT ol_o_id,ol_i_id FROM order_line WHERE ol_supply_w_id = 1
41	Give district id and year to date balance for them.	SELECT d_id, d_ytd FROM district
42	Give order ids of order from district 1 of warehouse 1	SELECT o_id FROM order WHERE o_d_id = 1 AND o_w_id = 1
43	Give details of stock maintained by warehouse 1	SELECT s_w_id,s_i_id,s_order_cnt,s_quantity FROM stock WHERE s_w_id = 1
44	Give order entry date for customer between 10 and 15	SELECT o_entry_d FROM order WHERE o_c_id BETWEEN 10 AND 15
45	What is next available order number of district 5	SELECT d_next_o_id FROM district WHERE d_id = 5
46	Give id and payment count of customer whose first name is ‘wxyz’	SELECT c_id,c_payment_cnt FROM customer WHERE c_first = '(rmDfI|m>'
47	Give name and id of warehouse which belong to state ‘wr'	SELECT w_id,w_name FROM warehouse WHERE w_state = 'wr'
48	List of items for order id 1	SELECT ol_i_id FROM order_line WHERE ol_o_id = 1
49	Number of districts which comes under warehouse 1	SELECT count(d_id) FROM district WHERE d_w_id = 1
50	Calculate the average price of items having id between 1 to 4	SELECT avg(i_price),i_id FROM item WHERE i_id BETWEEN 1 AND 4 GROUP BY i_id
51	Give delivery date of item id 520	SELECT ol_delivery_d,ol_i_id FROM order_line WHERE ol_i_id = 520

52	Entry date for order ids 9 to 14	SELECT o_id,o_entry_d FROM order WHERE o_id BETWEEN 9 AND 14
53	Count no. Of orders for district id 1 belonging to warehouse 1	SELECT count(ol_o_id) FROM order_line WHERE ol_d_id = 1 AND ol_w_id = 1
54	Give full name of customer ‘xyz’	SELECT c_first,c_middle,c_last FROM customer WHERE c_first = 'd<r7t2V\'
55	Print discount available with customer associated to district id 9	SELECT c_id,c_discount FROM customer WHERE c_d_id = 9
56	Give name and city WHERE customer belong to for c_id 25 and district id 1	SELECT c_id,c_d_id,c_first,c_city FROM customer WHERE c_id = 25 AND c_d_id = 1
57	Order ids of order coming from local warehouse	SELECT o_id FROM order WHERE o_all_local = 1
58	List the warehouse belonging to city ‘xyz’	SELECT w_id,w_name FROM warehouse WHERE w_city = 'jw{Nzy.hs,:q)H] '
59	Give next available order number of district 2	SELECT d_id,d_next_o_id, d_w_id FROM district WHERE d_id = 2
60	Give details of item which belong to same brand	SELECT i_id,i_name FROM item WHERE i_data = 'yqFHKAKH6rqb7?-M5"3LBAzWC0(eR%@u5G^gL!`bIB'
61	Entry date of orders from district id 10 of warehouse 2	SELECT o_d_id,o_w_id,o_entry_d FROM order WHERE o_d_id = 10 AND o_w_id = 2
62	Give amount paid for item id 10 to 16	SELECT ol_amount,ol_i_id FROM order_line WHERE ol_i_id BETWEEN 10 AND 16
63	Give sum of amount of tax paid by all the warehouses	SELECT sum(w_tax) FROM warehouse
64	Print phone number of customer xyz	SELECT c_phone FROM customer WHERE c_first = '3h;7qOV(;'
65	Give the order ids who have warehouse 2 as supplier	SELECT ol_o_id FROM order_line WHERE ol_supply_w_id = 2
66	Give full address (street, city, state, zip) of warehouse 1	SELECT w_id,w_street_1,w_street_2,w_city,w_state,w_zip FROM warehouse
67	Give order id, district id, customer id for orders which are from remote warehouse	SELECT o_id,o_d_id,o_c_id FROM order WHERE o_all_local = 0
68	Name of districts which comes under warehouse 2	SELECT d_name FROM district WHERE d_w_id = 2
69	Which city does customer xyz belong to	SELECT c_city FROM customer WHERE c_first = 'xyz'
70	Give order entry date for customer 1	SELECT o_entry_d,o_c_id FROM order WHERE o_c_id = 1
71	What is the maximum items count in an order	SELECT max(o_ol_cnt) FROM order
72	Give carrier id for order id 1	SELECT o_carrier_id FROM order WHERE o_id = 1
73	Calculate the sum of amount paid by customer id 12	SELECT sum(h_amount) FROM history WHERE h_c_id = 12
74	Give details of customer belonging to state ‘39'	SELECT c_id,c_d_id,c_w_id,c_city FROM customer WHERE c_state = '39'
75	Give the address of district 1 from warehouse 1.	SELECT d_id,d_w_id,d_street_1,d_street_2,d_city,d_state,d_zip FROM district WHERE d_id = 1 AND d_w_id = 1
76	Give city and state of district id 4 from warehouse 2	SELECT d_city,d_state FROM district WHERE d_id =4 AND d_w_id = 2
77	Print full name of customer belonging to district 10 of warehouse 2	SELECT c_first,c_middle,c_last FROM customer WHERE c_d_id = 10 AND c_w_id = 2
78	Print customer id of a customer associated with warehouse 1	SELECT c_id FROM customer WHERE c_w_id = 1

79	Give amount of tax paid by district 9.	SELECT d_id,d_name,d_tax FROM district WHERE d_id = 9
80	Give number of orders for item id 2	SELECT s_order_cnt,s_i_id FROM stock WHERE s_i_id = 1 AND s_w_id = 1
81	Give the name of items having item_id = 1	SELECT i_name FROM item WHERE i_id = 1
82	Give payment date of customer 21	SELECT h_c_id,h_c_d_id,h_c_w_id, h_date FROM history WHERE h_c_id = 21
83	Give item image id for item xyz	SELECT i_im_id FROM item WHERE i_name = 'o\xV=cQ0BMf_3;pUh'
84	Give quantity of stock for item id 54	SELECT s_i_id,s_quantity FROM stock WHERE s_i_id = 54
85	Give order entry date for customer from distict 4 having id 3000	SELECT o_entry_d FROM order WHERE o_c_id = 3000 AND o_d_id = 4
86	Give phone number of customer belonging to city ':oN5WpprbB*=5z'	SELECT c_id,c_phone FROM customer WHERE c_city = 'S=v3[F|2dvE'
87	Give count of order-lines for order id 'abcd'	SELECT o_d_id,o_w_id,o_ol_cnt FROM order WHERE o_id = 1
88	Count the no. of districts for a warehouse belonging to state 'xyz'	SELECT count(d_id), d_w_id FROM district WHERE d_state = 'i\' GROUP BY d_w_id
89	Calculate the minimum stock quantity needed	SELECT min(s_quantity), s_w_id FROM stock GROUP BY s_w_id
90	Give id and names of items having price < 10	SELECT i_id,i_name FROM item WHERE i_price < 10

91	Give year to date payment for customer id 35	SELECT c_ytd_payment FROM customer WHERE c_id = 35
92	Print address of customer xyz	SELECT c_id,c_first,c_street_1,c_str eet_2,c_city,c_state,c_zip FROM customer WHERE c_first = ']/;@,8@i'
93	Give district name and zip code of district id 10	SELECT d_name,d_zip FROM district WHERE d_id = 10
94	Give brand information of item id 1000	SELECT i_data FROM item WHERE i_id = 1000
95	Give details of order, district id and warehouse id for item id 6	SELECT ol_o_id,ol_d_id FROM order_line WHERE ol_i_id = 6
96	Give address of district 10 from warehouse 2	SELECT d_id,d_w_id,d_street_1,d_st reet_2,d_city,d_state,d_zip FROM district WHERE d_id = 10 AND d_w_id = 2
97	Give full name of customer belonging to city 'abc'	SELECT c_first,c_middle,c_last FROM customer WHERE c_city = 'TmnXjkCB[5zI#@l'
98	Print discount available with customer 250	SELECT c_discount FROM customer WHERE c_id = 250
99	Print order id and corresponding supplier warehouse id for orders id between 1 to 10	SELECT ol_o_id ,ol_supply_w_id FROM order_line WHERE ol_o_id BETWEEN 1 AND 10
100	Calculate sum of amount paid for item id 134	SELECT sum(ol_amount) FROM order_line WHERE ol_i_id = 134

Appendix B

Query Frequency List

Query#	Frequency
1	220
2	200
3	200
4	200
5	150
6	150
7	150
8	150
9	125
10	125
11	125
12	125
13	125
14	120
15	120
16	110
17	110
18	100
19	100
20	100
21	100
22	100
23	100
24	100
25	100
26	100
27	100
28	100
29	100
30	90
31	90
32	90
33	90
34	90
35	90
36	80
37	80
38	80
39	70
40	70
41	70
42	70
43	60
44	60
45	60
46	50
47	50
48	50
49	50
50	50

Query#	Frequency
51	50
52	50
53	45
54	40
55	40
56	38
57	35
58	30
59	30
60	30
61	30
62	30
63	25
64	20
65	20
66	20
67	15
68	15
69	15
70	15
71	15
72	15
73	15
74	12
75	12
76	12
77	12
78	10
79	10
80	10
81	10
82	10
83	10
84	10
85	6
86	5
87	5
88	5
89	5
90	5
91	5
92	4
93	4
94	4
95	3
96	3
97	3
98	2
99	2
100	1

Appendix C

QET for 47 Queries

QID	Average Cold Runs Original TPC-C Data (in ms)	Average Cold Runs Clusters (in ms)	Average Hot Runs forOriginal TPC-C Data (in ms)	Average Hot Runs forClusters (in ms)	Time Gain for Hot Runs (in ms)
Q1	97306.66667	97296.33333	97521.33333	97161.33333	360.0000033
Q2	11678.33333	1021	495.6666667	253	242.6666667
Q3	41	54	11.46666667	10.6	0.866666667
Q4	4266.666667	4305.666667	4283.666667	4280	3.666666667
Q5	12181.66667	1615.666667	522.6666667	261.3333333	261.3333334
Q7	26940.66667	26943.33333	26820	26796.66667	23.33333
Q9	655.3333333	49.33333333	651	11.33333333	639.6666667
Q10	19260.33333	608.6666667	521.3333333	154.3333333	367
Q11	45.66666667	45.66666667	11.66666667	11	0.666666667
Q14	32.33333333	48.33333333	12	11.33333333	0.66666667
Q17	3149	116.3333333	3146	112	3034
Q19	634.6666667	668.6666667	93.13333333	304.2666667	-211.1333333
Q20	92623	92918.33333	92840.33333	92858.86667	-18.53333333
Q29	31.33333333	65.66666667	10.93333333	11.73333333	-0.8
Q26	53	11	11.8	11.66666667	0.133333333
Q25	1607.666667	32.33333333	245	31.66666667	213.3333333
Q22	58	1325	12	12	0
Q23	590.6666667	205	589	114.3333333	474.6666667
Q30	7949.333333	7957.333333	7939.533333	7932.8	6.733333333
Q35	39492.66667	39591.33333	39589.53333	39569.26667	20.26666667
Q32	69	1681.333333	23.66666667	22	1.666666667
Q33	29421.33333	5015	30136	4931	25205
Q39	11533	264.3333333	526.3333333	253	273.3333333
Q41	34.33333333	18.66666667	22.33333333	22.33333333	3.3E-09
Q42	62.33333333	63.33333333	63	62.66666667	0.33333333
Q40	8418.333333	8437.666667	8423.933333	8424.4	-0.466666667
Q44	848.6666667	354.3333333	357	351	6
Q49	91.33333333	11.33333333	11.93333333	11.26666667	0.666666667
Q50	31.66666667	18.66666667	23.66666667	12	11.66666667
Q48	3069.666667	131.6666667	566.6666667	132.3333333	434.3333334
Q52	648	648	317.4	313.6	3.8
Q53	105	1568	21.46666667	23.33333333	-1.866666667

Q61	212.6666667	1596	164.6666667	162.6666667	1.999999967
Q65	16787.66667	5283.666667	5263.333333	5244.666667	18.66666633
Q68	72.33333333	55.66666667	11.66666667	11.66666667	0
Q71	1147.666667	1284.666667	283.3333333	273	10.33333333
Q70	230	249.6666667	228.4666667	247.8666667	-19.4
Q73	1628.333333	78.66666667	213	12	201
Q78	606.3333333	2703.333333	556	565.5333333	-9.533333333
Q81	95.33333333	68.66666667	11.6	11.86666667	-0.266666667
Q84	1884.333333	98	171	21.33333333	149.6666667
Q85	1184	1389	226.6666667	226.3333333	0.333333367
Q90	478.3333333	448.6666667	442	439.6666667	2.3333333
Q89	20879	35	1995.333333	31	1964.333333
Q87	1140.666667	206.3333333	216.3333333	206	10.33333333
Q95	2210.666667	83.33333333	2217.666667	79.33333333	2138.333333
Q99	17359.66667	1527.333333	2914	1503	1411
Average	**9337.184397**	**6557.432624**	**7036.947518**	**6244.774468**	**792.1730496**