UNIVERSITY OF CINCINNATI

26th July , 20 01

I, Rajeswari Malladi **,**
hereby submit this as part of the requirements for the degree of:

Master of Science

in:

Computer Science

It is entitled:

Applying Multiple Query Optimization in Mobile Databases

Approved by:

Dr. Karen Davis

Dr. Dharma Agrawal

Dr. Carla Purdy

Applying Multiple Query Optimization in Mobile Databases

A thesis submitted to the

Division of Graduate Studies and Research
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of Electrical & Computer Engineering and
Computer Science
of the College of Engineering

July 26, 2001

by
Rajeswari Malladi

B.E., College of Engineering, Osmania University, India, 1998
Thesis Advisor and Committee Chair: Dr. Karen C. Davis

Abstract

Mobile computing is a fast growing research and commercial area. An important application of mobile networks is data dissemination over limited bandwidth channels. There are different modes of data dissemination: push-based, pull-based, or a combination of both. In push-based, the data is broadcast in the form of broadcast disks. In pull-based, a mobile unit sends an uplink query to a central server, the server processes the data and sends the answer on a downlink channel. If the number of uplink queries is large, a lot of channel bandwidth is expended in sending the answers on the downlink channels. In this study, we apply multiquery optimization to batches of pull requests in mobile databases. Materialized views are created that can be used to answer several queries at once. The materialized views are then broadcast on a push-pull channel dedicated for this purpose (answers to multiple pull queries). Each mobile unit receives a short message from the server that contains information about when and for how long to tune to the channel to retrieve the requested information. We compare multiple query processing for pull requests (MQPR) with a basic pull request method (PR) in which each query is handled separately. Appropriate algorithms and formulae are given to calculate the bandwidth usage and the wait time for the mobiles sending the requests. A performance study is conducted by simulating different query loads over a testbed schema. The studies indicate a significant savings in the channel bandwidth usage and also a significant reduction in the wait time in MQPR compared to PR.

Acknowledgements

I sincerely thank Dr. Karen Davis for her able guidance and valuable advice throughout my thesis work. I appreciate all the time she invested in our discussions. Working with her, I learnt a lot about my area, approach to research and writing style.

I thank my Dad, Mom and sisters for always encouraging me. I really enjoyed the initial literature review discussions with Xiaoming. My thanks to Srikanth for his timely advice about the work. I thank all my lab mates for making my work in the lab a real pleasure. I also thank Krishnamoorthy for his constant support.

Contents

List of Figures ..... **iii**

List of Tables ..... **v**

Chapter 1 Mobile Databases **1**

1.1 Mobile Databases 1
1.2 A Motivating Example 5
1.3 General Research Objective 7
1.4 Specific Research Objective 7
1.5 Research Methodology 7
1.6 Expected Contributions 8
1.7 Overview of the Thesis 8

Chapter 2 Multiple Query Processing **9**

2.1 Applying MQP in Mobile Databases 10
2.2 MQP Techniques 13
2.3 Multigraph Processing 14
2.4 Illustrative Example 16
2.5 Implementation 20

Chapter 3 Performance Study **24**

3.1 Overview of the System 24
3.2 Performance Model 26
3.3 Testbed 28
3.4 Design of Experiments 29
3.5 Results 31
3.5.1 Bandwidth Usage 32
3.5.2 Wait Time 34

Chapter 4 Conclusions **38**

4.1 Contributions 38
4.2 Future Work 39

Bibliography **40**

Appendix A Testbed 42

A.1 Base Relations 42

A.2 Views 42

Appendix B Results from the Experiments 47

Appendix C Graphs 54

C.1 Balanced Load 54

C.2 Skewed Load1 56

C.3 Skewed Load2 58

C.4 Skewed Load3 60

C.5 Skewed Load4 62

C.6 Distinct Query Load 64

List of Figures

Figure 1.1: Centralized Wireless Network Architecture 2
Figure 1.2: A Broadcast Disk 4
Figure 1.3: Relationships between Queries and Broadcast Data 5
Figure 1.4: Bandwidth Usage for Different Channels 6

Figure 2.1: Two Sample Queries 9
Figure 2.2: Information Flow in PR 11
Figure 2.3: Information Flow in MQPR 12
Figure 2.4: Common Subexpression Processing Algorithm 15
Figure 2.5: Heuristics for Edge Processing 15
Figure 2.6: Algorithm for Identical Select Processing 16
Figure 2.7: An Example 17
Figure 2.8: Base Relations 17
Figure 2.9: Initial Query Graph for Query A 18
Figure 2.10: Initial Query Graph for Query B 18
Figure 2.11: Initial Multigraph for Queries A and B 18
Figure 2.12: Transformation Steps in Processing Multigraph for Queries A and B 19
Figure 2.13: Data Structure for Representing a Multigraph 20
Figure 2.14: Multigraph for Queries A and B 21
Figure 2.15: Data Structure Representation of Multigraph for Queries A and B 22

Figure 3.1: Phases in MQPR 25
Figure 3.2: Channel Assignment and Wait Time Calculation Algorithm 27
Figure 3.3: Relative Query Sizes 28
Figure 3.4: Subsumption Relationships between Queries 29
Figure 3.5: Query Loads 30
Figure 3.6: Fixed System Parameters 31
Figure 3.7: Number of Channels for Broadcast 31
Figure 3.8: Bandwidth Usage 33
Figure 3.9: Percentage Savings in MQPR Compared to PR 33
Figure 3.10: Percentage Savings of Bandwidth in MQPR Compared to PR 34
Figure 3.11: Average Wait Time with 5 Channels 35
Figure 3.12: Average Wait Time with 3 Channels 35
Figure 3.13: Average Wait Times for up to 30 Requests 36
Figure 3.14: Average Wait Times for up to 50 Requests 36
Figure 3.15: Percentage Reduction in Average Wait Time using $MQPR_2$ Compared to PR 37

Figure A.1: Testbed Schema 42
Figure A.2: Testbed Derivation from TPC-H-SPJ Query Set 43
Figure A.3: Subsumption Relationships in the Testbed 46
Figure A.4: Query Sizes 46

Figure C.1: Bandwidth Usage for Balanced Load 54
Figure C.2: Average Wait Time with 5 Channels for Balanced Load 55
Figure C.3: Average Wait Time with 3 Channels for Balanced Load 55
Figure C.4: Bandwidth Usage for Skewed Load1 56
Figure C.5: Average Wait Time with 5 Channels for Skewed Load1 56
Figure C.6: Average Wait Time with 3 Channels for Skewed Load1 57

Figure C.7: Bandwidth Usage for Skewed Load2.....58
Figure C.8: Average Wait Time with 5 Channels for Skewed Load2 58
Figure C.9: Average Wait Time with 3 Channels for Skewed Load2 59
Figure C.10: Bandwidth Usage for Skewed Load3.....60
Figure C.11: Average Wait Time with 5 Channels for Skewed Load3 60
Figure C.12: Average Wait Time with 3 Channels for Skewed Load3 61
Figure C.13: Bandwidth Usage for Skewed Load4.....62
Figure C.14: Average Wait Time with 5 Channels for Skewed Load4 62
Figure C.15: Average Wait Time with 3 Channels for Skewed Load4 63
Figure C.16: Bandwidth Usage for Distinct Query Load.....64
Figure C.17: Average Wait Time with 5 Channels for Distinct Query Load 64
Figure C.18: Average Wait Time with 3 Channels for Distinct Query Load 65

List of Tables

Table B.1: Balanced Load 48
Table B.2: Skewed Load1 49
Table B.3: Skewed Load2 50
Table B.4: Skewed Load3 51
Table B.5: Skewed Load4 52
Table B.6: Distinct Query Load 53

Chapter 1 Mobile Databases

Wireless technology is a fast growing research and commercial area. Wireless devices such as cellular phones, personal digital assistants (PDAs) or laptop computers have become popular because they are convenient and economical. Currently there are around 200 million users of some form of wireless networks; it is anticipated that there will be around 1 billion subscribers in the next five years [UC 1999]. Mobile device users need a way to communicate with other (probably larger and more powerful) systems in order to use remote data or services. However, there is no need to establish wired communication links. Often, a wireless network can be installed in places where it is not possible to install a wireline. Users of mobile devices have immediate access to services regardless of their location. Application areas include electronic mail, field audit, public safety, stock trading, airline activities, weather information, bill paying, warehouses, healthcare and the transportation industry [MA 2000]. Most of these applications need access to databases, digital libraries, online services, and location-dependent information, which is provided by mobile databases.

1.1 Mobile Databases

The main aim of mobile databases is to provide information to a mobile user. The term ‘mobile database’ does not necessarily mean that the database is mobile. Sistla and others [SWCD 1997] propose a centralized and distributed mobile database architecture where some data is present at the central server and other data is present at mobile nodes. In a distributed architecture there is a possibility of nodes being disconnected [IB 1993, B 1999], and thus may not be available to answer a request at all times. In order to focus on query optimization issues, we assume a purely centralized architecture in our work.

In a centralized wireless architecture the whole geographical area is divided into hexagonal cells where each cell mimics a circle (shown in Figure 1.1). At the center of each cell

is a Base Station (BS) that communicates with the Mobile Stations (MS) in its cell area through a wireless link. The BS is also referred to as server and the MS as mobile units, devices, or simply mobiles. BSs serving an area are connected by a backbone wired network through a Mobile Switching Center (MSC). The MSCs are connected to the Public Switched Telephone Network (PSTN). As a mobile station moves, the calls are relayed from one cell to another. When an MS moves from one cell to another the radio link with the old BS has to be broken and a new radio link has to be established with a new BS.

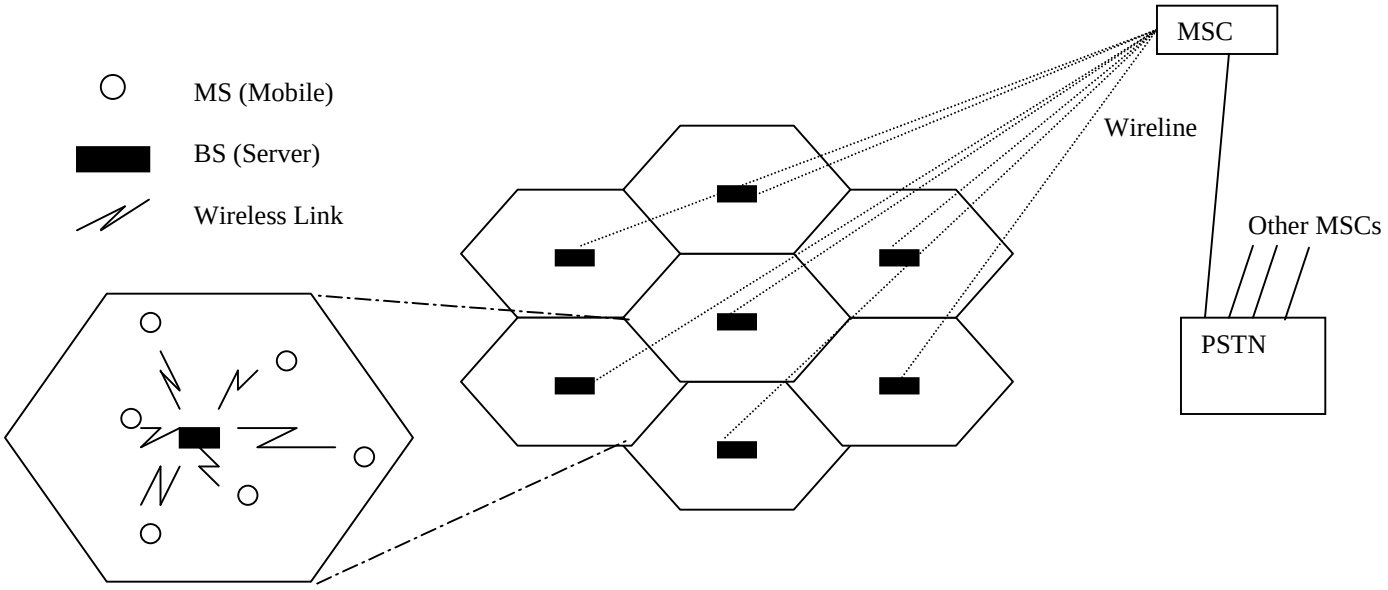


Figure 1.1: Centralized Wireless Network Architecture

In a centralized mobile database the database resides in the central server (or BS). There are two ways the server can provide data for a mobile user: pull-based and push-based. In a pull-based method the user sends a request for data on an uplink channel and the server processes the request and sends the data to the client on a downlink channel. An uplink channel is a channel on which a mobile can send its query to the server. The downlink channel or pull channel is the channel on which an answer to a query is sent to an individual mobile. Other mobiles cannot

access the downlink information. Uplink channel bandwidth is used to send queries and downlink channel bandwidth is required to send the answers to the queries. In the push-based method the server broadcasts the data on a broadcast channel and the mobiles tune to that particular channel to retrieve the information [AAFZ 1995]. A broadcast channel or push channel is a channel on which the server broadcasts information that all the mobiles can access. In this mode there is a wait for the data but there is a reduction in the channel bandwidth that is used since the data need not be sent to each client separately. In a hybrid model, the push-only model is augmented with a pull-based approach [AFZ 1997] by using an uplink channel to allow clients to send explicit requests for data to the server. This model accommodates that queries whose answers cannot be obtained from the broadcast information.

There are many limitations in using wireless devices such as frequent disconnections, limited power, limited screen size, security and authentication requirements, and limited channel bandwidth [IB 1993, AAFZ 1995]. In this thesis we address the issue of limited channel bandwidth.

In a wireless mobile network the servers have relatively high bandwidth broadcast capability while the clients can communicate only over a lower bandwidth link. Such an environment is called an asymmetric communication environment; a new architecture for this environment called broadcast disks has been proposed by Acharya and others [AAFZ 1995]. In this approach the server continuously and repeatedly broadcasts data to the clients. The broadcast channel becomes a disk from which clients can retrieve data as it goes by. The broadcast is created by assigning data items to different disks of varying sizes and speeds and then multiplexing the disks on the broadcast channel. Items stored on faster disks are broadcast more often than items on slower disks. Figure 1.2 shows a simple broadcast program in which disk pages *A*, *B*, *C*, *D*, and *E* are continuously broadcast. The mobiles that need this information tune to the channel and retrieve the required information. Lee and others give channel allocation methods for the broadcast data [LHL 1999].

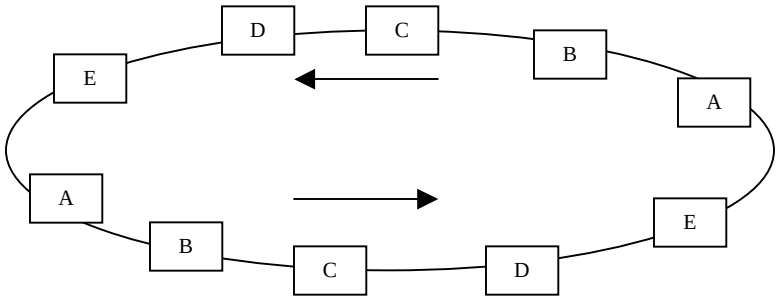


Figure 1.2: A Broadcast Disk

Unlike the push architecture, in a pull model the answers for each query are sent separately. When the queries are processed in groups there may be many common expressions among the queries. A multiple query optimizer processes a group of queries together and executes the common operations once. This technique is very useful for query workloads that contain identical queries (queries are repeated), subsumption queries (the answer for a query is a subset of the answer for another query), and overlapping queries (the two answers share some data). In the case of disjoint queries (all the queries are all distinct), this technique behaves as an ordinary query processor.

In this thesis, we apply the techniques of MQP for the queries sent as pull requests by a group of users and refer to the approach as multiple query processing for pull requests (MQPR). In the naïve approach for answering pull requests (PR), a user sends a request for data on an uplink channel, and the server processes all these requests independently and sends the answers separately on downlink channels; and this data is accessible only by a mobile that has sent a request. In MQPR, we group queries that are sent to the server in a given time window. If the queries in a particular group have some commonalities, we construct a query graph for these queries and obtain a common answer for them, called a materialized view. The answers for the queries in the group can be obtained from this materialized view. Instead of using the pull channel, which is a downlink channel, we use a push-pull channel, which is a broadcast channel for views that answer a group of pull requests. The individual mobiles are sent small packets

stating when and for how long to tune to the channel and retrieve the answers. In this method we use a broadcast channel instead of a downlink channel, and we eliminate redundancy in query answers. We study the savings in wait time and bandwidth that are obtained using this approach. In the next section, we motivate of our work with an example.

1.2 A Motivating Example

To illustrate the problem addressed in this thesis, consider a mobile database where data pages *A, B, C, D, E* and *F* are continuously broadcast over a push channel. Figure 1.3 gives the representation of the push data, while its channel bandwidth usage is given in Figure 1.4(a). The channel is allocated for the push data and the disks *A-F* continuously occupy this channel. The channel bandwidth usage for broadcasting the disks once is equal to the sum of the sizes of all the disks. It is assumed that there is no repetition of data in the disks. A time window occurs in which 5 mobiles cannot retrieve the answers to their queries (Q1-Q5) from the push data and thus they send pull requests to the server.

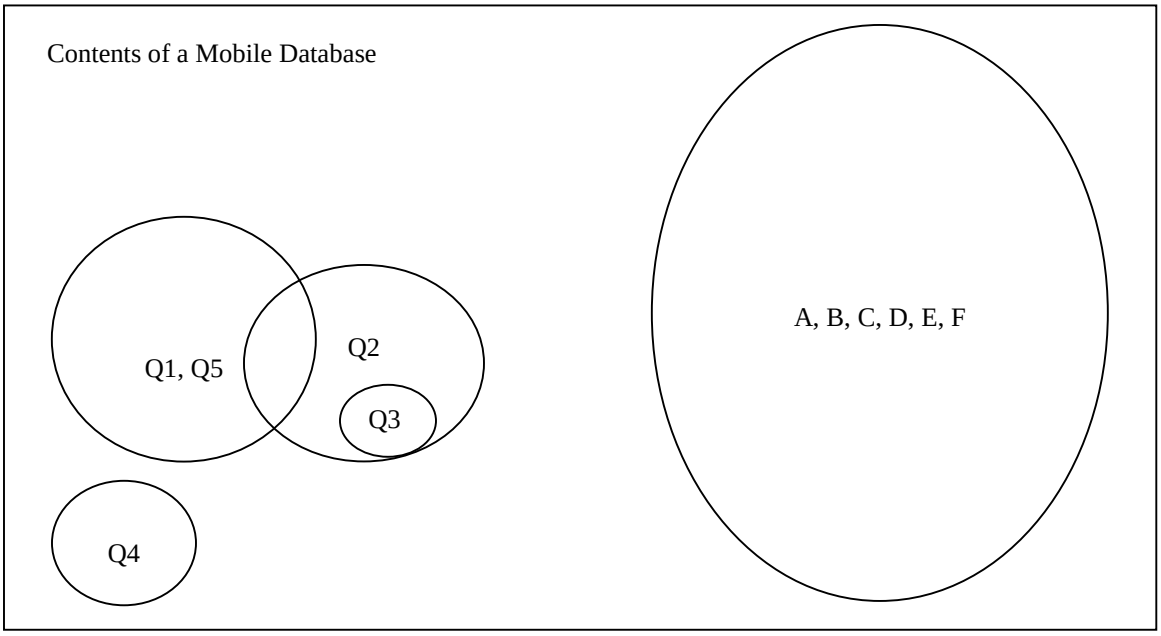
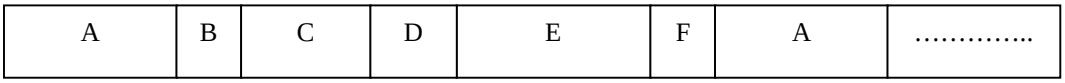


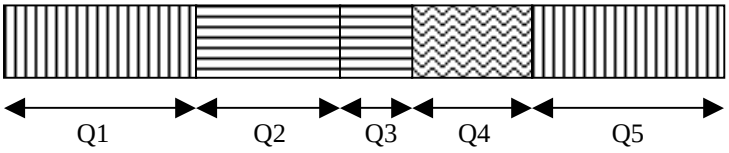
Figure 1.3: Relationships between Queries and Broadcast Data

It is assumed that the answers to Q1 and Q5 are identical, the answer to Q2 overlaps that of Q1, the answer to Q2 subsumes that of Q3 (i.e., answers for Q3 can be obtained by doing a simple select or join operation on the result of Q2), and the answer to Q4 does not have any data in common with any other query answers. The representation of these queries is given in Figure 1.3. The universe of discourse is a mobile database, and the ellipses represent subsets of the database.

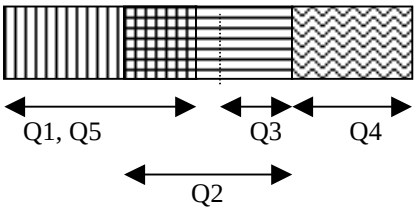
The server processes the queries and sends the answers on a downlink channel in the order the requests are sent. The channel bandwidth usage for the pull requests on a downlink channel is shown in Figure 1.4(b).



(a) Bandwidth Usage for Push Data



(b) Bandwidth Usage for Pull Requests on a Downlink Channel



(c) Bandwidth Usage for Pull Requests on a Push-Pull (Broadcast) Channel

Figure 1.4: Bandwidth Usage for Different Channels

In our approach we apply MQP, which helps us in determining the commonalities between queries in the given time window. The resulting broadcast data is shown in Figure 1.4(c).

Since Q1 and Q5 are identical, the results are broadcast just once. The overlapping portion of Q1 and Q2 is broadcast just once. The result of Q3 is obtained from the result of Q2. Q4 is broadcast. Channel bandwidth and wait time savings are expected to result from using MQP to form a broadcast program.

1.3 General Research Objective

This thesis applies techniques from multiple query optimization to obtain bandwidth and wait time reduction for answering pull requests in a mobile database environment.

1.4 Specific Research Objective

The specific research objectives are as follows:

a) To determine what techniques are proposed for processing pull requests and devise improvements.
b) To measure the effectiveness of algorithms for processing pull requests.
c) To design experiments to determine performance characteristics.
d) To analyze the results to determine conditions under which savings occur.

Our approach to meeting these objectives is given in next section.

1.5 Research Methodology

The research methodology used to achieve our research objectives is given here.

a) In order to define a new method for handling pull requests, we investigate the use of multiple query processing techniques to process a set of requests.
b) In order to determine the effectiveness of algorithms for processing pull requests, we
    i. propose characteristics such as bandwidth and wait time,
    ii. develop metrics to calculate the bandwidth and wait time, and
    iii. select a testbed to conduct the experiments.

c) To conduct experiments, we develop a software system for implementing MQP techniques that computes bandwidth and wait time.

d) To analyze the results, we examine them in the form of graphs and draw conclusions about the savings in bandwidth and also see how the wait time varies under different conditions.

1.6 Expected Contributions

By accomplishing our research objectives we expect to make the following contributions:

a) an improved method to handle the pull-requests using MQP techniques,

b) cost metrics to calculate the bandwidth and wait time and also define a testbed appropriate for MQP scenarios,

c) a software system that implements MQP and also computes bandwidth and wait time in both the methods, and

d) identification of the key factors that affect bandwidth and wait time.

1.7 Overview of the Thesis

In Chapter 2, we discuss multiple query processing, a multigraph MQP technique, and details of our software implementation. In Chapter 3, we present the overview of our complete system, the testbed, design of experiments, and the results and their analysis. In Chapter 4, we present contributions of the thesis and discuss future work.

Chapter 2 Multiple Query Processing

Multiple Query Processing (MQP) optimizes a set of queries together by executing the common operations once in order to save query execution time and evaluation cost. It has been shown by Sellis that MQP typically offers substantial improvement to the performance of a system [S 1998]. Exhaustive algorithms have been proposed for doing MQP [SG 1990]. These are impractical and explore an exponential search space. Roy and others propose heuristic algorithms that are practical and provide significant benefits in the optimization of queries [RSSB 2000]. We adopt a heuristic approach here.

```
a) SELECT customer.name, customer.custkey, orders.orderkey,
          orders.orderdate, orders.totalprice
   FROM customer, orders, lineitem
   WHERE orders.custkey = customer.custkey
         AND lineitem.orderkey = orders.orderkey
         AND lineitem.quantity = '24';

b) SELECT customer.name, customer.custkey, orders.orderkey,
          orders.orderdate, orders.totalprice
   FROM customer, orders, lineitem
   WHERE orders.custkey = customer.custkey
         AND lineitem.orderkey = orders.orderkey
         AND lineitem.quantity = '24'
         AND orders.orderstatus = 'shipping';
```

Figure 2.1: Two Sample Queries

We illustrate MQP with an example. Figure 2.1 contains two queries that retrieve information from an order processing database. The first query retrieves customer and order information for a particular quantity of items ordered. The second query retrieves customer and order information for a particular quantity of items ordered and whose order status is shipping. The answer for the second query is a subset of the results of first query. The results of the first query enable fast computation of the second. These requests can be optimized using MQP by first

finding the Customers whose *lineitem* quantity is 24 and then using this to find the Orders with the additional constraint that the *order status* is shipping.

Section 2.1 discusses how MQP techniques can be applied to mobile databases. Section 2.2 gives an overview of MQP techniques from the literature and we select one for our work; Section 2.3 details the multigraph processing algorithms we use here. Section 2.4 illustrates an approach with an example, while Section 2.5 gives the details of software implementation.

2.1 Applying MQP in Mobile Databases

In the mobile database architecture that we adopt, there are hundreds of mobile users served by a central server (BS) that receives requests on an uplink channel and sends answers on the downlink channel. In a naïve approach, when a server receives an uplink request, it processes the request and sends a request for data to the data source, and obtains the answer from it. This answer is sent to the mobile on a downlink channel. Figure 2.2 shows the information flow between a mobile, a server, and a data source in the naïve method (called PR here for pull request). The vertical lines represent the information source, server, and a mobile and the horizontal lines show the data flow. The numbers on the lines show the sequence of the flow. A dotted horizontal line shows that the link is an air link. The continuous horizontal lines show that the link is a wireline. The direction of an arrow represents the direction of the flow of data.

Significant channel bandwidth can be expended in sending query answers on the downlink channel. If there is an overlap in the results of the queries meant for different mobiles, channel bandwidth is wasted by sending the same data on the downlink more than once. Response time and channel traffic are both increased. To address these issues, we propose a multiple query processing method for pull requests (called MQPR) to batch the queries and perform common sub-expression processing.

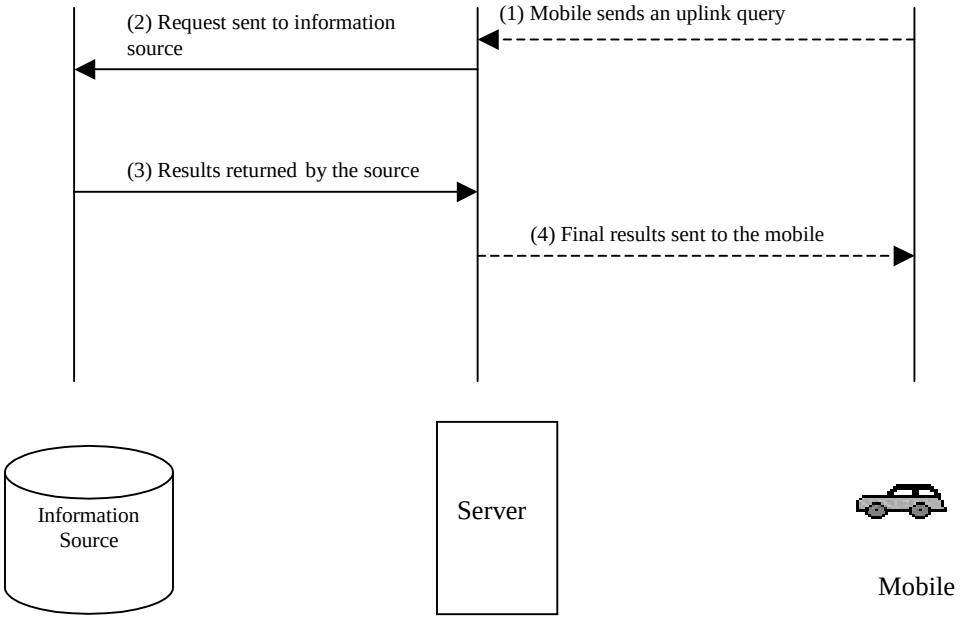


Figure 2.2: Information Flow in PR

The major tasks in MQP are common operation/sub-expression identification and global plan construction. In our approach, MQPR, we develop a view from the results of various queries and broadcast this view on a channel we call the push-pull channel. The push-pull channel broadcasts information on a broadcast channel instead of sending answers on a downlink channel. On a downlink channel, mobiles are sent a small packet containing information about when to tune to the push-pull channel and which part of the view they need to download to find the results to their queries. When there are subsumptions between queries, i.e., when the answer for a query is to be obtained by doing a select operation on the answer being broadcast, we can sort the answer before it is broadcast so that the mobile does not have to do a select. It can just tune at the time given in the packet and get its answer. This is referred as filtering of data [IVB1 1994]. This method reduces the bandwidth that would previously be wasted by transmitting almost the same information for different users separately. Figure 2.3 shows the information flow between the mobile, the server, the data source, and the broadcast channel for MQPR.

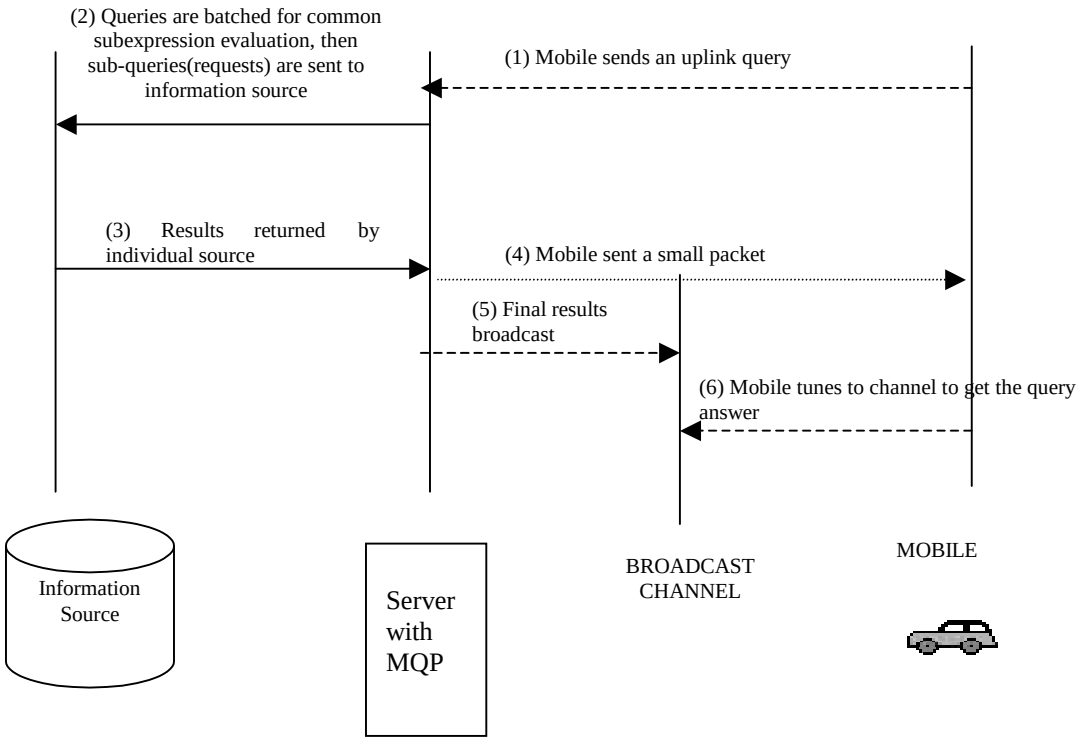


Figure 2.3: Information Flow in MQPR

In Figure 2.3, the mobile initially sends an uplink query. The server collects all the requests it obtains in a given time window, applies an MQP technique, and evaluates the common expressions. It sends the requests for data to the information source and obtains the results. The mobile is sent a small packet containing information regarding when and for how long to tune to the channel to get the information. Later these results are broadcast on a broadcast channel. The mobile then tunes to the channel according to the information in the packet and obtains the information.

In the next section, various MQP techniques from the literature are reviewed and one is selected for our work.

2.2 MQP Techniques

The problem of identifying common subexpressions is NP-hard [J 1985, RH 1980, SKL 1989]. Jarke indicates that multi-relation subexpressions can only be addressed heuristically since determining subexpressions has an exponential search space [J 1985]. There are two main heuristic approaches, one using AND/OR graphs and the other using multigraphs. In an AND/OR graph, the AND nodes represent the relations and the OR nodes represent the operators [RC 1988]. Roy and others have demonstrated that multiquery optimization using heuristics with AND/OR graphs is practical and provides significant benefits [RSSB 2000]. Their method detects subsumption by comparing each pair of operator nodes from distinct queries. There may be many AND/OR graphs corresponding to a particular set of queries; thus the performance of an algorithm depends on the chosen representation. AND/OR graphs are also procedural (i.e., they specify an evaluation order) so that some potential optimization choices are not considered. Both of these disadvantages do not occur in the multigraph approach.

We use a multigraph technique for processing subsumptions [CE 1994, CD 1998]. The multigraph approach can be used to identify common subexpressions for identical, subsumption and overlap cases. A multigraph is a non-procedural representation of multiple queries and the representation of the multiple queries is unique. We choose a multigraph for MQP in our work as it represents queries in a unique way compared to operator graphs and also because of the ease of detection of common subexpressions. The detection of common subexpressions using a multigraph is the same as the detection of common edges, i.e., edges connecting to the same node(s) on the graph. The time and space complexities for processing common subexpressions using a multigraph are much lower than those that use an operator graph [CE 1994]. In our work we consider the identical, subsumption, and disjoint cases. We describe the multigraph technique in the following sections.

2.3 Multigraph Processing

A definition and common sub-expression processing algorithms for multigraphs are given here [CE 1994]. A multigraph G(R, SE, JE) is defined as follows:

1. A node, $r \in R$, of the multigraph represents a relation or an intermediate result derived from relational operation(s).
2. A selection edge, $se_i \in SE$, loops on a node and represents a select operation on the relation. A selection edge is labeled with a query ID and selection condition(s).
3. A join edge, $je_i \in JE$, between two relations represents the join operation. A join edge is labeled with a query ID and a join condition(s).

R is a relation, SE is a select edge, JE is a join edge and QL is the query list.

A multigraph is constructed for a given set of queries. Each relation is represented as a node. A select edge is represented as a loop on a relation. A join edge is represented by a line connecting two relations. When a relational operation is evaluated, the node(s) and edge(s) related to the operation are contracted into a new node. At the end of a sequence of operations, the single remaining node represents the final result. The main idea of MQP is to execute the common operations only once. Two conditions may have nothing in common, be identical, or the result of one condition subsumes the result of other condition; we refer to these relationships as the *commonality* between two conditions. Chen's subsumption processing algorithm is shown in Figure 2.4. A group of edges and type of commonality among the edges in the multigraph are selected for processing based on the heuristics given in Figure 2.5. The type of commonality among the edges is identified and the common operations are performed by calling appropriate procedures.

Algorithm comsubproc
Input: A multigraph G(R, SE, JE)
Output: A multigraph G'(R', SE', JE') with no related conditions on same type of edges (SE or JE)
Method:
While common operations exist
Use the heuristics to select a group of edges/operations for processing;
Identify the type of edges/operations (select or join) and the type of commonality (identical, subsumption or overlap) among the operations;
Perform the common operation for this group of operations and contract the node for this operation;

Figure 2.4: Common Subexpression Processing Algorithm

Edge Processing Heuristics

1. Select a group of edges SE(u, u) such that they have identical select conditions and $QL\{JE(u, v)\} \subseteq QL\{SE(u, u)\}$;
2. Select a group of edges SE(u, u) such that they have identical select conditions and $QL\{JE(u, v)\} \not\subset QL\{SE(u, u)\}$;
3. Select a group of edges SE(u, u) such that they have subsumed select conditions and $QL\{JE(u, v)\} \subseteq QL\{SE(u, u)\}$;
4. Select a group of edges JE(u, v) such that they have identical join conditions and $QL\{JE(u, v)\} \subseteq QL\{SE(u, u)\}$ where $w \neq u$ or $w \neq v$.
5. Select a group of edges JE(u, v) such that they have subsumed join conditions and $QL\{JE(u, v)\} \subseteq QL\{SE(u, u)\}$ where $w \neq u$ or $w \neq v$.

Figure 2.5: Heuristics for Edge Processing

All six algorithms for processing the edges found with the heuristics are discussed by Chen [CE 1994]. The algorithms are for processing (1) identical selects, (2) subsumption selects, (3) overlap selects, (4) identical joins, (5) subsumption joins, and (6) overlap joins. In our work, algorithms (1), (2), (4) and (5) are implemented. Figure 2.6 describes the algorithm for processing identical select conditions. This algorithm is illustrative of other identical and subsumption processing algorithms. In the algorithm for processing identical selects all the select operations with identical common subexpressions are performed just once and a contracted node is created by removing all the identical edges representing the same common subexpressions. The algorithm in Figure 2.4 continues to find the common subexpressions and processes them until no

further common conditions exist. The output of this algorithm is a multigraph with all the common conditions removed; it is a materialized view that can answer all of the queries represented in the input multigraph.

Algorithm S_identical
Input: A multigraph
G(R, SE, JE), QL(sc, Ri) // list of query Ids and their associated identical select operation σ_{sc}(Ri);
Output: A contracted multigraph G'(R', SE', JE')
Method:
Perform σ_{sc}(Ri) and create a contracted node n for this operation;
Delete edges representing the σ_{sc}(Ri) operation;
For all selection edges se(Ri,Ri) with query ID in QL(sc,Ri) do
Move se(Ri,Ri) to se(n, n);
For all edges (t,Ri) where t ≠ Ri do
If query ID of edge (t,Ri) is in the QL(sc,Ri), change original links from (t,Ri) to (t,n);
If there are no remaining edges on node Ri, delete node Ri;

Figure 2.6: Algorithm for Identical Select Processing

In the following section we give an illustrative example of the multigraph MQP technique.

2.4 Illustrative Example

Consider the example Queries A and B shown in Figure 2.7. These queries are from an order-processing database. The base relations underlying these queries are given in Figure 2.8. The construction of the multigraph for these queries and the transformations are given here. First the base relations for these queries are identified as *Orders*, *Customers* and *Lineitem.* In the following figures *O* stands for *Orders* relation, *C* for *Customer* relation and *L* for the *Lineitem* relation. Figure 2.9 shows the representation of Query A in the form of a query graph. The nodes are the relations, the edge joining two nodes is a join edge and the dotted edge looping to a relation is a select condition. Figure 2.10 shows the query graph representation for Query B. Figure 2.11 shows the multigraph for Queries A and B.

```
A) SELECT customer.name, customer.custkey, orders.orderkey,
        orders.orderdate, orders.totalprice
    FROM customer, orders, lineitem
    WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND lineitem.quantity = '24';

B) SELECT customer.name, customer.custkey, orders.orderkey,
        orders.orderdate, orders.totalprice
    FROM customer, orders, lineitem
    WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND lineitem.quantity = '24'
        AND orders.orderstatus = 'shipping';
```

Figure 2.7: An Example

Part (**partkey,** name, mfgr, brand, type, size, container, retailprice, comment)
Supplier (**suppkey**, name, address, *nationkey*, phone, acctbal, comment)
Partsupp (**partkey**, *suppkey*, availqty, supplycost, comment)
Customer (**custkey**, name, address, *nationkey*, phone, acctbal, mktsegment, comment)
Nation (**nationkey**, name, regionkey, comment)
LineItem (*orderkey*, *partkey*, *suppkey*, linenumber, quantity, extendedprice, discount, tax, returnflag, linestatus, shipdate, commitdate, receiptdate, shipinstinct, shipmode, comment)
Region (**regionkey**, name, comment)
Orders (**orderkey**, *custkey*, orderstatus, totalprice, orderdate, orderpriority, clerk, shippriority comment)

Figure 2.8: Base Relations

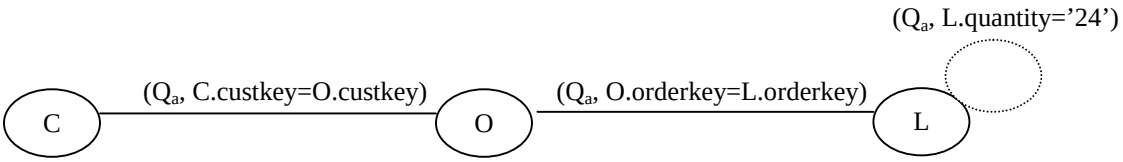


Figure 2.9: Initial Query Graph for Query A

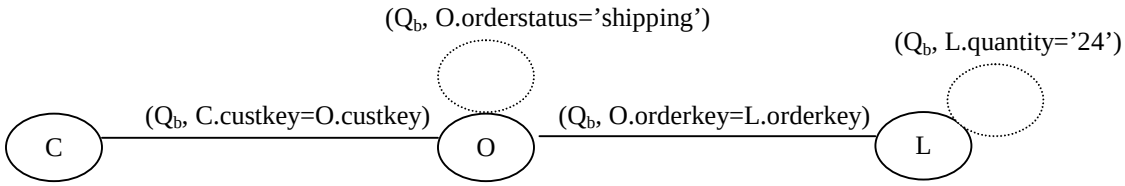


Figure 2.10: Initial Query Graph for Query B

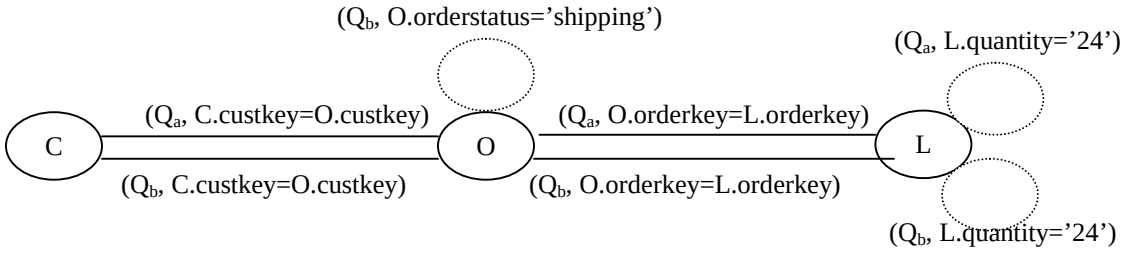
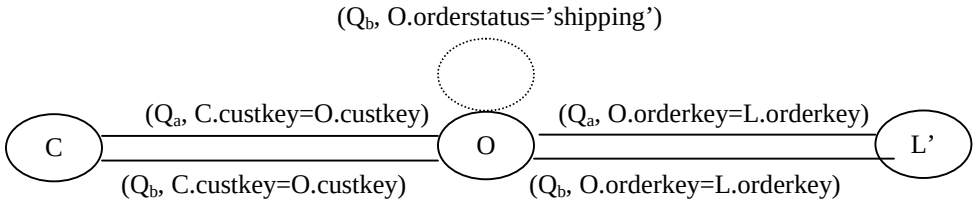
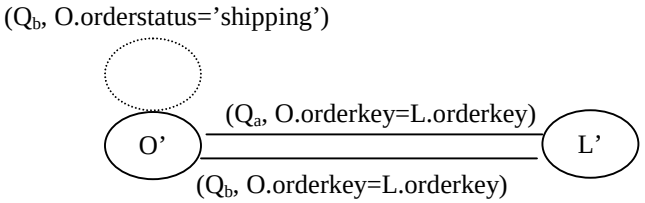


Figure 2.11: Initial Multigraph for Queries A and B

(a)



(b)



(c)

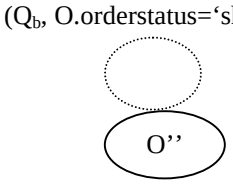
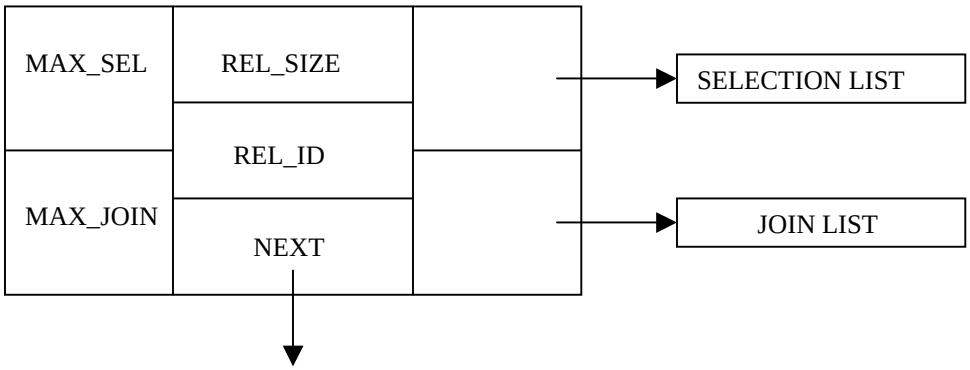


Figure 2.12: Transformation Steps in Processing Multigraph for Queries A and B

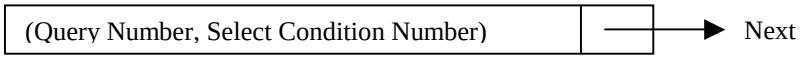
Figure 2.12 gives the transformation steps in the processing of the multigraph. In Figure 2.12(a) the select condition 'L.quantity=24' is processed and the node L is replaced by the reduced node L'. In Figure 2.12(b) the join condition 'c.custkey=o.custkey' is processed and the node O is replaced by O'. The condition 'O.orderkey=L.orderkey' is processed and the node O' is replaced by node O". The view in Figure 2.12(c) is now broadcast. The broadcast is sorted on *orderstatus* so that the mobiles that have sent Query B can get the data without doing a select operation. A small packet is sent to the mobiles that have sent Query B when they need to tune in so that they get the required information (i.e., they can tune when the information for 'orderstatus = *shipping*' is being broadcast). For any select conditions on other attributes that cannot be reduced for different queries, the mobile is sent information in order to perform a select.

2.5 Implementation

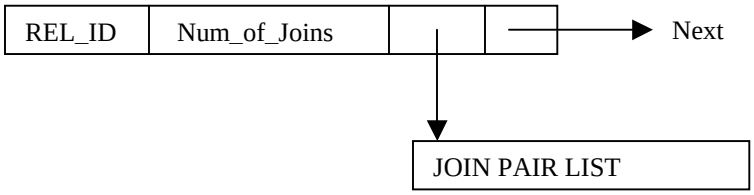
The multiquery processing algorithm is coded in C++. The algorithm takes a manually created multigraph as input. Creating a multigraph from a given query set could be automated, but we focus on the actual MQP coding rather than pre-processing.



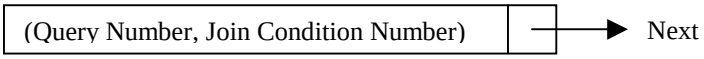
(a) A Multigraph Node



(b) A Node in the Selection List



(c) A Node in the Join List



(d) A Node in Join Pair List

Figure 2.13: Data Structure for Representing a Multigraph

Figure 2.13(a) gives the data structure for our implementation of a multigraph. The data structure is the same as that given by Chen [CE 1994] with small modifications. All the relations

have a unique relation identifier, *REL_ID*. The select and join conditions are given select condition numbers and join condition numbers, respectively. We introduce a variable *REL_SIZE* to store the size of the node and a next pointer to facilitate traversal of the multigraph. Consider a relation with *REL_ID R1*. *MAX_SEL* is the number of select conditions on node *R1*, and *MAX_JOIN* is the number of relations that *R1* is joined with. There are two pointers, one to the selection list and other to the join list. Figure 2.13(b) shows a node in a selection list. Each node in the selection list stores the query number and the selection condition number along with a pointer to the next node in the selection list. Figure 2.13(c) shows a node in the join list. Each node in the join list has the *REL_ID* of the node to which the node *R1* is joined with. If node *R1* is joined with relation *R2*, *REL_ID* is *R2* for this join node. The *Num_ of_ Joins* in the join node is the number of joins between node *R1* and node *R2*. There is a pointer to the join pair list and a pointer to the next node in the join list. Join pair list has all the join pairs between the nodes *R1* and *R2*. Figure 2.13(c) shows a node in the join tuple list. Each node in a join tuple list is a join tuple that is a query number and a join condition number.

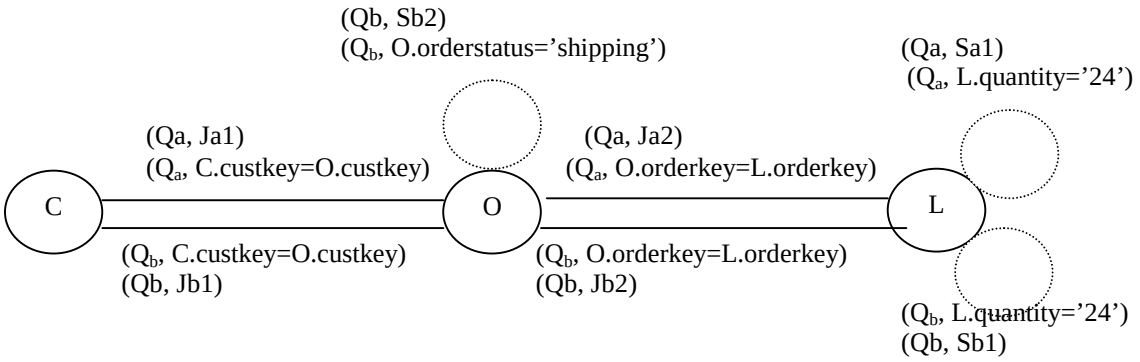


Figure 2.14: Multigraph for Queries A and B

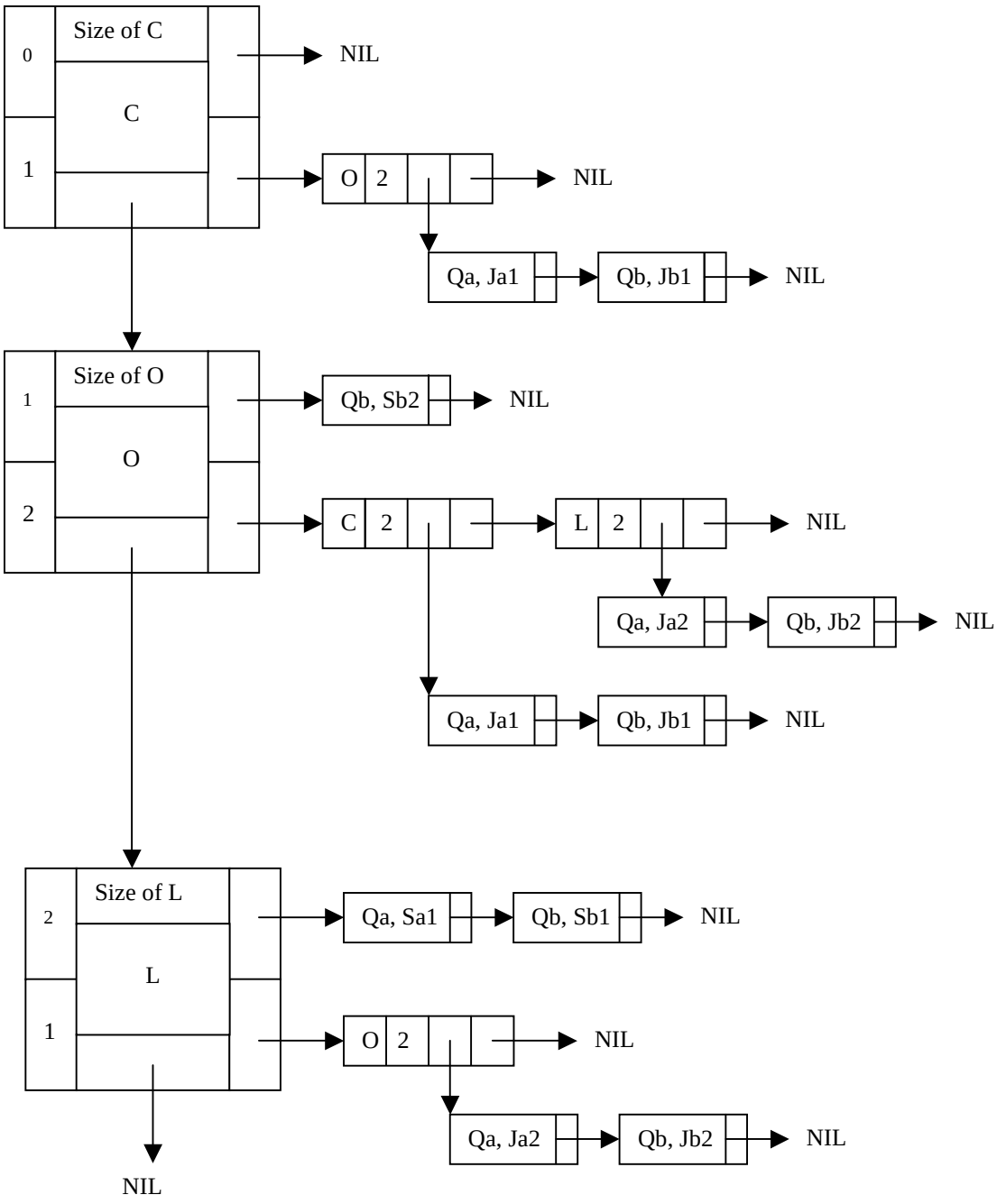


Figure 2.15: Data Structure Representation of Multigraph for Queries A and B

In Figure 2.14, we repeat the multigraph of Figure 2.11 for Queries A and B given in Section 2.4. We give selection condition numbers and join condition numbers for the selection and join conditions. The data structure representing the multigraph is given in Figure 2.15. There are three nodes representing the three relations in the multigraph. Each node has a selection list representing the select conditions on that relation and a join list representing the joins it has with other relations.

A multigraph such as that shown in Figure 2.15 is given as input to the MQPR algorithm. The algorithm uses heuristics (Figure 2.5) to select a group of edges for processing. The type of edge and commonality among the operations is identified and the common operations are performed and a contracted node is obtained. The multigraph is modified (i.e., nodes are constructed and edges are eliminated) when an operation is carried out. This processing of common edges is done until no further common operations exist. The size of the result to be broadcast for the given set of queries is computed from the final multigraph obtained after the transformations. This is used in determining the channel bandwidth and the wait time; the cost metrics for these calculations are discussed in the next chapter.

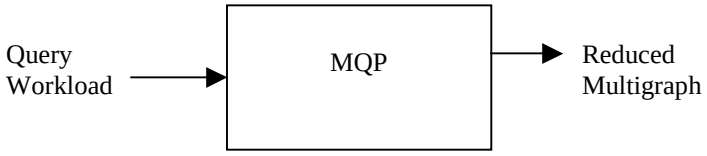
Chapter 3 Performance Study

In this chapter the performance of algorithms to process pull requests are compared. Since the algorithms determine the content of a push-pull broadcast channel, two relevant performance criteria are bandwidth usage and wait time. We design simulations with different query workloads to investigate the impact of the algorithms in different scenarios for these two metrics. Section 3.1 details the complete system. Section 3.2 gives the performance model. Section 3.3 gives the testbed we use in our experiments. Section 3.4 outlines the design of experiments and Section 3.5 gives the results and their analysis.

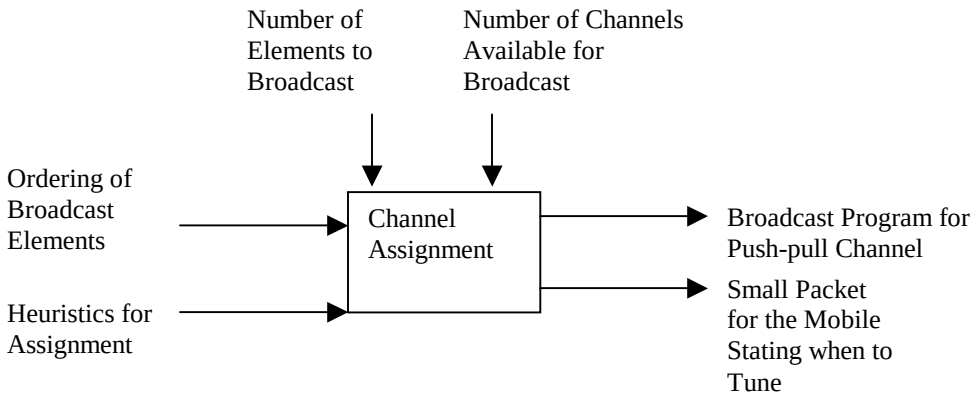
3.1 Overview of the System

There are two phases for the creation of a push-pull broadcast program. One is to determine the content (answers to queries) and the other is to assign the broadcast elements to the available push-pull channels.

In the PR method the elements to be broadcast are the answers to all the queries that the server receives. In MQPR the queries are grouped based on subsumptions and the elements to be broadcast are the answers of each of these groups. Figure 3.1(a) shows Phase 1 in MQPR. The MQP algorithms are given in Chapter 2. The query workload is a form of an unreduced multigraph. Figure 3.1(b) shows Phase 2, the channel assignment algorithm. This is the same for both PR and MQPR; only the input values change.



(a) Phase 1 in MQPR



(b) Phase 2

Figure 3.1: Phases in MQPR

In PR the ordering of the elements is the order in which the server received the queries. In MQPR we define two ways of ordering the elements: $MQPR_1$ and $MQPR_2$. In $MQPR_1$, the elements are sorted in the order of their size and the smaller elements are given higher priority for broadcast than the larger elements. $MQPR_2$ is an improved method for broadcasting the elements. In this the elements to be broadcast are sorted in increasing value of the ratio of the size of the element to the count of the number of users waiting for the element (S_i/Sc_i). The element with smallest S_i/Sc_i value has the highest priority to be broadcast. In our study, significant lowering of the average wait time is obtained using this method.

The other input for the channel assignment algorithm is the heuristics of assignment. These heuristics give us the strategy to assign elements to the channels. The heuristic we use here takes one element at a time from the ordered list and assigns it to the channel with the least sum of sizes of all the previously assigned elements.

Our channel assignment algorithm gives the broadcast program for each of the channels. The results within a broadcast element are ordered so that the subsumed queries can avoid doing

selects. Each mobile is sent a packet stating when to tune to get its answer. In the next section we give the channel assignment algorithm.

3.2 Performance Model

This section introduces terms and variables used for describing our performance model, followed by an algorithm for calculating wait time.

The broadcast is done in the form of buckets that are the smallest logical unit of broadcast. All buckets are the same size. Bucket size is given in bytes.

The parameters we use in this chapter and their explanations are as follows:

- C: channel bandwidth usage in bytes,
- N: number of requests,
- b: the size of the that needs to be broadcast for a query set in bytes,
- L: bucket length in bytes,
- B: the number of buckets to be broadcast (B = b/L),
- t: time to broadcast a single bucket,
- T: time to broadcast the given data (T = B * t),
- n: number of elements to be broadcast,
- nc: number of channels available for broadcast,
- S_i: size of the i^{th} element that is being broadcast,
- Sc_i: count of the number of users who are waiting for the i^{th} element, and
- w_i: wait time for a mobile waiting on the i^{th} element.

The total channel bandwidth is the sum of the sizes of all the elements that are broadcast. Access time is the time a mobile spends determining where in the broadcast the answer for its query can be obtained. In our system, access time is not computed because the mobile is sent a packet detailing when the mobile can tune in to retrieve the desired information. Only tuning time

is considered. Tuning time is measured in terms of the number of buckets. The wait time is the time elapsed between the times a request is sent to the time the mobile starts downloading the data. The algorithm for calculating the wait time is given in Figure 3.2. It takes as input the number of elements to be broadcast, the element sizes in a given order, and the number of available channels. It gives as output the wait time for each mobile waiting on these elements. The algorithm combines our channel assignment heuristic with the computation of individual wait times.

```
Assign_and_find
input:   n,   //number of elements that need to be broadcast
         int size[] //array having sizes of elements to be broadcast in the order
         float waittime[] //array to hold the calculated wait_times
         nc          //number of channels available for broadcast
output:  waittime[] //the wait times are returned
BEGIN
    initialize the size of each channel to zero     //channel size is the sum of elements scheduled to
                                                                    be broadcast on that channel
     for each of the elements in the array
         compute the channel with smallest size
         add this element to that particular channel, compute the new channel size
         add this element in the element list of that channel
      wait time for an element in the element list of a channel=
                                        (sum of sizes of elements before it in the element list* t/L)
return waittime[]
END
```

Figure 3.2: Channel Assignment and Wait Time Calculation Algorithm

After obtaining the wait times for each mobile from the algorithm in Figure 3.2, the average wait time is calculated as follows:

$$\text{Average wait time} = \left(\sum_{i=1}^{N} w_i\right) / \text{N}$$

The next section gives the testbed for our experiments.

3.3 Testbed

This thesis extends the TPC-H-SPJ query set [T 1999] to include additional queries with subsumption relationships. A detailed explanation of how the queries are derived and the subsumption relationships between the queries is described in Appendix A. TPC-H-SPJ contains select-project-join (SPJ) subset of the TPC-H benchmark's query set [TB 1999]. The TPC-H-SPJ query set modifies the TPC-H queries to discard the aggregation, ORDERBY and GROUPBY functions, but retains the original schema. TPC-H benchmark is a decision support benchmark that consists of a suite of business-oriented ad-hoc queries with broad industry-wide relevance. We choose this benchmark as our basis since it does not represent the activity of any particular business segment, but rather an industry that manages, sells, or distributes products worldwide.

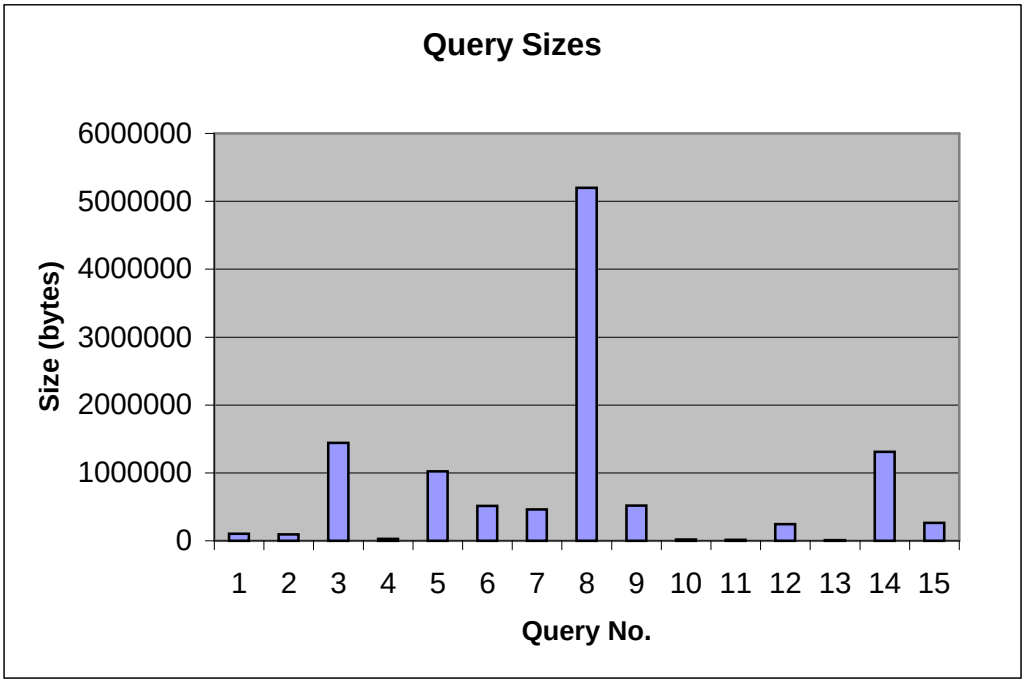


Figure 3.3: Relative Query Sizes

Our testbed extends the TPC-H-SPJ query set to a total of fifteen queries. The queries are either the same or modifications of queries in TPC-H-SPJ benchmark. We introduce subsumption

relationships in order to apply an MQP technique. A summary of the relative sizes of the queries appears in Figure 3.3. The subsumption relationships are given as a Venn diagram in Figure 3.4.

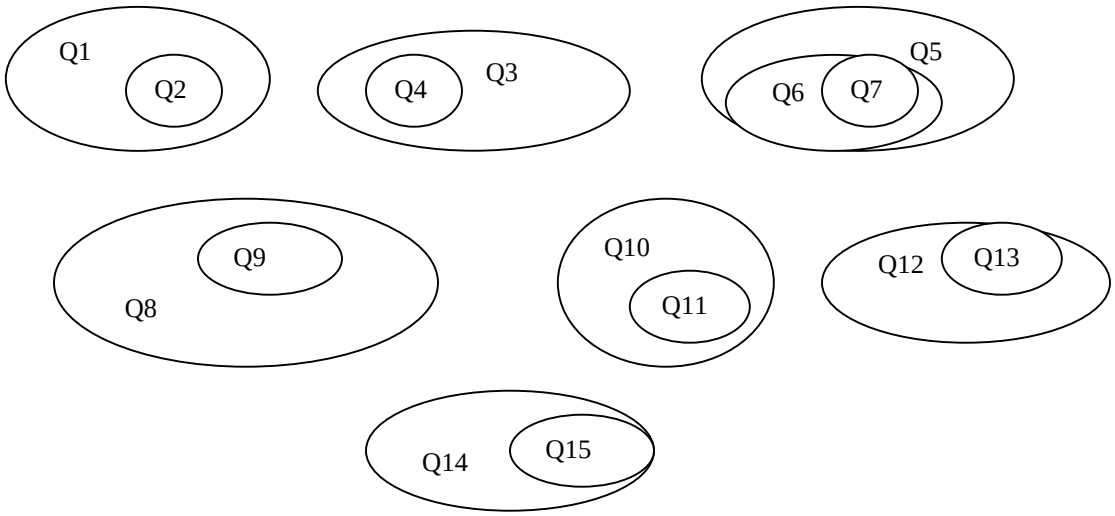


Figure 3.4: Subsumption Relationships between Queries

3.4 Design of Experiments

We design simulations to study the bandwidth savings between PR and MQPR, and study the effect on wait time for the PR, $MQPR_1$ and $MQPR_2$ algorithms for different kinds of workloads at the server. Some examples of loads that a server may have are light loads (small queries), heavy loads (large queries) and balanced loads (all types of queries). For each experiment we vary the number of user requests from one to fifty in increments of size 1. For a given number of requests, we randomly select the queries from the testbed query set.

In the first experiment, we use a balanced load, i.e., all the queries have equal probability of being selected. In the second one, termed Skewed Load1, the first k queries have higher probability of occurring than the rest. By increasing the probability of subsumption relationships between queries, we can study how this impacts bandwidth. The third load, termed Skewed Load2, k small queries (queries for which the results are small) have higher probability of occurring than other queries. This query set helps us to determine how performance

characteristics vary with a light load. In the fourth one, termed Skewed Load3, k large queries (queries for which the results are large) have higher probability of occurring and this load depicts the system performance with a heavier load. In the fifth one, termed Skewed Load4, k randomly selected queries have higher probability than the rest. This query set is used to study the system behavior for a skewed load with no particular pattern. The sixth experiment is conducted with distinct queries, called Distinct Query Load that consists of queries with no subsumption relationships between them, although some queries are repeated according to random selection. Figure 3.5 summarizes all the query loads.

Type of Load	**Load Description**	**Rationale**
Balanced	All queries have equal probability	General workload
Skewed Load1	First k queries have higher probability	Increased subsumptions
Skewed Load2	k small queries have higher probability	Lighter load
Skewed Load3	k large queries have higher probability	Heavier load
Skewed Load4	k random queries have higher probability	Random skewed load
Distinct Query Load	All the queries are disjoint queries	No subsumptions

Figure 3.5: Query Loads

For our study, we double the likelihood of a query occurring in a workload if it has a higher probability according to a particular strategy for that workload. We let k=7 for our study since it represents approximately half of the queries in the testbed. In Figure 3.6 we give the fixed parameters that are constant for all the experiments. These parameters are identical to those used in other performance studies in the literature [IVB 1994]. For each query set we vary the number of channels available for broadcast as shown in Figure 3.7. The value 10 is chosen for PR, but a smaller number of channels for MQPR is sufficient to give comparable or better performance.

Variable	Description	Value
C	Channel bandwidth	10Kbps
L	Bucket length in bytes	128 bytes
t	Time to broadcast a single bucket	0.1 sec

Figure 3.6: Fixed System Parameters

Method	Number of Channels
PR	10
$MQPR_1$	3, 5
$MQPR_2$	3, 5

Figure 3.7: Number of Channels for Broadcast

Each experiment has a given number of requests (from 1 to 50), and we group the selected queries so that queries with subsumptions fall in a particular group. We construct a multigraph for each of these groups and use it as an input to our MQP algorithm given in Section 3.1. The MQP algorithm computes the size of elements that needs to be broadcast. The elements are ordered for the different methods (PR, $MQPR_1$, $MQPR_2$), and passed to the channel assignment algorithm for computing broadcast plan. The bandwidth is computed for PR and MQPR methods. The bandwidth is same for $MQPR_1$ and $MQPR_2$, they only differ in the ordering of the elements in the broadcast. Wait times are computed for PR, $MQPR_1$ and $MQPR_2$ techniques.

3.5 Results

For different numbers of user requests, for each of the channel number variations and push-pull broadcast (program creation algorithm), the bandwidth usage and the wait times are calculated. Thus 600 simulations are conducted for bandwidth and 1,500 simulations are conducted for wait time. The complete results are tabulated in Appendix B. Representative examples are given in this section. We plot three graphs for each type of load. The first one is the bandwidth usage graph, the second graph gives the average wait time with 3 channels, and the third graph gives the

average wait time with 5 channels. The complete graphs are given in Appendix C. We introduce an example of each of these graphs and analyze the results in the following subsections.

3.5.1 Bandwidth Usage

Figure 3.8 shows the bandwidth usage graph for a balanced load. The queries are randomly selected for a given number of user requests (from 1 to 50). On the x-axis is the number of user requests and on the y-axis is the bandwidth usage in bytes. The bandwidth plot for MQPR levels out after a certain number (in this case 39) of user requests. The bandwidth plot for PR continues to increase. The bandwidth usage in MQPR is always less than or equal to that of PR. In some points the plot in PR increases but the plot of MQPR decreases (e.g., where x=18 and x=19). The reason for this is the savings due to identical queries (at x=19 there are more repetitions of certain queries than at x=18). Figure 3.9 gives the percentage savings of MQPR compared to PR. On the x-axis is the number of user requests and on the y-axis is the percentage savings in MQPR compared to PR. The findings from the graph are tabulated in Figure 3.10. We observe that significant savings result from MQPR for any type of load. The savings are lowest for queries with smaller sizes (Skewed Load2) and highest for the load with large queries (Skewed Load3).

We have also computed the bandwidth usage in the Disjoint Query Load and similar results are observed. The percentage savings are 47.35% for up to 30 requests and 59.86% for up to 50 requests. The results of this experiment depict the savings in the channel bandwidth usage when there are no subsumptions (but there are identical queries); we cannot compare the results directly to those of the other workloads since the Distinct Query Load has only 8 possible queries to choose from and the repetitions of individual queries is higher.

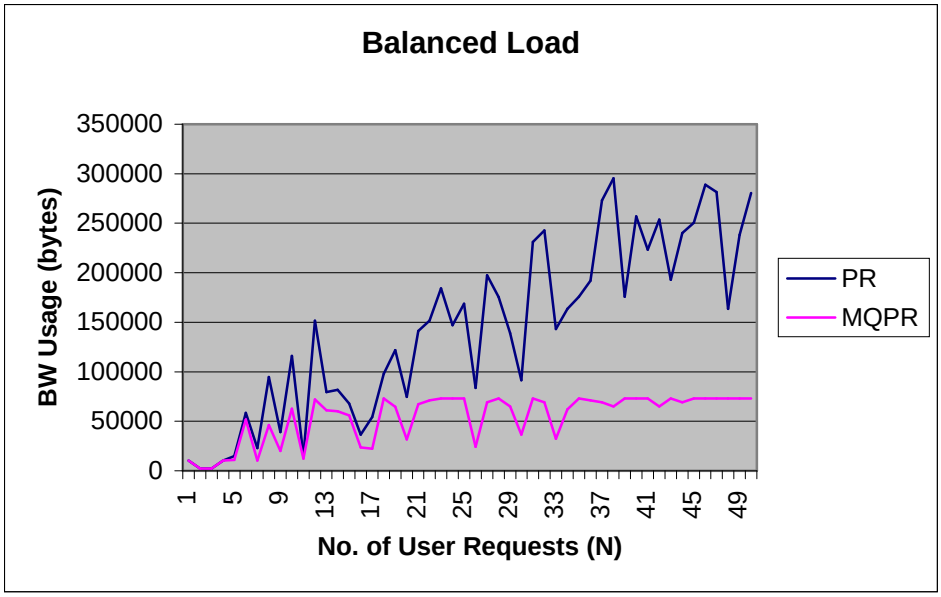


Figure 3.8: Bandwidth Usage

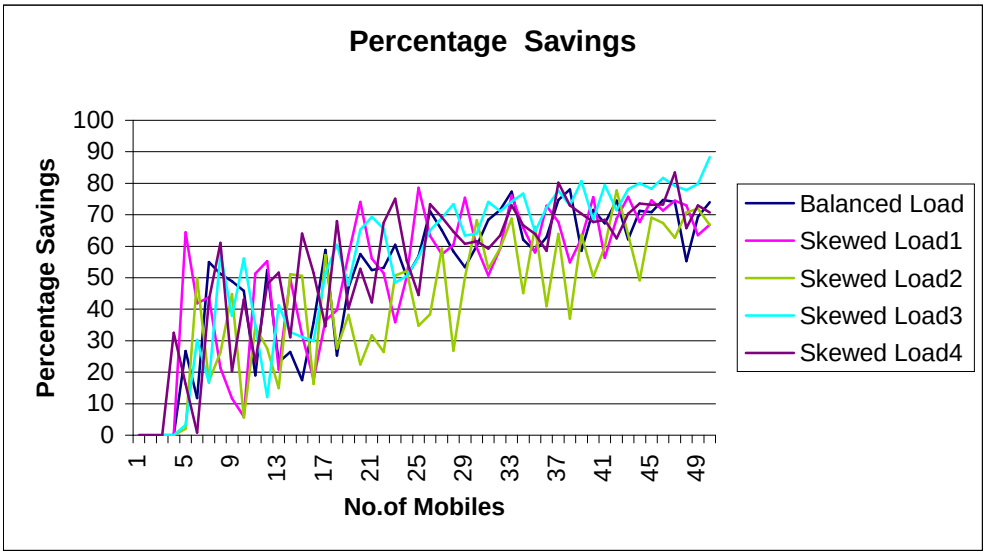


Figure 3.9: Percentage Savings in MQPR Compared to PR

Type of Load	Up to 30 requests	Up to 50 requests
Balanced	39.44%	51.21%
Skewed Load1	40.39%	50.95%
Skewed Load2	30.83%	42.63%
Skewed Load3	41.12%	55.21%
Skewed Load4	43.23%	53.74%

Figure 3.10: Percentage Savings of Bandwidth in MQPR Compared to PR

3.5.2 Wait Time

For calculating wait times, we assume that 10 channels are available for broadcast in PR. In $MQPR_1$ and $MQPR_2$, we vary the number of channels as 3 and 5. In Figure 3.11 the number of user requests is on the x-axis and the average wait time in seconds is on the y-axis. The wait times are reduced using $MQPR_1$ although just 5 channels are used for broadcast compared to 10 for PR. It is further observed that the wait times with $MQPR_2$ are slightly lower than $MQPR_1$. The average wait times in PR and $MQPR_1$ are almost the same up to 17 user requests. After that the wait time calculated for $MQPR_1$ remains almost the same but the wait time calculated for PR increases rapidly. Figure 3.12 gives the plot of average wait time. This is similar to Figure 3.11, but only 3 channels are used for broadcast in $MQPR_1$ and $MQPR_2$. Even when using just 3 channels for broadcast the average wait times are either below PR or comparable. Wait times increase up to a certain maximum value (in this case 250 seconds) for 12 requests and after that they remain almost constant using either MQPR technique.

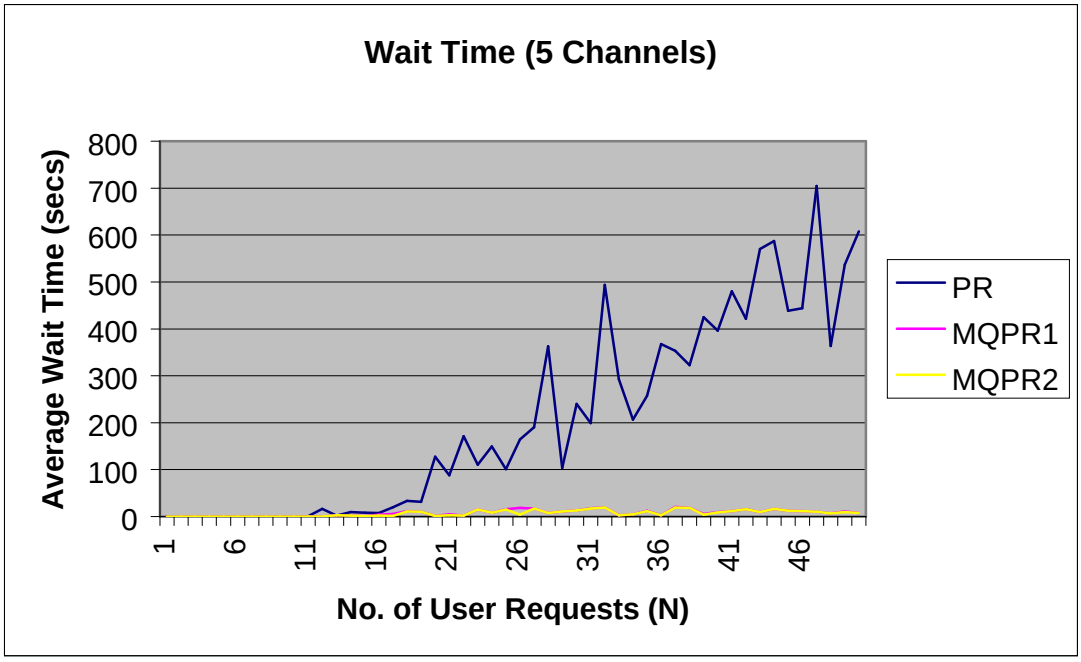


Figure 3.11: Average Wait Time with 5 Channels

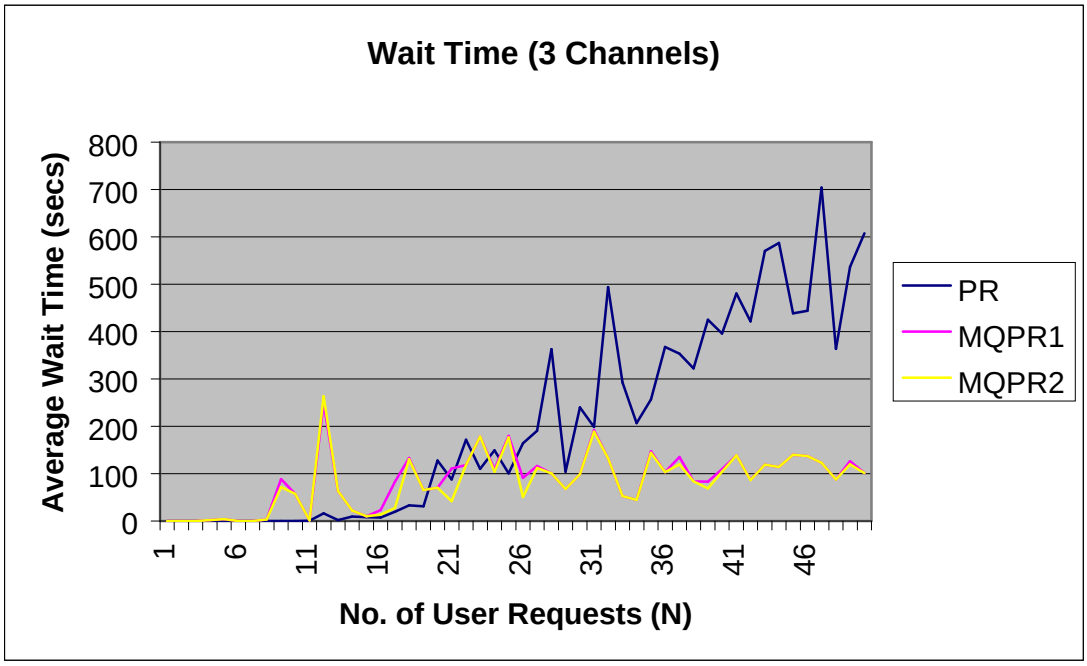


Figure 3.12: Average Wait Time with 3 Channels

Figure 3.13 compares the average wait times for the mobiles for up to 30 requests (which we refer to as Case 1). We do the comparison for varying loads, broadcast methods, and

broadcast channels. The same comparisons for up to 50 requests (which we refer to as Case 2) are given in Figure 3.14.

Type of Load	Avg Wait Time in PR with 10 Channels	Avg Wait Time in $MQPR_1$ with 5 Channels	Avg Wait Time in $MQPR_2$ with 5 Channels	Avg Wait Time in $MQPR_1$ with 3 Channels	Avg Wait Time in $MQPR_2$ with 3 Channels
Balanced	64.61	4.98	4.17	68.19	61.89
Skewed Load1	81.92	4.36	3.34	51.08	43.64
Skewed Load2	31.50	3.10	2.77	36.97	31.49
Skewed Load3	105.82	6.53	5.38	78.62	66.85
Skewed Load4	98.57	3.93	3.21	58.21	48.27

Figure 3.13: Average Wait Times for up to 30 Requests

Type of Load	Avg Wait Time in PR with 10 Channels	Avg Wait Time in $MQPR_1$ with 5 Channels	Avg Wait Time in $MQPR_2$ with 5 Channels	Avg Wait Time in $MQPR_1$ with 3 Channels	Avg Wait Time in $MQPR_2$ with 3 Channels
Balanced	208.11	7.51	6.91	86.05	81.31
Skewed Load1	221.04	7.19	5.37	76.59	64.28
Skewed Load2	121.35	4.80	4.35	52.10	47.48
Skewed Load3	346.53	9.15	8.33	118.77	110.36
Skewed Load4	262.28	5.91	4.09	78.55	63.54

Figure 3.14: Average Wait Times for up to 50 Requests

The wait times for PR are high even though it uses 10 channels. These wait times are comparable to the wait times in MQPR approaches with just 3 channels in Case 1, but they are much higher in Case 2. The wait times for $MQPR_2$ are uniformly lower than wait times calculated for $MQPR_1$. The large impact of a small increase in the number of channels using MQPR techniques is evident from the relatively lower wait times. The average wait times computed by MQPR approaches remain almost the same in Case 1 and Case 2, but the average wait times

computed by PR are higher for large number of requests than smaller number of user requests. In Case 1 the wait times calculated for Balanced and Skewed Load2 using $MQPR_1$ are slightly higher than those calculated using PR, but this is still reasonable since we are using just 3 channels in $MQPR_1$ compared to 10 in PR. $MQPR_2$ overcomes this and the wait times in these cases are almost the same as those in PR. In other words, $MQPR_2$ always outperforms $MQPR_1$ and PR in our study for average wait time calculations. Figure 3.15 gives the percentage reduction of the average wait time for $MQPR_2$ compared to PR. It can be seen that reasonably good reduction in wait time is obtained by using $MQPR_2$.

Type of Load	**Up to 30 Requests with 3 Channels**	**Up to 30 Requests with 5 Channels**		**Up to 50 Requests with 3 Channels**	**Up to 50 Requests with 5 Channels**
Balanced	4.21%	93.55%		60.93%	96.68%
Skewed Load1	46.73%	95.92%		70.92%	97.57%
Skewed Load2	0.03%	91.21%		60.87%	96.42%
Skewed Load3	36.83%	94.92%		68.15%	97.60%
Skewed Load4	51.03%	96.74%		75.77%	98.44%

Figure 3.15: Percentage Reduction in Average Wait Time using $MQPR_2$ Compared to PR

We have seen that MQPR techniques outperform PR for both bandwidth and wait time calculations. $MQPR_2$ performs well in wait time calculations. We have seen that by grouping the queries and adopting the push-based broadcast mode for pull queries, we obtain reduction in the bandwidth usage. Wait times computed by MQPR approaches are lowered in case of higher number of requests and they are comparable to the wait times computed by PR approach in case of lower number of queries.

Chapter 4 Conclusions

We present the contributions of our thesis, followed by a discussion of open research questions.

4.1 Contributions

We give below the contributions of the thesis.

- We investigate MQP techniques, adopt a multigraph technique, and use it to propose an improved method for handling pull requests that creates multiquery materialized views to be broadcast.
- We give cost metrics to calculate bandwidth usage and wait time for materialized views that are broadcast to answer queries.
- We extend the TPC-H-SPJ query set with additional subsumption queries to test MQP scenarios.
- We develop a software system to implement MQP using multigraphs.
- We conduct a performance study for different types of loads and see how the bandwidth and wait time are affected by the type of method used and the nature of loads.

In our performance study, we observe that by grouping queries and creating materialized views to broadcast, we obtain reductions in bandwidth usage and also lowering of wait times. MQPR techniques outperform PR for both bandwidth and wait time calculations in our study; $MQPR_2$ performs better than $MQPR_1$ in all cases. More savings in bandwidth usage are obtained for heavy load compared to the lighter load, the reason for this being more bandwidth is used for large size queries and by taking care of repetitions and subsumptions a lot of savings are obtained.

4.2 Future Work

In this section, we outline future problems that can be investigated in the framework presented here.

In our work we do not consider the case of queries with overlap. A performance evaluation including this type of query relationship can be done using our framework. The study can be done using the AND/OR operator graphs and applying MQP techniques suggested by Roy and others [RSSB 2000] instead of muligraphs to compare their relative merits.

Determining optimal time window is an important issue and algorithms can be developed to determine the length of time window for which the server needs to wait. In our work we assume a centralized architecture but a distributed architecture is a more realistic one and work can be done on how different requests can be handled in this environment.

Algorithms can be developed to study the frequency of pull requests and determine what to broadcast on the push channels to reduce the number of pull requests. Dynamic broadcasting techniques can be developed and broadcast disks can be reorganized so that the number of pull requests remains low.

Subieta proposes a method based on stored queries in a cache, which is a <query, response> pair [SR 1987]. When a collection of stored queries is available, responses to some queries may be obtained locally. In the mobile environment algorithms can be developed for updating the views stored in the cache so that the cache can answer future queries. This may reduce the number of pull requests that a mobile needs to send to a remote server.

Bibliography

[AAFZ 1995] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. "Broadcast Disks: Data Management for Asymmetric Communication Environments." *Proceedings of ACM SIGMOD Conference*, San Jose, CA, May 1995.

[AFZ 1997] S. Acharya, M. Franklin, and S. Zdonik. "Balancing Push and Pull for Data Broadcast." *Proceedings of ACM SIGMOD Conference*, Phoenix, AZ, May 1997.

[B1999] D. Barbara. "Mobile Computing and Databases - A Survey." *IEEE Transactions on Knowledge and Data Engineering*, Vol 11, No. 1, pp. 108-117, Jan/Feb 1999.

[CD 1998] F. Chen and M.H. Dunham. "Common Sub-expression Processing in Multiple-Query Processing." *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 3, pp. 493-499, May/June 1998.

[CE 1994] F. Chen and M.H. Eich. "Decomposition and Common Subexpression Processing in Multiple-Query Processing." *Southern Methodist Univ. Technical Report 94-CSE-30*, Aug 1994.

[IB 1993] T. Imielinski and B.R. Badrinath. "Data Management for Mobile Computing." *SIGMOD RECORD*, Vol. 22, No. 1, pp. 34-39, 1993.

[IVB 1994] T. Imielinski, S. Viswanathan and B.R. Badrinath. "Energy Efficient Indexing on Air." *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pp. 25-36, Minneapolis, MN, USA, May 1994.

[IVB1 1994] T. Imielinski, S. Viswanathan and B.R. Badrinath. "Power Efficient Filtering of Data on the Air." *Proceedings of the EDBT Conference*, Cambridge, UK, March 1994.

[J 1985] M. Jarke. "Common Subexpression Isolation in Multiple Query Optimization." *Query Processing in Database Systems*, pp. 191-205, Springer Verlag, New York, 1985.

[LHL 1999] W. Lee, Q. Hu and D.L. Lee. "A Study on Channel Allocation for Data Dissemination in Mobile Computing Environments." *Mobile Networks and Applications*, Vol. 4, No. 2, pp. 117-129, 1999.

[MA 2000] R. Malladi, and D.P. Agrawal. "Wireless and Mobile Networks: Advances and Challenges." *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI2000) and the 6th International Conference on Information Systems Analysis and Synthesis (ISAS2000)*, pp. 218-223, Orlando, FL, USA, July 2000.

[RC 1988] A. Rosenthal and U.S. Chakravarthy. "Anatomy of a Modular Multiplier Query Optimizer." *Proceedings of the 14th Conference on VLDB*, pp. 230-239, Los Angeles, CA, USA, 1988.

[RH 1980] D.J. Rosenkrantz and H.B. Hunt. "Processing Conjunctive Predicates and Queries." *IEEE International Conference on Data Engineering*, pp. 64-72, Quebec, Canada, 1980.

[RSSB 2000] P. Roy, S. Seshadri, S. Sundarshan and S. Bhobe. "Efficient and Extensible Algorithms for Multi Query Optimization." *ACM SIGMOD Intl. Conference on Management of Data*, pp. 249-260, Dallas, TX, USA, 2000.

[S 1998] T.K. Sellis. "Multiple-Query Optimization." *ACM Transactions on Database Systems*, Vol. 13, No. 1, pp. 23-52, Mar 1988.

[SG 1990] T. Sellis and S. Ghosh. "On the Multi-Query Optimization Problem." *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 2, pp. 262-266, Jun 1990.

[SKL 1989] X. Sun, N. Kamel, and L.M. Li. "Solving Implication Problems in Database Applications." *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 185-192, Portland, Oregon, 1989.

[SR 1987] K. Subieta and W. Rzeczkowski. "Query Optimization by Stored Queries." *Proceedings of the 13th VLDB Conference*, pp. 369-380, Brighton, England, 1987.

[SWCD 1997] P. Sistla, O. Wolfson, S. Chanberlain and S. Dao. "Modeling and Querying Moving Objects." *Proceedings of the International Conference on Data Engineering*, pp. 422-432, Birmingham, UK, 1997.

[T 1999] S. Turlapaty. "Performance Analysis of Self-Maintainable Data Warehousing Algorithms." *Master's Thesis*, ECECS Department, University of Cincinnati, 1999.

[TB 1999] Transaction Processing Performance Council (TPC). "*TPC Benchmark-H.*" http://www.tpc.org.

[UC 1999] "Future Directions in Mobile Computing and Networking Systems," NSF Workshop, University of Cincinnati, Jun 1999, http://www.ececs.uc.edu/~dpa/.

Appendix A Testbed

This appendix gives the base relations and the database schema that are used in the experiments.

A.1 Base Relations

The testbed schema shown in Figure A.1 has eight relations [TB 1999]. The base relations along with the attributes of each where the primary keys are in **bold** and the foreign keys are in *italics.*

Part (**partkey,** name, mfgr, brand, type, size, container, retailprice, comment)
Supplier (**suppkey**, name, address, *nationkey*, phone, acctbal, comment)
Partsupp (**partkey**, *suppkey*, availqty, supplycost, comment)
Customer (**custkey**, name, address, *nationkey*, phone, acctbal, mktsegment, comment)
Nation (**nationkey**, name, regionkey, comment)
LineItem (*orderkey*, *partkey*, *suppkey*, linenumber, quantity, extendedprice, discount, tax, returnflag, linestatus, shipdate, commitdate, receiptdate, shipinstinct, shipmode, comment)
Region (**regionkey**, name, comment)
Orders (**orderkey**, *custkey*, orderstatus, totalprice, orderdate, orderpriority, clerk, shippriority comment)

Figure A.1: Testbed Schema

In the next section, the adaptation of TPC-H-SPJ benchmark and the definitions of the selected queries that are used as materialized views in this thesis are given.

A.2 Views

We present the definitions of fifteen queries we selected and modified from the TPC-H-SPJ benchmark. The fifteen queries that are used as materialized views are presented in SQL. We give below the queries we considered in our test bed. The explanation of how these queries are derived from the TPC-H-SPJ [T 1999] query set is given in Figure A.2.

Testbed	TPC-H-SPJ
Q1	Same as Q2
Q2	Select clause added to Q2
Q3	Select clause removed from Q3
Q4	Same as Q3
Q5	Same as Q4
Q6	Select clause added to Q4
Q7	Select clauses added to Q4
Q8	Same as Q5
Q9	Select clause added to Q5
Q10	Select clause removed from Q6
Q11	Same as Q6
Q12	Same as Q7
Q13	Select clause added to Q7
Q14	Select clause removed from Q8
Q15	Same as Q8

Figure A.2: Testbed Derivation from TPC-H-SPJ Query Set

```
Q1)  CREATE VIEW extended_price AS
       SELECT part.type, lineitem.extendedprice
       FROM lineitem, part
       WHERE lineitem.partkey = part.partkey
             AND lineitem.shipdate = '1995-09-01'


Q2)  CREATE VIEW extended_price1 AS
       SELECT part.type, lineitem.extendedprice
       FROM lineitem, part
       WHERE lineitem.partkey = part.partkey
             AND lineitem.shipdate = '1995-09-01'
             AND part.retailprice > 50,000


Q3)  CREATE VIEW parts_supplier_relationship AS
       SELECT part.brand, part.type, part.size
       FROM PartSupp, part
       WHERE PartSupp.partkey = part.partkey
             AND part.brand = 'brandno45'


Q4)  CREATE VIEW parts_supplier_relationship1 AS
       SELECT part.brand, part.type, part.size
       FROM PartSupp, part
       WHERE PartSupp.partkey = part.partkey
             AND part.brand = 'brandno45'
             AND part.size = '45'


Q5)  CREATE VIEW important_stock_identification AS
```

```
SELECT PartSupp.partkey, PartSupp.supplycost, PartSupp.availqty
FROM PartSupp, supplier, nation
WHERE PartSupp.suppkey = supplier.suppkey
        AND supplier.nationkey = nation.nationkey
        AND nation.name = 'germany'
```

Q6)
```
CREATE VIEW important_stock_identification1 AS
SELECT PartSupp.partkey, PartSupp.supplycost, PartSupp.availqty
FROM PartSupp, supplier, nation
WHERE PartSupp.suppkey = supplier.suppkey
        AND supplier.nationkey = nation.nationkey
        AND nation.name = 'germany'
        AND supplycost > 12,000
```

Q7)
```
CREATE VIEW important_stock_identification2 AS
SELECT PartSupp.partkey, PartSupp.supplycost, PartSupp.availqty
FROM PartSupp, supplier, nation
WHERE PartSupp.suppkey = supplier.suppkey
        AND supplier.nationkey = nation.nationkey
        AND nation.name = 'germany'
        AND supplycost > 12,000
        AND availqty > 10
```

Q8)
```
CREATE VIEW large_volume_customer AS
SELECT customer.name, customer.custkey, orders.orderkey,
orders.orderdate, orders.totalprice
FROM customer, orders, lineitem
WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND lineitem.quantity = '24'
```

Q9)
```
CREATE VIEW large_volume_customer1 AS
SELECT customer.name, customer.custkey, orders.orderkey,
orders.orderdate, orders.totalprice
FROM customer, orders, lineitem
WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND lineitem.quantity = '24'
        AND orders.orderstatus = ‘shipping’
```

Q10)
```
CREATE VIEW shipping_priority AS
SELECT lineitem.orderkey, orders.orderdate, orders.shippriority
FROM customer, orders, lineitem
WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND orders.orderdate = '1995-03-15'
```

```
        AND lineitem.shipdate = '1995-03-15'
```

Q11)
```
CREATE VIEW shipping_priority1 AS
 SELECT lineitem.orderkey, orders.orderdate, orders.shippriority
  FROM customer, orders, lineitem
  WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND orders.orderdate = '1995-03-15'
        AND lineitem.shipdate = '1995-03-15'
        AND customer.mktsegment = 'building'
```

Q12)
```
CREATE VIEW returned_item_reporting AS
 SELECT customer.custkey, customer.name, customer.acctbal,
        nation.name, customer.address, customer.phone
 FROM customer, orders, lineitem, nation
 WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND customer.nationkey = nation.nationkey
        AND orders.orderdate = '1994-10-01'
```

Q13)
```
CREATE VIEW returned_item_reporting1 AS
 SELECT customer.custkey, customer.name, customer.acctbal,
        nation.name, customer.address, customer.phone
 FROM customer, orders, lineitem, nation
 WHERE orders.custkey = customer.custkey
        AND lineitem.orderkey = orders.orderkey
        AND customer.nationkey = nation.nationkey
        AND orders.orderdate = '1994-10-01'
        AND nation.name=’canada’
```

Q14)
```
CREATE VIEW cost_supplier AS
 SELECT supplier.acctbal, supplier.name, nation.name, part.partkey, part.mfgr
        supplier.address, supplier.phone, supplier.comment
 FROM part, supplier, PartSupp, nation, region
 WHERE PartSupp.partkey = part.partkey
        AND PartSupp.suppkey = supplier.suppkey
        AND supplier.nationkey = nation.nationkey
        AND nation.regionkey = region.regionkey
        AND part.size=’15’
```

Q15)
```
CREATE VIEW cost_supplier1 AS
 SELECT supplier.acctbal, supplier.name, nation.name, part.partkey, part.mfgr
        supplier.address, supplier.phone, supplier.comment
 FROM part, supplier, PartSupp, nation, region
 WHERE PartSupp.partkey = part.partkey
        AND PartSupp.suppkey = supplier.suppkey
```

```
AND supplier.nationkey = nation.nationkey
AND nation.regionkey = region.regionkey
AND  part.size=’15’
AND region.name = '  europe’
```

Figure A.3 gives the subsumptions introduced in the testbed query set.

Query	Subsumes
1	2
3	4
5	6,7
6	7
8	9
10	11
12	13
14	15

Figure A.3: Subsumption Relationships in the Testbed

Figure A.4 gives the sizes of the queries.

Query	Size in Bytes
Q1	105,035
Q2	94,532
Q3	1,440,000
Q4	28,800
Q5	1,024,000
Q6	512,000
Q7	460,800
Q8	5,201,053
Q9	520,106
Q10	18,000
Q11	12,000
Q12	244,122
Q13	9,765
Q14	1,312,000
Q15	262,400

Figure A.4: Query Sizes

The ordering of the queries computed in decreasing order of their final sizes is as follows:

Q8, Q3, Q14, Q5, Q9, Q6, Q7, Q15, Q12, Q1, Q2, Q4, Q10, Q11, Q13.

The distinct query load considered includes the following queries:

Q1, Q3, Q5, Q6, Q8, Q10, Q12, Q14.

These queries do not have any subsumption relationships between them.

Appendix B Results from the Experiments

The results of all the experiments are given in the following tables for different kinds of loads. Table B.1 gives the results for the Balanced Load. Table B.2, Table B.3, Table B.4 and Table B.5 give the results for Skewed Load1, Skewed Load2, Skewed Load3 and Skewed Load4, respectively. Table B.6 gives the results for the Distinct Query Load. The description of these loads is given in Chapter 3. The following are the notations used in the tables.

- N Number of queries
- (1- 15) Frequencies of queries 1-15
- BW(N) Total bandwidth used in PR (in bytes)
- BW(M) Total bandwidth used in MQPR (in bytes)
- Wait(N)10 Average wait time in PR (in secs)
- Wait(M)5 Average wait time in $MQPR_1$ using 5 channels (in secs)
- Wait(NM)5 Average wait time in $MQPR_2$ using 5 channels (in secs)
- Wait(M)3 Average wait time in $MQPR_1$ using 3 channels (in secs)
- Wait(NM)3 Average wait time in $MQPR_2$ using 3 channels (in secs)

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	BW(N)	BW(M)	Wait(N)10	Wait(M)5	Wait(NM)5	Wait(M)3	Wait(NM)3
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	10250	10250	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	2126	2126	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	2272	2272	0	0	0	0	0
4	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1	10368	10368	0	0	0	2.34375	2.34375
5	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	14964	10964	0	0	0	3.75	3.75
6	2	0	0	0	0	0	0	1	1	0	0	0	0	1	1	58637	51703	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	3	1	0	0	2	1	23065	10390	0	0	0	0	0
8	0	0	0	0	0	0	1	2	1	0	1	3	0	0	0	94745	46234	0	0	0	3.51562	3.51562
9	0	1	2	1	0	2	1	0	0	0	0	1	0	0	1	39020	19945	0	0	0	88.1911	71.7793
10	1	1	1	1	2	0	1	2	0	0	0	0	0	0	1	115950	62753	0	0	0	57.4117	57.4117
11	3	1	1	1	0	0	0	0	0	1	3	0	1	0	0	15173	12287	0.69354	0	0	1.38707	1.38707
12	0	0	3	0	1	0	0	2	1	0	0	1	1	2	1	151613	72040	16.5291	0	0	247.68	263.573
13	0	0	0	1	1	1	0	1	3	0	2	1	1	1	1	79519	61109	2.02915	2.88462	2.88462	63.587	63.587
14	1	0	0	1	1	1	2	1	1	2	0	0	0	1	3	81623	60069	9.4774	2.00893	2.00893	22.1691	22.1691
15	1	2	0	4	0	2	1	1	0	0	2	0	0	1	1	67918	56022	8.125	0.625	0.625	10.3456	10.3456
16	2	0	0	2	1	1	1	0	1	3	0	0	2	1	2	36678	23575	7.75854	4.06714	2.30933	22.8677	13.4146
17	2	1	3	3	0	1	2	0	1	1	1	0	1	0	1	54428	22400	19.7195	5.41199	1.27597	82.8028	28.0223
18	2	0	2	1	1	1	0	1	1	3	1	2	1	1	1	97769	73001	33.4213	11.4614	10.6801	132.918	130.659
19	0	4	1	2	2	0	1	2	0	1	1	2	2	0	1	121771	64719	31.3426	9.99441	9.99441	65.5741	65.5741
20	0	0	2	0	0	2	3	0	2	1	1	1	5	2	1	74499	31611	127.847	1.40625	1.40625	70.0143	70.0143
21	1	3	2	1	0	1	2	2	3	1	3	0	1	1	0	141165	67170	87.6975	4.43806	2.48605	110.537	42.0474
22	0	2	1	0	1	5	3	2	1	1	1	0	2	1	2	151593	71088	171.349	2.26438	2.26438	118.527	118.527
23	0	2	1	2	1	3	0	3	1	3	0	1	3	2	1	184248	72919	110.327	14.6783	14.6783	178.531	178.531
24	1	2	2	0	2	0	1	1	1	2	1	1	3	5	2	146955	73001	149.447	8.01009	8.01009	109.423	103.757
25	2	1	3	2	1	2	0	2	2	1	3	2	1	2	1	168835	73001	101.002	15.9419	14.8169	179.929	176.675
26	0	1	3	3	0	4	5	0	2	2	2	2	1	0	1	83699	24149	164.218	18.1248	3.92224	91.3291	50.8845
27	4	1	2	2	0	3	1	3	2	2	1	1	2	2	1	197582	69001	190.453	17.2794	16.7585	116.134	112.11
28	2	0	3	0	3	1	5	1	1	1	1	2	2	4	2	175389	73001	362.804	7.36802	7.36802	100.686	100.686
29	1	2	2	3	1	1	2	2	1	5	2	3	3	0	1	138893	64801	104.006	10.9134	10.9134	67.3478	67.3478
30	5	1	2	2	1	1	4	0	2	2	2	2	2	2	2	91353	36431	240.168	12.8161	12.8161	98.8313	98.8313
31	3	3	1	3	2	0	2	4	2	3	2	3	0	1	2	231142	73001	198.813	17.6968	17.2432	191.925	188.42
32	2	3	2	1	0	2	0	3	4	2	0	1	3	6	3	242802	69001	494.005	19.2687	19.2687	132.414	132.414
33	1	3	4	2	2	2	3	0	0	3	4	3	0	5	1	143104	32368	292.784	2.55682	2.55682	52.5791	52.5791
34	2	3	0	2	2	2	4	2	3	4	2	2	3	2	1	163506	61976	206.831	4.54963	4.54963	44.7177	44.7177
35	2	3	2	5	1	4	0	2	2	2	2	3	1	2	4	175841	73001	256.835	12.1906	11.7888	147.256	144.151
36	1	3	2	2	5	2	2	2	2	2	5	0	2	1	5	191981	71170	367.64	2.41016	2.41016	103.105	103.105
37	4	0	4	3	0	3	4	4	4	2	3	2	1	1	2	272946	69001	353.333	20.4029	18.8826	134.923	119.755
38	3	2	2	1	1	6	3	5	3	1	3	2	3	0	3	295435	64801	322.457	18.3857	18.3857	83.8798	83.8798
39	2	4	4	6	4	1	3	1	0	1	5	1	2	3	2	175897	73001	424.955	5.70984	3.90695	83.181	69.25
40	4	3	3	1	5	1	7	3	1	1	3	5	1	1	1	256970	73001	396.25	9.61211	8.90898	109.152	103.718
41	2	2	2	3	2	2	2	2	3	4	4	4	2	6	1	223218	73001	480.186	11.7221	11.7221	138.602	138.602
42	2	4	4	2	2	6	3	3	4	2	1	2	2	0	5	253590	64801	421.653	15.6854	15.6854	86.9218	86.2522
43	3	2	2	4	3	4	2	1	3	2	3	5	2	5	2	192963	73001	569.974	9.59557	9.59557	118.641	118.641
44	2	1	2	3	0	3	4	3	5	2	5	2	4	3	5	240040	69001	587.302	16.5178	16.5178	114.114	114.114
45	3	3	2	2	4	4	2	3	3	2	5	2	4	2	4	250486	73001	438.759	12.1911	12.1911	139.561	139.561
46	2	5	3	1	2	2	3	4	2	8	0	2	5	3	4	289039	73001	443.91	11.9261	11.9261	137.394	137.394
47	1	6	4	1	4	3	6	3	2	2	2	2	4	2	5	281441	73001	704.185	10.2256	10.2256	123.003	123.003
48	2	4	3	3	2	3	4	1	2	4	6	6	2	1	5	163400	73001	363.499	6.88647	6.88647	87.613	87.613
49	3	5	3	5	2	8	3	2	3	1	1	5	3	3	2	238135	73001	536.404	10.6692	9.80828	126.31	119.657
50	6	2	3	1	5	2	4	3	1	1	4	2	5	3	8	280600	73001	607.704	7.68969	7.68969	101.529	101.529

Table B.1: Balanced Load

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	BW(N)	BW(M)	Wait(N)10	Wait(M)5	Wait(NM)5	Wait(M)3	Wait(NM)3
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	10250	10250	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	12070	12070	0	0	0	0	0
3	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	8139	8139	0	0	0	0	0
4	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	15466	15466	0	0	0	1.90723	1.90723
5	1	1	0	0	0	3	0	0	0	0	0	0	0	0	0	13559	4820	0	0	0	0	0
6	1	0	0	1	0	1	1	0	0	2	0	0	0	0	0	8926	5186	0	0	0	4.6875	4.6875
7	0	0	0	1	1	2	0	0	0	0	0	0	2	0	1	18427	10351	0	0	0	3.26953	3.26953
8	1	0	1	0	0	0	2	0	0	0	0	2	0	1	1	35384	27827	0	0	0	44.3547	44.3547
9	1	3	1	2	1	0	0	0	0	1	0	0	0	0	0	22876	20211	0	0	0	4.6875	1.5625
10	3	2	1	1	0	0	0	1	0	1	0	0	1	0	0	56263	52920	0	0	0	2.93203	2.93203
11	1	1	2	0	1	1	2	0	0	1	1	1	0	0	0	45400	22118	1.27841	0	0	20.0334	20.0334
12	0	0	1	1	0	0	1	3	0	0	3	1	0	1	1	151463	67734	1.5625	2.34375	2.34375	123.349	123.349
13	1	1	1	2	2	1	0	1	0	0	0	3	0	0	1	81663	64661	9.14255	6.3122	6.3122	78.7182	78.7182
14	0	1	0	1	1	1	0	2	1	1	2	1	0	0	3	106678	53694	3.95089	6.83036	6.83036	60.2584	59.0996
15	2	2	0	2	1	4	1	1	0	0	0	0	0	0	2	75901	51728	29.2588	0	0	14.4706	14.4706
16	2	0	1	6	1	1	2	1	0	0	0	1	1	0	0	76057	62611	10.3206	0	0	47.8207	32.4347
17	1	3	0	1	3	4	0	1	0	0	2	0	1	1	0	94408	60098	23.0928	1.00023	1.00023	12.2546	10.9083
18	1	1	3	0	1	1	3	1	2	0	1	2	1	1	0	121103	72954	85.3466	15.2389	14.1973	173.846	161.772
19	1	5	1	2	1	3	3	0	0	1	2	0	0	0	0	47341	20211	24.9491	0	0	2.22039	2.22039
20	1	1	6	1	3	0	2	0	0	1	0	2	1	2	0	125015	32368	184.77	4.92188	1.40625	78.4736	51.3082
21	1	3	2	4	1	2	1	0	1	0	2	1	0	3	0	82944	36384	82.8372	24.7846	5.24684	162.096	86.6645
22	0	4	2	5	1	2	1	2	0	0	1	1	1	2	0	150022	72872	104.923	9.69688	7.56619	142.682	113.19
23	4	3	1	1	0	3	1	1	0	3	3	0	0	3	0	104659	67094	43.1365	0.61141	0.61141	26.3611	26.3611
24	1	5	4	1	1	2	1	1	0	0	1	1	1	3	2	146898	72954	203.434	5.37223	5.37223	92.1154	92.1154
25	2	2	4	1	4	2	0	5	1	0	1	2	0	0	1	301530	64754	340.868	21.5691	21.5691	109.663	112.57
26	5	3	2	0	1	1	2	3	0	3	2	2	1	0	1	176468	64801	70.9195	10.55	10.55	53.1125	53.1125
27	3	3	1	1	4	1	5	2	0	0	1	4	0	1	1	171441	72954	250.166	6.77286	6.77286	83.6318	83.6318
28	2	2	3	1	7	1	5	1	1	2	0	1	1	0	1	164104	64801	369.149	7.87026	7.87026	81.4939	53.694
29	5	1	5	3	5	2	2	0	0	1	1	1	1	1	1	131484	32368	202.608	3.87931	0.96983	62.6359	37.7296
30	2	2	4	3	4	5	1	1	2	1	2	0	2	0	1	155683	62970	415.895	3.18633	1.66055	45.3517	37.2654
31	4	4	0	2	2	6	1	1	2	1	2	2	1	2	1	125815	61976	196.159	3.53831	3.53831	35.652	35.652
32	3	6	3	1	1	3	4	2	2	2	1	2	1	1	0	179176	73001	293.011	12.0151	4.32214	131.678	63.0524
33	3	3	5	3	1	3	3	4	2	1	1	1	0	0	3	271353	64801	280.854	18.3288	15.3459	104.75	101.037
34	2	3	2	3	5	3	6	2	1	0	3	1	2	0	1	190352	64754	270.803	8.61914	8.61914	81.0271	57.9952
35	1	2	5	3	2	2	7	1	2	3	1	3	2	1	0	173822	73001	314.042	10.2479	7.43538	120.134	98.4018
36	1	2	6	3	2	3	5	3	3	0	1	2	2	1	2	268973	72954	472.466	16.0202	14.4577	192.018	173.908
37	1	0	4	4	4	3	3	2	3	2	1	1	3	2	4	226188	73001	459.314	14.1295	13.3694	168.353	160.199
38	2	3	3	6	0	2	5	1	1	1	3	3	2	3	3	152849	69001	287.708	7.64947	6.53927	82.5104	73.9319
39	2	2	2	6	6	3	5	1	3	1	1	1	1	3	2	194859	73001	521.165	11.3009	10.2192	138.184	125.859
40	2	7	4	4	4	2	3	4	0	2	2	1	2	3	0	299322	73001	537.778	11.0184	9.26055	128.869	115.286
41	6	4	3	5	3	3	5	1	1	2	1	1	4	0	2	148136	64801	364.938	6.74676	6.74676	70.6014	58.9923
42	1	5	4	2	5	5	5	2	2	3	2	2	2	1	1	234232	73001	443.209	9.82403	8.48475	113.705	103.357
43	8	1	7	5	1	3	6	0	2	4	2	1	2	1	0	149964	36431	344.655	23.2271	2.56241	141.492	48.9261
44	3	2	5	4	5	4	3	1	5	2	2	2	2	3	1	226074	73001	517.418	14.0662	12.4682	161.315	148.967
45	4	9	4	3	1	4	4	0	2	6	1	2	3	0	2	111211	28231	275.961	15.5772	5.83455	76.4925	48.8853
46	4	6	4	6	3	3	2	3	2	1	2	1	5	2	2	254506	73001	419.285	11.9765	10.1422	139.527	125.354
47	3	5	8	5	6	5	2	2	4	1	1	2	1	1	1	286424	73001	824.916	14.3652	11.074	164.057	134.436
48	3	3	5	4	4	2	7	2	3	4	2	2	1	4	2	270224	73001	547.081	11.1845	10.3056	134.624	125.196
49	2	4	0	6	5	3	10	1	0	6	2	5	1	2	2	169822	61976	556.671	1.60714	1.60714	20.8651	20.8651
50	9	5	5	3	3	6	4	1	2	2	1	3	1	3	2	220185	73001	667.343	7.17352	6.32977	91.2211	84.7014

Table B.2: Skewed Load1

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	BW(N)	BW(M)	Wait(N)10	Wait(M)5	Wait(NM)5	Wait(M)3	Wait(NM)3
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	40633	40633	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	8140	8140	0	0	0	0	0
3	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	10132	10132	0	0	0	0	0
4	0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	17361	17361	0	0	0	3.51562	3.51562
5	1	0	0	0	0	1	0	0	0	2	0	1	0	0	0	7009	6868	0	0	0	2.8125	2.8125
6	0	0	2	0	0	0	0	0	0	1	2	0	1	0	0	22904	11466	0	0	0	0	0
7	0	0	0	0	1	1	0	0	0	2	0	1	1	1	0	24514	20297	0	0	0	2.00893	2.00893
8	0	0	0	0	1	1	0	0	0	2	1	1	1	0	1	16408	12097	0	0	0	3.51562	3.51562
9	2	1	0	0	0	0	1	0	0	0	3	0	0	2	0	26760	14764	0	0	0	2.08333	2.08333
10	1	1	1	1	1	0	0	0	0	1	1	0	3	0	0	21497	20287	0	0	0	3.57539	2.93203
11	0	0	1	1	2	1	0	0	1	2	0	1	0	0	2	41826	27411	1.27841	2.55682	2.55682	90.5658	58.4462
12	1	0	0	0	0	1	1	0	1	2	2	1	3	0	0	15088	10931	1.27148	0	0	9.18197	9.18197
13	2	2	0	1	1	1	0	1	1	1	2	0	0	1	0	70617	60069	3.53365	2.16346	2.16346	16.5186	16.5186
14	1	1	0	0	0	0	0	2	0	0	6	0	1	2	1	106014	51873	2.55385	0	0	2.97405	2.97405
15	1	3	0	0	1	0	1	2	1	2	1	0	3	0	0	100569	49670	3.71328	0	0	3.82969	3.82969
16	1	2	1	3	1	1	0	1	1	0	1	2	0	1	1	87127	72954	15.4426	12.6011	11.4292	160.281	146.698
17	2	2	1	4	4	1	1	0	1	0	0	1	0	0	0	60838	26041	20.6159	0	0	85.0561	35.3537
18	2	2	2	3	1	1	1	1	0	0	1	2	2	0	0	86587	62704	26.2717	0.52083	0.52083	34.9521	26.8762
19	1	2	3	2	2	0	0	1	1	3	1	2	1	0	0	101600	62751	33.0287	1.48026	1.48026	43.1505	32.4142
20	3	2	1	1	0	0	1	0	0	5	1	2	2	1	1	36077	27968	15.7484	1.40625	1.40625	27.981	27.981
21	4	0	2	1	0	0	2	1	2	2	2	2	2	0	1	88452	60401	42.0879	13.7316	13.7316	67.0252	73.501
22	1	1	0	4	0	0	1	1	1	2	1	4	2	0	4	67112	49376	67.8048	2.68466	2.68466	30.2727	27.0767
23	2	1	1	2	1	1	3	0	3	3	1	2	1	2	0	73976	36431	71.5207	11.9261	8.96977	91.0916	81.1057
24	1	2	1	1	1	1	3	0	3	1	3	4	2	0	1	59015	28231	102.4	9.7679	7.42415	68.8319	59.7767
25	2	2	2	3	1	1	1	1	1	4	0	1	3	2	1	111838	73001	92.3398	9.37719	8.25219	114.804	106.111
26	2	1	3	1	0	3	0	1	1	3	2	3	4	1	1	111987	69001	50.8438	8.47566	7.39393	69.1273	60.7688
27	3	2	0	4	0	0	1	0	3	4	3	3	2	0	2	31446	12806	34.6393	3.02083	2.39583	30.0637	20.973
28	5	1	1	3	2	1	1	1	0	0	3	3	5	1	1	99683	72954	68.2684	4.26995	3.60031	63.3526	55.5911
29	3	2	0	2	4	1	1	1	0	2	4	2	2	3	2	124095	61976	173.331	3.20043	3.20043	27.8043	27.8043
30	3	1	0	1	0	4	0	3	3	1	5	3	3	2	1	182624	57976	118.372	5.90625	5.90625	54.9749	54.9749
31	4	3	1	2	0	2	0	2	2	1	3	6	4	0	1	128811	60801	90.7042	11.9491	11.9491	52.4707	59.0509
32	2	1	0	2	2	0	2	2	1	0	6	4	4	0	6	132155	53729	137.466	3.28125	3.28125	35.5788	34.4069
33	3	2	0	3	1	3	2	3	2	1	4	4	4	0	1	172339	53776	115.691	5.96591	3.83523	50.0384	41.0143
34	5	3	4	1	1	2	0	1	1	3	4	5	3	1	0	133051	73001	129.242	6.89499	5.24058	79.5874	66.8037
35	4	2	0	1	5	2	4	2	2	2	2	4	0	1	4	183325	61976	481.18	4.58036	4.58036	43.805	43.805
36	7	5	2	0	0	1	2	1	2	5	4	3	2	0	2	102948	60801	131.814	7.61947	7.61947	36.4702	36.4702
37	2	1	2	2	1	2	1	3	2	4	3	3	5	1	5	202402	73001	206.516	12.6093	12.6093	145.454	150.967
38	6	3	1	5	2	3	0	1	0	2	4	4	4	0	3	102887	64801	132.32	4.37983	4.37983	47.7859	47.7859
39	3	3	1	3	1	3	0	2	1	0	5	3	7	7	0	200406	72954	365.232	7.27374	7.27374	97.5107	97.5107
40	1	3	1	4	5	1	2	1	2	3	7	5	5	0	0	126141	62751	162.698	1.05469	1.05469	27.3738	27.3738
41	3	4	1	5	1	4	3	2	2	5	2	2	2	3	2	181691	73001	259.242	10.0636	9.72066	120.082	117.432
42	2	4	0	4	4	1	2	4	4	3	5	6	2	0	1	242017	53776	365.611	6.62946	4.62054	55.897	47.3886
43	3	5	0	3	4	1	3	2	1	3	3	3	8	2	2	170594	61976	171.368	2.87791	2.87791	29.704	29.704
44	4	5	3	4	1	1	1	1	2	6	3	5	3	2	3	143524	73001	222.891	7.83212	6.55371	96.1298	92.1148
45	1	2	0	2	1	0	0	3	8	6	5	8	4	1	4	200479	61976	231.014	7.0625	3.9375	63.4256	31.172
46	2	6	2	3	1	3	2	3	1	2	4	6	5	2	4	223591	73001	259.931	8.66406	8.66406	104.056	104.056
47	2	6	1	3	4	4	2	2	1	4	8	6	0	2	2	195882	73001	314.737	6.43459	6.43459	78.1687	78.1687
48	4	3	1	3	6	1	2	3	5	3	4	7	2	1	3	249539	73001	589.072	14.8483	14.8483	161.045	161.045
49	3	4	0	2	1	3	3	3	2	5	6	2	3	3	9	221201	61976	342.642	5.7398	4.30485	45.6414	39.564
50	4	2	3	2	4	2	2	2	4	5	8	5	3	2	2	219497	73001	413.133	11.2533	10.972	125.574	123.401

Table B.3: Skewed Load2

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	BW(N)	BW(M)	Wait(N)10	Wait(M)5	Wait(NM)5	Wait(M)3	Wait(NM)3
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	738	738	0	0	0	0	0
2	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	15313	15313	0	0	0	0	0
3	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	40998	40998	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0	29640	29640	0	0	0	3.51562	3.51562
5	0	0	0	0	1	0	0	1	0	0	0	1	0	1	1	62840	60790	0	0	0	38.1441	38.1441
6	1	0	0	1	0	2	0	0	1	0	1	0	0	0	0	13202	9202	0	0	0	6.875	6.875
7	1	0	0	0	2	1	0	1	0	0	1	0	0	1	0	71797	59797	0	0	0	13.0619	13.0619
8	1	0	0	0	2	2	0	0	0	0	0	0	1	2	0	45396	19146	0	0	0	1.90723	1.90723
9	0	0	1	0	2	1	2	0	0	0	1	1	0	1	0	50700	31500	0	0	0	22.2328	22.2328
10	0	0	0	0	1	2	0	2	1	0	0	0	0	3	1	134129	58883	0	0	0	0	0
11	1	2	0	0	0	1	1	0	2	0	0	1	1	1	1	32307	21041	0.69354	0	0	49.5962	49.5962
12	0	2	1	1	1	0	0	0	1	1	1	2	0	1	1	41364	36349	2.65625	14.6526	8.49818	109.179	80.3004
13	1	2	3	0	2	1	0	1	2	0	0	0	0	0	1	106857	62753	23.9864	0	0	66.2443	66.2443
14	0	0	0	0	1	1	2	1	1	0	3	2	0	2	1	90542	60884	16.3015	0	0	29.2547	29.2547
15	0	1	0	2	0	6	0	1	1	1	0	2	0	1	0	84090	57894	42.0816	3.9375	3.9375	43.1029	38.2279
16	2	1	0	1	1	2	1	1	3	2	1	0	0	1	0	85652	60069	19.5021	3.51562	0.87891	25.4365	14.2693
17	0	2	1	1	4	1	2	0	1	2	0	1	0	1	1	74703	36349	70.8566	10.343	10.343	103.133	85.7561
18	0	1	0	1	1	2	0	2	3	2	1	0	1	4	0	151871	60063	77.6451	5.60156	5.17773	30.4015	29.9327
19	0	0	1	3	6	0	1	1	3	0	1	0	2	0	1	118644	62103	80.1277	1.60609	1.60609	47.9422	47.1392
20	1	1	1	1	3	1	1	3	0	2	0	1	1	4	0	209798	73001	145.78	13.715	13.715	161.109	161.109
21	1	1	1	0	0	5	0	4	0	3	0	0	3	2	1	218542	67170	80.6384	3.04185	2.12277	85.3765	38.8664
22	1	0	3	1	2	2	4	3	1	1	1	1	1	1	0	211626	73001	205.018	17.4766	6.28675	191.531	91.7126
23	2	0	2	1	0	2	5	1	4	1	0	1	0	2	2	133900	69001	170.283	19.6731	19.6731	133.441	128.55
24	3	1	3	0	1	2	3	1	1	0	1	1	1	3	3	147424	72954	224.881	8.01009	8.01009	114.146	114.146
25	1	1	5	0	0	1	3	1	2	0	2	2	3	3	1	158399	68954	270.368	11.722	11.347	101.898	97.552
26	0	2	1	0	2	1	3	2	4	0	2	1	0	6	2	208741	72872	492.598	17.4036	5.00397	219.002	97.1892
27	1	1	2	2	1	2	2	3	3	2	0	1	2	5	0	235389	73001	346.019	20.3186	20.3186	226.958	226.958
28	0	1	1	1	4	1	5	4	2	1	2	2	0	3	1	273815	72919	448.52	16.8301	16.8301	203.638	197.611
29	1	2	2	1	1	2	1	2	2	2	0	3	1	5	4	199544	73001	264.827	12.7732	12.7732	159.42	157.076
30	4	1	2	2	1	3	2	3	2	2	2	2	1	1	2	202906	73001	191.905	15.5514	15.0827	172.125	168.503
31	2	2	2	0	4	0	1	4	3	3	2	1	2	4	1	281660	73001	378.981	19.4366	19.4366	211.628	211.628
32	0	0	5	0	5	3	3	2	3	3	2	2	0	3	1	249730	72181	546.219	2.19727	2.19727	156.558	151.037
33	1	1	2	5	2	6	3	4	3	0	1	0	1	2	2	275477	71123	568.844	3.60689	2.91335	192.401	185.793
34	2	2	4	3	5	3	1	4	3	1	1	1	0	3	1	314057	73001	493.475	19.7896	18.5488	220.243	202.106
35	0	1	1	3	2	8	2	2	1	2	1	4	2	4	2	206449	72919	387.543	7.93741	7.93741	109.055	105.841
36	2	0	1	2	3	3	5	4	3	3	3	2	1	1	3	263057	73001	350.054	17.1277	17.1277	187.598	187.598
37	1	3	2	0	5	6	4	4	1	0	2	1	2	5	1	326079	72954	728.272	11.5958	11.5958	136.792	132.991
38	2	3	2	2	1	3	5	4	3	2	2	3	2	2	2	270472	73001	553.531	16.5963	16.5963	182.003	182.003
39	1	2	2	2	3	2	6	6	5	1	2	1	2	3	1	378151	73001	825.464	24.587	24.587	261.551	261.551
40	4	2	7	2	2	6	2	1	5	2	2	0	1	3	1	225454	71170	605.587	3.82588	2.87227	142.921	134.421
41	1	4	4	1	1	5	3	5	7	1	1	0	3	2	3	346522	71170	745.664	5.04621	5.04621	249.776	249.776
42	4	4	3	0	4	2	5	2	1	2	1	1	3	6	4	255527	73001	592.537	6.86579	6.86579	94.991	94.991
43	4	2	1	4	6	3	2	4	5	1	2	1	0	6	2	334794	73001	656.969	18.8102	18.8102	211.426	203.685
44	0	1	2	1	3	3	4	5	2	4	1	3	4	6	5	363589	72919	1007.43	12.7082	12.7082	164.173	164.173
45	3	3	5	2	4	3	6	4	4	3	0	4	1	2	1	336440	73001	751.672	16.7757	15.5257	183.923	174.264
46	0	2	3	0	7	8	2	5	2	2	1	5	2	4	3	398933	72919	918.115	12.1556	12.1556	152.753	152.753
47	1	3	3	3	6	6	6	4	2	0	0	4	1	3	5	350425	72861	911.861	10.4756	10.4756	140.442	140.442
48	3	8	2	2	3	2	2	5	2	4	0	4	4	3	4	329259	73001	635.476	13.1388	13.1388	148.628	148.628
49	5	3	7	1	2	4	3	4	4	3	0	1	3	4	5	360687	73001	994.711	15.6932	15.6932	181.701	181.701
50	1	2	8	4	1	5	2	11	1	3	3	1	3	4	1	625315	73001	1499.62	23.0691	21.1003	251.604	247.27

Table B.4: Skewed Load3

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	BW(N)	BW(M)	Wait(N)10	Wait(M)5	Wait(NM)5	Wait(M)3	Wait(NM)3
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	820	820	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	2645	2645	0	0	0	0	0
3	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	17363	17363	0	0	0	0	0
4	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	12301	8301	0	0	0	3.51562	3.51562
5	0	0	1	0	2	0	1	1	0	0	0	0	0	0	0	71483	59883	0	0	0	38.1441	38.1441
6	0	0	1	2	0	0	1	1	0	0	0	0	1	0	0	56009	55559	0	0	0	6.875	6.875
7	1	1	2	1	0	1	0	0	0	0	1	0	0	0	0	28377	16164	0	0	0	13.0619	13.0619
8	0	0	3	0	2	0	0	0	0	1	1	0	1	0	0	50060	19466	0	0	0	1.90723	1.90723
9	1	1	0	1	0	1	1	0	0	0	1	1	1	1	0	21711	17296	0	1.04167	1.04167	22.2328	22.2328
10	0	0	2	0	0	1	0	0	0	2	2	0	3	0	0	27197	15466	0	0	0	0	0
11	2	1	1	0	0	1	1	0	1	0	0	1	1	0	2	31376	24091	0.69354	7.45987	7.45987	49.5962	49.5962
12	3	1	0	2	1	2	2	0	0	0	0	0	0	1	0	37100	19295	3.75	0	0	109.179	80.3004
13	2	0	2	1	2	1	1	0	1	1	0	1	1	0	0	54153	26181	3.98618	3.24519	1.08173	66.2443	66.2443
14	1	0	2	1	1	1	1	1	2	1	0	1	0	0	2	94053	64801	23.1158	20.5974	20.5974	29.2547	29.2547
15	0	2	3	0	3	1	1	0	2	0	1	0	0	2	0	95547	34395	62.1859	1.875	1.25	43.1029	38.2279
16	3	3	2	0	0	0	1	0	3	0	1	0	2	0	1	45263	21953	15.387	2.60229	1.64868	25.4365	14.2693
17	5	1	3	2	0	2	0	1	0	0	1	1	1	0	0	89751	58704	18.9691	0.55147	0.55147	103.133	85.7561
18	0	0	4	1	2	3	2	0	1	1	2	0	0	2	0	105316	33703	156.329	0	0	30.4015	29.9327
19	4	0	2	2	1	2	0	0	1	2	1	1	1	1	1	60954	36431	42.238	18.7558	9.37788	47.9422	47.1392
20	0	1	4	1	3	1	4	1	0	0	1	0	2	0	2	133343	62841	94.3057	2.37598	2.37598	161.109	161.109
21	3	0	5	0	1	1	0	1	0	3	2	1	2	1	1	126314	73001	34.1738	7.25577	5.24684	85.3765	38.8664
22	2	2	4	1	1	3	1	3	1	0	0	0	2	2	0	218558	71030	219.153	1.38707	1.38707	191.531	91.7126
23	2	2	4	1	0	4	4	0	0	0	1	1	0	3	1	113544	28321	171.233	2.03804	1.63043	133.441	128.55
24	3	0	2	1	0	2	1	2	2	3	1	2	1	2	2	155186	69001	110.68	15.4342	15.4342	114.146	114.146
25	1	2	3	3	1	3	2	1	0	2	1	1	1	2	2	131514	73001	131.061	6.65734	5.53234	101.898	97.552
26	2	1	8	1	4	1	0	3	1	1	1	0	1	1	1	267178	71170	495.47	4.80424	2.7503	219.002	97.1892
27	0	1	2	1	3	6	3	0	0	1	1	2	2	1	4	104914	32286	321.307	1.5625	1.5625	226.958	226.958
28	1	2	3	4	2	2	2	3	0	1	1	2	3	1	1	206624	73001	190.195	12.3076	9.79646	203.638	197.611
29	5	0	7	4	4	1	2	1	1	0	0	1	1	1	1	185932	72861	412.813	5.65921	5.65921	159.42	157.076
30	5	1	6	2	2	2	3	1	0	0	1	0	1	3	3	185294	71123	450.062	2.34687	1.83828	172.125	168.503
31	3	0	3	3	5	1	4	1	1	0	2	1	1	3	3	179054	72954	298.509	7.10862	7.10862	211.628	211.628
32	3	1	5	3	3	3	2	2	0	2	3	1	2	1	1	199514	73001	543.657	8.64429	6.00757	156.558	151.037
33	2	0	2	3	5	7	6	0	1	1	1	2	1	1	1	134904	36431	355.656	13.2854	5.39938	192.401	185.793
34	5	2	5	3	1	4	1	0	1	1	6	1	2	1	1	109231	36431	180.615	20.1351	5.24058	220.243	202.106
35	2	2	4	3	3	0	3	2	1	3	6	2	4	0	0	174026	62751	213.539	1.20536	1.20536	109.055	105.841
36	3	1	4	6	2	1	4	1	4	2	2	2	1	3	0	175946	73001	328.392	15.3033	12.5689	187.598	187.598
37	3	2	6	4	8	2	3	0	1	1	1	1	2	2	1	184046	36431	571.773	23.3182	3.358	136.792	132.991
38	3	2	4	4	4	1	3	3	4	2	1	2	2	3	0	269883	73001	497.461	18.0766	16.2263	182.003	182.003
39	4	1	7	5	1	6	5	2	3	1	1	0	1	1	1	239962	71170	500.202	4.15024	2.19411	261.551	261.551
40	2	2	6	2	5	5	2	1	4	3	0	3	1	2	2	225974	73001	610.446	13.0698	11.6636	142.921	134.421
41	2	3	6	3	4	4	3	1	4	0	2	0	3	3	3	225034	71123	562.426	2.81793	2.25972	249.776	249.776
42	8	3	5	1	2	5	4	1	2	1	4	0	2	2	2	189683	71170	394.274	2.09431	1.73103	94.991	94.991
43	4	4	10	4	3	5	4	1	2	2	1	0	1	1	1	239547	71170	603.478	3.46493	1.33594	211.426	203.685
44	5	2	9	5	6	6	3	1	1	1	2	0	2	0	1	237982	62970	704.599	3.06658	0.81259	164.173	164.173
45	5	2	9	2	7	4	6	1	0	0	1	2	1	2	3	272147	72954	805.914	4.11519	2.86519	183.923	174.264
46	7	1	6	6	2	2	3	3	3	2	2	2	3	2	2	273334	73001	574.955	14.3718	11.9261	152.753	152.753
47	5	4	8	5	4	4	5	0	0	2	2	3	1	2	2	195048	32368	541.091	3.88963	1.19681	140.442	140.442
48	3	4	6	5	2	5	6	1	1	2	4	4	3	0	2	188951	64801	489.296	6.64176	4.00505	148.628	148.628
49	5	1	5	6	5	7	7	2	1	0	2	2	2	2	2	269725	72954	659.573	7.12859	5.7893	181.701	181.701
50	4	7	8	1	4	2	6	1	1	1	6	4	1	0	4	221581	64801	721.118	5.81359	5.81359	251.604	247.27

Table B.5: Skewed Load4

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	BW(N)	BW(M)	Wait(N)10	Wait(M)5	Wait(NM)5	Wait(M)3	Wait(NM)3
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	140	140	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	19250	19250	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	54883	54883	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	1	0	0	0	2	0	61273	51023	0	0	0	0	0
5	1	0	1	0	2	0	0	0	0	1	0	0	0	0	0	28211	20211	0	0	0	2.8125	2.8125
6	0	0	1	0	1	1	0	1	0	1	0	1	0	0	0	65931	61931	0	0	0	34.1305	34.1305
7	1	0	0	0	1	1	0	3	0	0	0	0	0	1	0	144970	59703	0	0	0	35.168	35.168
8	1	0	1	0	1	2	0	0	0	0	0	1	0	2	0	50477	32227	0	0	0	44.3547	44.3547
9	1	0	1	0	2	1	0	1	0	2	0	1	0	0	0	74892	62751	0	1.5625	1.5625	34.9963	34.9963
10	1	0	2	0	1	0	0	2	0	1	0	0	0	3	0	143477	71094	0	2.8125	2.8125	180.63	187.43
11	1	0	1	0	1	2	0	1	0	0	0	3	0	2	0	94925	72861	7.45987	7.45987	7.45987	104.985	104.985
12	1	0	1	0	1	2	0	2	0	1	0	4	0	0	0	117106	62751	8.01009	2.34375	2.34375	42.1406	42.1406
13	1	0	3	0	2	1	0	1	0	1	0	2	0	2	0	119658	73001	31.505	9.55739	8.47566	122.502	114.144
14	0	0	3	0	1	1	0	4	0	3	0	1	0	1	0	220861	72181	17.6407	4.01786	1.00446	270.445	114.648
15	4	0	5	0	1	1	0	2	0	0	0	1	0	1	0	164956	72861	56.4793	10.9411	10.9411	175.711	141.264
16	4	0	1	0	2	0	0	1	0	4	0	1	0	3	0	104385	73001	19.7805	6.00757	6.00757	79.9427	79.9427
17	2	0	1	0	3	2	0	2	0	3	0	1	0	3	0	159236	73001	70.4132	10.4812	10.4812	125.608	121.472
18	1	0	1	0	1	4	0	2	0	1	0	3	0	5	0	174449	73001	167.951	9.89887	9.89887	127.747	127.747
19	2	0	3	0	1	3	0	3	0	4	0	2	0	1	0	191917	73001	75.0229	15.177	13.6968	165.929	154.491
20	2	0	2	0	2	3	0	6	0	5	0	0	0	0	0	296643	60844	285.437	0	0	26.0238	26.0238
21	2	0	3	0	1	4	0	2	0	3	0	1	0	5	0	194236	73001	165.16	9.82403	9.82403	127.661	125.653
22	3	0	4	0	5	3	0	4	0	0	0	2	0	1	0	276059	72861	342.879	14.9197	3.72994	183.861	85.9598
23	5	0	4	0	2	2	0	2	0	2	0	5	0	1	0	174436	73001	119.025	9.58118	7.74694	109.97	95.7969
24	6	0	1	0	2	3	0	5	0	2	0	2	0	3	0	282185	73001	210.695	17.6815	6.3488	190.73	86.8397
25	4	0	1	0	0	3	0	4	0	3	0	7	0	3	0	233587	69001	155.908	13.6919	5.53234	85.4133	58.6123
26	2	0	4	0	4	3	0	2	0	5	0	4	0	2	0	200739	73001	234.864	8.47566	7.39393	102.06	93.7014
27	1	0	5	0	4	5	0	3	0	5	0	1	0	3	0	264330	73001	471.854	11.7218	10.6801	139.575	127.359
28	2	0	4	0	5	2	0	4	0	4	0	4	0	3	0	296115	73001	470.161	13.7316	13.2294	155.848	151.967
29	4	0	5	0	3	3	0	2	0	4	0	3	0	5	0	234332	73001	723.025	8.08378	8.08378	106.083	106.083
30	2	0	3	0	6	4	0	3	0	3	0	5	0	4	0	272248	73001	663.888	9.61211	9.61211	116.107	116.107
31	4	0	4	0	2	3	0	5	0	5	0	6	0	2	0	312094	73001	412.296	15.0498	14.1425	163.472	156.461
32	8	0	5	0	4	4	0	1	0	5	0	3	0	2	0	178372	73001	267.632	4.7616	3.44324	63.8838	53.6968
33	1	0	6	0	2	9	0	5	0	3	0	2	0	5	0	378973	73001	719.829	14.9899	14.5638	175.14	168.012
34	4	0	3	0	7	3	0	4	0	2	0	7	0	4	0	322196	73001	531.547	10.8948	10.8948	126.39	126.39
35	5	0	5	0	2	4	0	3	0	3	0	11	0	2	0	256153	73001	481.744	9.04252	7.83717	104.122	94.8085
36	5	0	2	0	7	5	0	4	0	8	0	3	0	2	0	292482	73001	569.799	9.89887	9.89887	110.293	110.293
37	4	0	2	0	4	6	0	8	0	2	0	5	0	6	0	478165	73001	1064.54	18.5025	7.47614	203.43	102.348
38	4	0	6	0	5	4	0	2	0	7	0	6	0	4	0	261476	73001	553.676	6.53927	5.79914	84.9275	79.2084
39	10	0	6	0	6	3	0	4	0	2	0	4	0	4	0	347148	73001	940.039	10.5797	9.85857	124.497	118.924
40	3	0	11	0	2	1	0	5	0	3	0	8	0	7	0	436807	73001	1203.19	14.1245	11.312	169.621	156.371
41	4	0	4	0	1	7	0	2	0	7	0	8	0	8	0	263790	73001	723.64	5.37481	5.37481	77.0726	77.0726
42	9	0	3	0	6	5	0	7	0	4	0	4	0	4	0	442759	73001	926.225	14.6809	14.6809	160.798	160.798
43	8	0	4	0	8	5	0	3	0	7	0	3	0	5	0	315420	73001	646.362	7.03316	7.03316	88.3296	85.3863
44	6	0	9	0	6	5	0	5	0	3	0	3	0	7	0	455233	73001	1186.65	12.2013	11.5621	148.088	143.149
45	9	0	7	0	3	6	0	4	0	6	0	3	0	7	0	374983	73001	921.185	9.4816	9.4816	117.606	117.606
46	6	0	7	0	10	2	0	4	0	5	0	6	0	6	0	407852	73001	1227.57	9.27548	8.96977	114.182	111.82
47	4	0	8	0	7	5	0	3	0	10	0	5	0	5	0	353374	73001	909.192	7.6314	6.73379	96.7445	89.8087
48	6	0	4	0	8	7	0	5	0	5	0	8	0	5	0	412300	73001	1152.54	9.71965	9.71965	113.634	113.634
49	2	0	10	0	7	7	0	7	0	4	0	6	0	6	0	556079	73001	1891.64	14.5926	13.4446	169.283	160.413
50	8	0	8	0	6	3	0	6	0	6	0	4	0	9	0	501086	73001	1143.27	12.097	12.097	145.505	145.505

Table B.6: Distinct Query Load

Appendix C Graphs

This appendix gives the graphs plotted for different loads. Section C.1 shows the graphs for a Balanced Load and Sections C.2 to C.5 give graphs for Skewed Loads 1-5. Section C.6 gives the graphs for a Distinct Query Load. There are three graphs plotted for each type of load: bandwidth usage, average wait time with 5 channels, and average wait time with 3 channels. Discussion of these graphs appears in Chapter 3.

C.1 Balanced Load

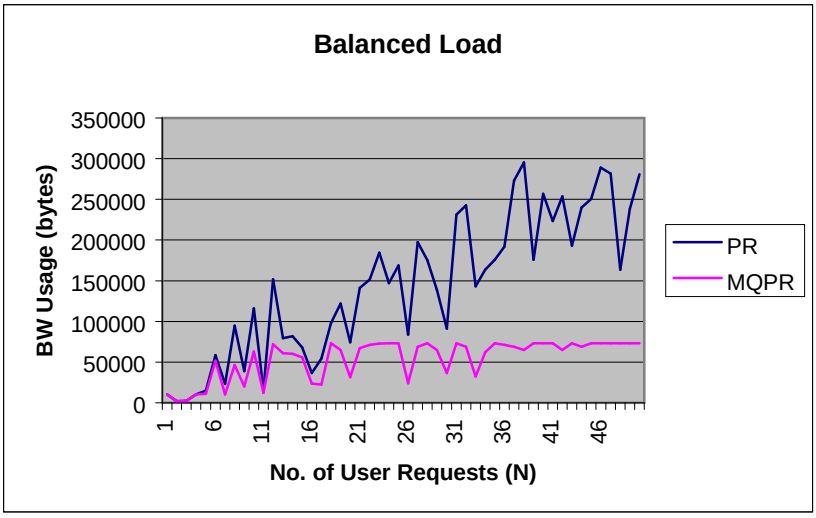


Figure C.1: Bandwidth Usage for Balanced Load

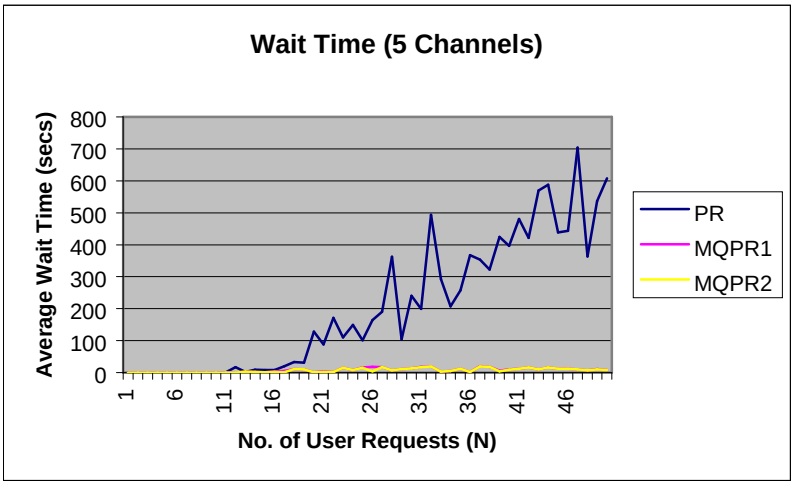


Figure C.2: Average Wait Time with 5 Channels for Balanced Load

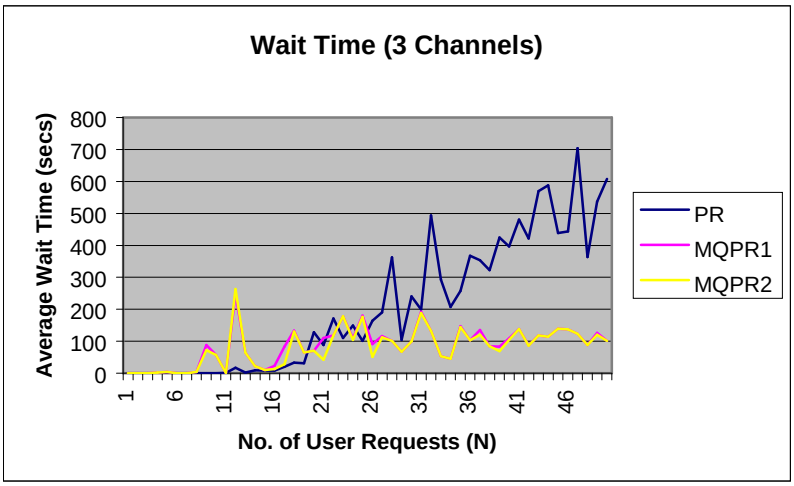


Figure C.3: Average Wait Time with 3 Channels for Balanced Load

C.2 Skewed Load1

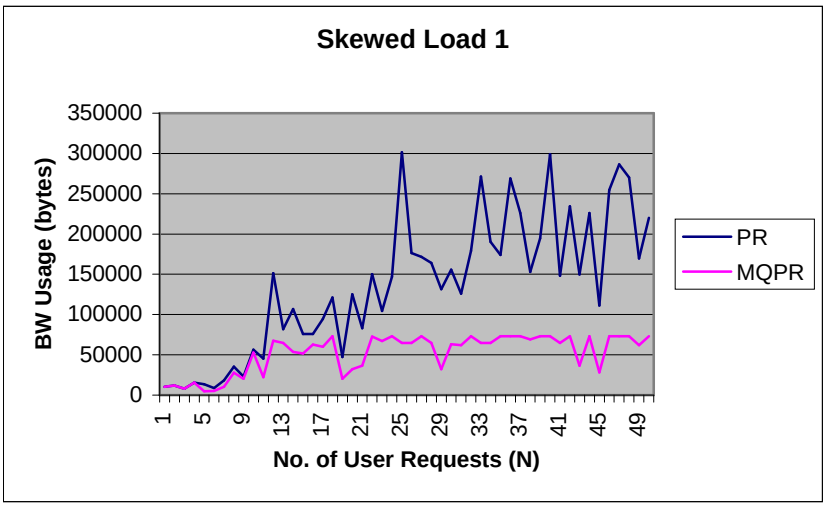


Figure C.4: Bandwidth Usage for Skewed Load1

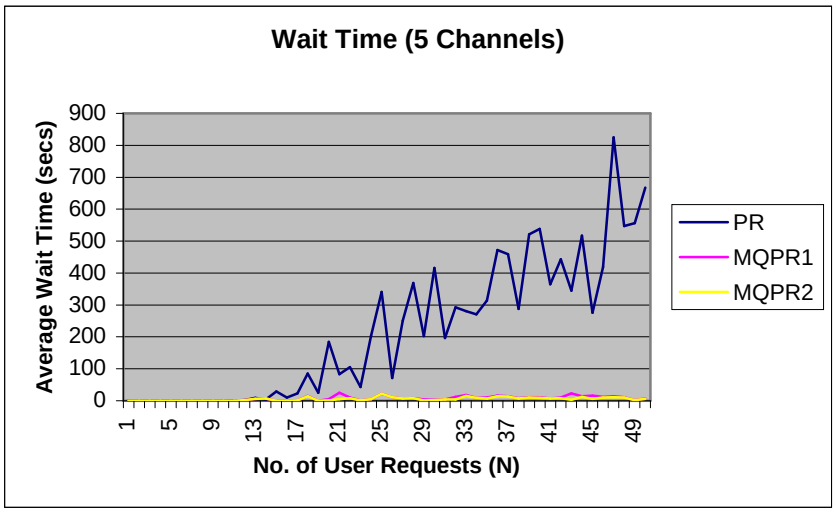


Figure C.5: Average Wait Time with 5 Channels for Skewed Load1

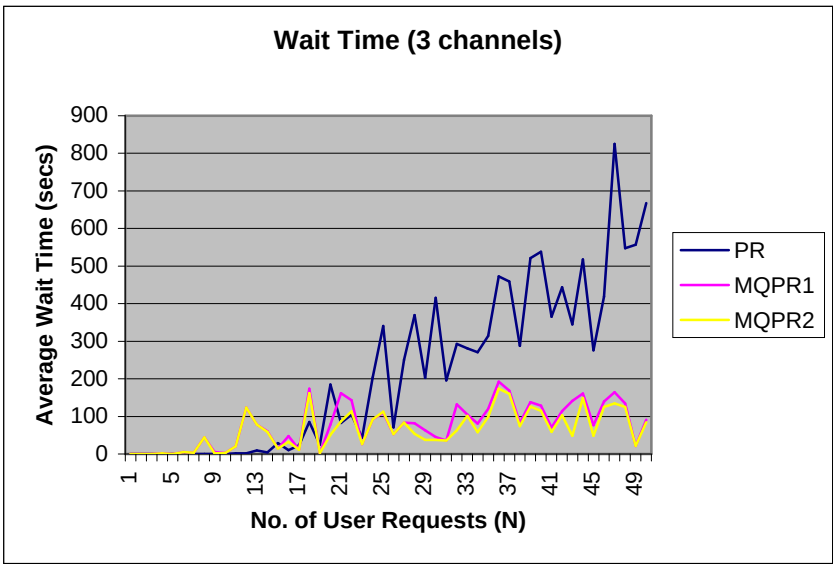


Figure C.6: Average Wait Time with 3 Channels for Skewed Load1