Optimizing Communication Cost in Distributed Query Processing

Abdeslem Belghoul

N^o d'ordre : D.U 2825
EDSPIC : 804

Universite Clermont Auvergne

Ecole Doctorale
Sciences Pour l'Ingenieur de Clermont-Ferrand

Thèse

Présentée par

ABDESLEM BELGHOUL

pour obtenir le grade de

Docteur d'Université

SPECIALITE : INFORMATIQUE

Titre de la thèse : Optimizing Communication Cost in Distributed Query Processing

Soutenue publiquement le 7 juillet 2017, devant le jury :

– Prof. Bernd Amann	Université Pierre et Marie Curie, Paris	*Président*
– Prof. Mourad Baïou	CNRS-Université Clermont Auvergne	*Co-directeur*
– MCF. Marinette Bouet	Université Clermont Auvergne	*Examinatrice*
– MCF. Radu Ciucanu	Université Clermont Auvergne	*Membre invité*
– Prof. Franck Morvan	Université Paul Sabatier, Toulouse	*Rapporteur*
– Prof. Claudia Roncancio	Institut Polytechnique de Grenoble	*Examinatrice*
– Prof. Farouk Toumani	Université Clermont Auvergne	*Directeur de thèse*
– MCF-HDR. V.Marian Scuturici	Institut National des Sciences Appliquées-Lyon	*Rapporteur*

Declaration

This dissertation has been completed by Abdeslem Belghoul under the supervision of Professors Farouk Toumani and Mourad Baïou. This work has not been submitted for any other degree or professional qualification. I declare that the work presented in this dissertation is entirely my own except where indicated by full references.

Abdeslem Belghoul

Acknowledgment

In this work founded by our Department and LIMOS of Clermont Auvergne University, I wish to express my sincere thanks to my supervisors Prof. Farouk Toumani and Prof. Mourad Baïou for the continuous support of my PhD study. They always find a way to make a complex thought understandable and I hope to have gained a bit of this ability as well. I could not image to have finished this thesis without their immense patience, motivation and guidance throughout the years of research and while writing this thesis.

My sincere thanks also goes to Prof. Nourine El-Houari, Prof. Mephu Nguifo Engelbert and MCF Ciucanu Vasile-Radu for their great advises during my research and for sharing their expertise. Also, my thanks to Engineer Frédéric Gaudet for the availability of experimental means. Furthermore, I would like to thank all of my colleges at LIMOS for their discussions and the good time I spend at the laboratory.

I would also like to take this opportunity to express my gratitude to my family for their support during the years of study.

Abstract

In this thesis, we take a complementary look to the problem of optimizing the time for communicating query results in distributed query processing, by investigating the relationship between the communication time and the middleware configuration. Indeed, the middleware determines, among others, *how* data is divided into batches and messages before being communicated over the network. Concretely, we focus on the research question: given a query Q and a network environment, what is the best middleware configuration that minimizes the time for transferring the query result over the network?

To the best of our knowledge, the database research community does not have well-established strategies for middleware tuning.

We present first an intensive experimental study that emphasizes the crucial impact of middleware configuration on the time for communicating query results. We focus on two middleware parameters that we empirically identified as having an important influence on the communication time: (i) the *fetch size* F (i.e., the number of tuples in a batch that is communicated at once to an application consuming the data) and (ii) the *message size* M (i.e., the size in bytes of the middleware buffer, which corresponds to the amount of data that can be communicated at once from the middleware to the network layer; a batch of F tuples can be communicated via one or several messages of M bytes). Then, we describe a cost model for estimating the communication time, which is based on *how* data is communicated between computation nodes. Precisely, our cost model is based on two crucial observations: (i) batches and messages are communicated differently over the network: batches are communicated synchronously, whereas messages in a batch are communicated in pipeline (asynchronously), and (ii) due to network latency, it is more expensive to communicate the first message in a batch compared to any other message that is not the first in its batch. We propose an effective strategy for calibrating the network-dependent parameters of the communication time estimation function i.e, the costs of first message and non first message in their batch. Finally, we develop an optimization algorithm to effectively compute the values of the middleware parameters F and M that minimize the communication time. The proposed algorithm allows to quickly find (in small fraction of a second) the values of the middleware parameters F and M that translate a good trade-off between low resource consumption and low communication time. The proposed approach has been evaluated using a dataset issued from application in Astronomy.

Keywords: middleware; distributed query processing; communication cost model; fetch size; message size; optimizing communication cost.

Résumé

Dans cette thèse, nous étudions le problème d'optimisation du temps de transfert de données dans les systèmes de gestion de données distribuées, en nous focalisant sur la relation entre le temps de communication de données et la configuration du middleware. En réalité, le middleware détermine, entre autres, *comment* les données sont divisées en lots de F tuples et messages de M octets avant d'être communiqués à travers le réseau. Concrètement, nous nous concentrons sur la question de recherche suivante : étant donnée requête Q et l'environnement réseau, quelle est la meilleure configuration de F et M qui minimisent le temps de communication du résultat de la requête à travers le réseau?

A notre connaissance, ce problème n'a jamais été étudié par la communauté de recherche en base de données.

Premièrement, nous présentons une étude expérimentale qui met en évidence l'impact de la configuration du middleware sur le temps de transfert de données. Nous explorons deux paramètres du middleware que nous avons empiriquement identifiés comme ayant une influence importante sur le temps de transfert de données: (i) la taille du *lot* F (c'est-à-dire le nombre de tuples dans un lot qui est communiqué à la fois vers une application consommant des données) et (ii) la taille du *message* M (c'est-à-dire la taille en octets du tampon du middleware qui correspond à la quantité de données à transférer à partir du middleware vers la couche réseau). Ensuite, nous décrivons un modèle de coût permettant d'estimer le temps de transfert de données. Ce modèle de coût est basé sur la manière dont les données sont transférées entre les nœuds de traitement de données. Notre modèle de coût est basé sur deux observations cruciales: (i) les lots et les messages de données sont communiqués différemment sur le réseau : les lots sont communiqués de façon synchrone et les messages dans un lot sont communiqués en pipeline (asynchrone) et (ii) en raison de la latence réseau, le coût de transfert du premier message d'un lot est plus élevé que le coût de transfert des autres messages du même lot. Nous proposons une stratégie pour calibrer les poids du premier et non premier messages dans un lot. Ces poids sont des paramètres dépendant de l'environnement réseau et sont utilisés par la fonction d'estimation du temps de communication de données. Enfin, nous développons un algorithme d'optimisation permettant de calculer les valeurs des paramètres F et M qui fournissent un bon compromis entre un temps optimisé de communication de données et une consommation minimale de ressources. L'approche proposée dans cette thèse a été validée expérimentalement en utilisant des données issues d'une application en Astronomie.

Mots clés : middleware; traitement des requêtes distribuées; coût de communication de données; taille du fetch; taille du message; optimisation du coût de communication.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Data transfer over a network is an inherent task of distributed query processing in the various existing distributed data management architectures [25, 36] (e.g., *Client-Server*, *Peer-to-Peer*, *Parallel* and *data integration* (*mediated*) systems). In all such architectures, a given node (playing the role of a server, a client, a mediator, etc.) may send a query (or a subquery) to another node (a server or a mediator) which will execute the query and send back the query results to the requester node.

Despite the tremendous advances made both in networking and telecommunication technology from one side, and distributed computing and data management techniques from another side, the cost underlying data transfer (called also communication time) is still often an important source of performance problems. This is due to the ever-increasing load imposed by modern data-intensive applications. As a consequence, minimizing the communication time has been recognized for a long time as one of the major research challenges in distributed data management area [25, 36]. A long-standing research effort from both academia and industry focused on developing techniques that minimize the total amount of data that needs to be communicated over the network [5, 14, 17, 25, 29, 36]. When dealing with the communication cost, all but few state-of-the-art distributed query optimization techniques [17, 25, 29] focus on the generation of query plans that minimize the amount of data to be exchanged over the network using various techniques, for example filtering outer relation with *semijoin* or *Bloom-filter* to reduce the volume of data [25, 29] and pushing the join to remote sites [17] to avoid the communication overhead, etc. Query result prefetching and caching have also been used to reduce the latency of network and query execution e.g., by anticipating the computation of query results before they are needed by an application [37], just to mention a few.

In this thesis, we take a complementary look to the problem of optimizing the time for communicating query results in a distributed environment, by focusing on *how data* is transferred over a network. To achieve this goal, we investigate the relationship between the communication time and the *middleware* configuration. Indeed, today, most programs (including application programs, DBMSs, and modern massively parallel frameworks like Apache Hive [1] and Apache Spark [2]) interact with data management systems using a remote data access middleware such as ODBC [15], JDBC [40], or a proprietary middleware [6]. A remote data access middleware (or simply, a middleware in the sequel) is a layer on top of a network protocol that is in charge of managing the connectivity and data transfer between a client application and a data server in

[1] `https://cwiki.apache.org/confluence/display/Hive/HiveClient`
[2] `http://spark.apache.org/sql/`

distributed and heterogeneous environments. Of particular interest to our concerns, a middleware determines *how* data is divided into batches and messages before being communicated over the network. As we demonstrate in the sequel, this impacts drastically the communication time.

We analyze the middleware-based communication model and we identify empirically two middleware parameters that have a crucial impact on the communication time:

- The *fetch size*, denoted F, which defines the number of tuples in a batch that is communicated at once to an application consuming the data, and
- The *message size*, denoted M, which defines the size in bytes of the middleware buffer and corresponds to the amount of data that can be communicated at once from the middleware to the network.

The middleware parameters F and M can be tuned in virtually all standard or DBMS-specific middleware [7, 15, 16, 30, 40], where they are usually set manually by database administrators or programmers. The main thesis of this work is that tuning the middleware parameters F and M is:

- An important problem because the middleware parameters F and M have a great impact on the communication time of a query result and on resource consumption, and
- A non-trivial problem because the optimal values of the parameters are *query-dependent* and *network-dependent*.

We briefly illustrate these points via Example 1.1.

Example 1.1. We consider two queries (that we present later on in detail in Figure 3.3) having same selectivities and different tuple sizes:

– Q_1: result of $\sim 32GB = \sim$165M tuples $\times$ $205B$/tuple;
– Q_3: result of $\sim 4.5GB = \sim$165M tuples $\times$ $27B$/tuple.

Moreover, we take two networks: high-bandwidth ($10Gbit/s$) and low-bandwidth ($50Mbit/s$). Finally, we consider the following two different middleware configurations:

– *Configuration* C_1: F=110K tuples and M=$4KB$;
– *Configuration* C_2: F=22K tuples and M=$32KB$.

For the moment ignore the choice of the actual values of middleware parameter; in Chapter 3, we discuss in greater detail the precise meaning of each parameter and we present extensive results for multiple combinations of parameter values that strengthen the points that we already make in this example.

(i) To show that the communication time is sensitive to the middleware configuration, we report in Table 1.1 the communication times (in seconds) for Q_1 and Q_3, in the high-bandwidth network. We observe that the time needed to transfer a given volume of data varies depending on the considered middleware configuration. For each query, we observe that different middleware configurations drive dramatically different communication times.

(ii) To illustrate that the best middleware configuration is *query-dependent*, we consider again Table 1.1, which reports the communication times (in seconds) for Q_1 and Q_3, in the high-bandwidth network. We observe that C_1 is the best configuration for Q_3, whereas C_2 is the best configuration for Q_1.

Table 1.1: Communication times in high-bandwidth network.

	Q_1	Q_3
C_1	25.61	5.09
C_2	20.48	8.22

Table 1.2: Communication times for query Q_3.

	High-bandwidth	*Low-bandwidth*
C_1	5.09	452.16
C_2	8.22	66.49

(iii) To show that the best middleware configuration is *network-dependent*, we report in Table 1.2 the communication times (in seconds) for Q_3, in both high- and low- bandwidth networks. We observe that C_1 is the best configuration for the high-bandwidth network, whereas C_2 is the best for the low-bandwidth one.

□ ◊

To our knowledge, no existing distributed DBMS is able to automatically tune the middleware parameters, nor is able to adapt to different queries (that may vary in terms of selectivity and tuple size) and network environments (that may vary in terms of bandwidth). It is currently the task of the database administrators and programmers to manually tune the middleware to improve the system performance.

In this thesis, we present MIND (MIddleware tuNing by the Dbms), a framework for tuning the fetch size F and the message size M. Our approach is:

- *Automatic*, to alleviate the effort of database administrators and programmers.
- *Query-adaptive*, since every query has its own optimal middleware parameters.
- *Network-adaptive*, since every network has its own optimal middleware parameters.

To this purpose, we formalize and solve the problem of middleware tuning as an optimization problem:

Input: Query result of Q and network environment.
Output: Best values of middleware parameters F and M that minimize the time for communicating the query result over the network.

To the best of our knowledge, the database research community does not have well-established strategies for tuning the middleware parameters F and M. However, existing technical documentations e.g., [7, 40] put forward some recommendations, none of which being *query-* and *network-dependent*. Our experimental study shows that these strategies do not usually yield the best communication time in practice, even when they consume large resources by the middleware parameters F and M.

The distributed query processing mainstream literature [5, 10, 14, 17, 25, 29, 36, 37] typically focuses on designing distributed query plans that minimize the communication time. To this

purpose, they mainly rely on communication cost models where the total amount of data that needs to be communicated over the network is considered to have a major impact.

Our work is complementary to those cited in the previous paragraph because we focus on *how* a query result is communicated over the network, more precisely on *how to tune the middleware parameters in order to minimize the communication time of a query result.* Indeed, on the one hand, none of the aforementioned works take into account the interaction between the DBMS and the middleware, whereas on the other hand, we do not take into account the actual query plan that constructs a query result as we are interested only in *how* a query result can be optimally communicated over the network from the middleware point of view.

The state-of-the-art communication cost models in distributed databases area [14, 29, 36] have two components: (i) the *per-message cost* component i.e., the cost of constructing a message by the middleware of a data node, and (ii) the *per-byte cost* component i.e., the cost of communicating bytes of data over the network. Our experiments presented at Chapter 4 show that the estimation results returned by such communication cost models are not very accurate since the used estimation functions do not take into account the round-trips and the pipelining effects. To estimate the time needed to communicate a query result over the network, we develop a novel estimation function that is at the core of MIND. The proposed function takes into account the middleware parameters, the size of the query result, and the network environment. We show in Chapter 4 an experiment emphasizing the accuracy of the proposed estimation function.

An important point is that the low-level network parameters (e.g., part of the TCP/IP protocol [39, 42]) are not in the scope of our work. However, we take into account the network environment in our calibration phase, which allows to dynamically configure the weights of messages (*network-dependent* parameters) of our communication time estimation function. The parameter calibration achieved by MIND is in the spirit of the recent line of research on calibrating cost model parameters (for centralized DBMS) to take into account the specificities of the environment [18].

Also, we point out that the MIND framework is in the spirit of the research line on DBMS self-tuning [9, 27] and query-driven tuning [38]. However, we are complementary to such approaches as we allow the DBMS to tune external parameters (i.e., from the middleware level), which falls outside the scope of existing DBMS. There are also recent works on software-defined networking for distributed query optimization [43] that tune parameters outside the DBMS. We are orthogonal on such approaches since they tune the network bandwidth needed to communicate a query result, whereas we focus on tuning the middleware parameters.

We also emphasize that our middleware study is complementary to the distributed query processing mainstream literature [5, 14, 17, 25, 29, 36] in the sense that we investigate *how a query result is communicated over the network according to the middleware parameters* F and M and not how to design an optimal query plan communicating as less data as possible.

Main contributions of the thesis

The goal of this thesis is to present the design and an empirical study of the MIND framework. Our main contributions are as follows:

- We present an experimental study (Chapter 3) having as goal to emphasize that the middleware configuration has a crucial impact on the time of communicating query results, and that research efforts need to be made to integrate the parameters F and M into the DBMS optimizer. Our study is extensive in the sense that we did a total number of ∼43K tests,

spread over ∼7K distinct scenarios (two networks of different bandwidth × six queries of different selectivity × up to 629 different middleware configurations, depending on the result tuple size of each query). In particular, we show that the values of the middleware parameters F and M that minimize the communication time are *query-* and *network-dependent*. Moreover, we point out that none of the current recommendations found in technical documentations for tuning the middleware parameters is able to find the optimal values since such strategies do not take into account the query- and network-dependency.

- We introduce the MIND's function for estimating the communication time (Chapter 4). At the outset of our method are two crucial observations:
  - Batches and messages are communicated differently over the network: batches are communicated synchronously, whereas messages in a batch are communicated in pipeline (assuming that a batch has several messages), hence it is possible to exploit the pipelining for minimizing the communication time, and
  - Due to network latency, it is more expensive to communicate the first message in a batch compared to any other message that is not the first in its batch.

  These observations led to an estimation function where a message is treated differently depending on whether or not it is the first in its batch. Then, we propose an effective strategy for calibrating the costs (weights) of messages (first and non first in its batch), which are *network-dependent* parameters, of the communication time estimation function based on the actual network environment. We also show an experiments emphasizing the accuracy of our estimation.

- We develop an optimization algorithm to effectively compute the values of the middleware parameters F and M that minimize the communication time (Chapter 5). The proposed algorithm is iterative in the sense that it starts with initial (small) values of the two middleware parameters F and M and iterates to improve the estimation by updating the initial values. This allows us to quickly find (in small fraction of a second) values of the middleware parameters F and M for which the improvement in terms of communication time estimation between two consecutive iterations is insignificant. In practice, this translates to a good trade-off between low resource consumption and low communication time.

- We present an evaluation of the MIND framework (Chapter 5). In particular, we point out the improvement that we obtain over the current strategies for middleware tuning (in terms of communication time and/or resource consumption), the query- and network-adaptivity of MIND, and how the time estimation and the two middleware parameters F and M change during the iterations of the optimization algorithm.

Chapter 2

Communication cost in distributed data management systems

In this chapter, we briefly present different distributed DBMS architectures (Section 2.1) and we discuss cost models and techniques developed for optimizing communication time in distributed query processing (Section 2.2).

2.1 Distributed DBMS architectures

Distributed database management systems are at the convergence of two technologies: data processing and computer network technologies [36]. In this section, we focus on main distributed DBMS architectures proposed in the mainstream literature [25, 36]: *Client-Server* systems, *Peer-to-Peer* distributed DBMS and *data integration* (*mediated*) systems.

Client-Server architecture

A distributed DBMS architectures, as presented in Figure 2.1, are based on a general paradigm *Client-Sever* (or Master-Slave). This paradigm refers to a class of protocols that allows one node, namely *client*, to send a request to another node, called *server*, that processes the query and sends an answer as a response to this request. In this paradigm every node has a fixed role and always acts either as a client (*query source*) or as a server (*data source*) [25].

[36] considers that the general idea behind *Client-Server* architecture is that the query processing functions are divided into two classes: server functions and client functions. In fact, the query processing, optimization, transaction management and storage management are done at

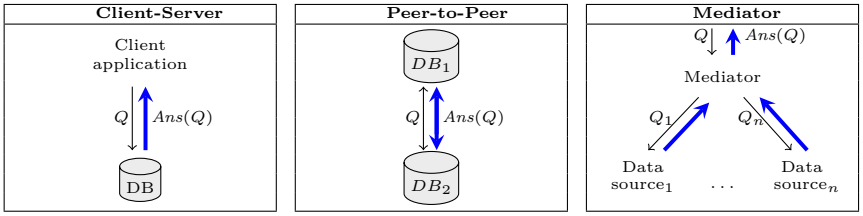


Figure 2.1: Main distributed architectures.

the server node. In addition to the application and the user interface, the client has a DBMS module, which is in charge of managing the cached data that is gathered from DBMS node (data server) and sometimes managing transactions.

Peer-to-peer architecture

As presented in Figure 2.1, in *Peer-to-Peer* architectures every node can act as a server that stores parts of the database and as a client that executes application programs and initiates queries [25]. [36] considers that the modern *Peer-to-Peer* systems go beyond this simple characterization and differ from the old *Peer-to-Peer* systems. The first difference is the massive distribution of data sources in current systems. The second difference is the inherent heterogeneity of distributed data sources and their autonomy. The third major difference is the considerable volatility of distributed data sources.

Data integration (mediated) architecture

In *virtual data integration* (*mediated*) systems, data stays at remote data sources and can be accessed as needed at query time [13]. As presented in Figure 2.1, these systems consist of integrating heterogeneous remote data sources in a virtual global schema, which is constructed at mediator node [36]. In this architecture, client node sends query to mediator node, which is in charge of constructing distributed query plan and decomposing query into subqueries that are sent to remote nodes. Each remote node processes the subquery and sends back results to the mediator node, which combines the results and send then back answer to the user.

2.2 Query optimization in distributed DBMS

The essence of *Client-Server* architecture is at the core of the aforementioned distributed architectures, in the sense that data is persistently stored in remote data servers and queries are initiated at client nodes [25]. In distributed DBMS architectures, the dominant cost of computing a distributed query plan is typically the cost of transferring (intermediate) results over the network [25, 36]. As a consequence, minimizing the communication time has been recognized for a long time as one of the major research challenges in distributed data management area [25, 36]. A long-standing research effort has been devoted to the investigation of this problem, which led to the development of numerous distributed query optimization techniques [5, 14, 17, 25, 29, 36].

Next, we present different DBMS cost models used in query optimizer (Section 2.2.1) and we discuss different techniques developed to minimize the communication time in distributed query processing (Section 2.2.2).

2.2.1 Cost models in distributed DBMS

The query optimizer is a central component of a DBMS, having as goal to compute for a given query an execution plan that reduces the response time. A key difficulty underlying query optimization lies in the design of an accurate cost model that is used by the optimizer to estimate the costs of candidate query plans. Centralized DBMS_s consider that the main components impacting query execution time is the *CPU* instructions and *I/O* operations. For instance, in a commercial open-source DBMS, namely *PostgreSQL*, the cost model used by the query optimizer consists of a vector of five components related to *CPU* instructions and *I/O* operations [18]. The cost of query plan is composed of the following five components:

- *CPU* instructions:
  - The *CPU* cost to process tuples.
  - The *CPU* cost to process tuples via index access.
  - The *CPU* cost to perform operations such as a hash or aggregation.
- *I/O* operations:
  - The *I/O* cost to sequentially access a page.
  - The *I/O* cost to randomly access a page.

Distributed DBMS_s optimizer, in addition to the cost components (*CPU* and *I/O*) of centralized DBMS_s, takes into account the cost of transferring query results over the network. The state-of-the-art communication cost models in distributed databases area [5, 14, 24, 29, 36] consider as major parameters impacting the communication cost: the volume of data to be transferred (called *per-byte* cost component) and number of messages constructed (called *per-message* cost component). Next we present cost models proposed in mainstream literature [5, 14, 24, 29, 36].

2.2.1.1 Cost models based on volume of data and constructed messages

This class of models consider that the communication cost can be estimated using two components:

- The *per-message cost* component i.e., the cost of constructing a message at a data node. Precisely, it consists of estimating the overhead due to the construction of a message by DBMS node. Hence, the total number of messages is an important parameter in such cost models [14, 29].
- The *per-byte cost* component i.e., the cost of transferring bytes of data over the network. Concretely, it is the needed time of communicating a unit of data (e.g., byte or packet) of query result via network channel. Hence, the total size of query result influences the estimated cost in such models [14, 29].

This cost model is at the core of many distributed query optimizer [14, 29, 36]. For instance, the popular R^* *optimizer* [29] considers the combination of the aforementioned cost components to estimate the communication time in distributed query plan. Precisely, it considers:

- The cost of initiating all messages of query result (*per-message* cost). This cost encodes the time needed to construct a message before communicating it over the network. It is estimated by dividing the approximate number of instructions to initiate and receive a message by the MIP rate (Million Instructions Per Second), which measures the number of machine instructions that a computer can execute in one second.
- The cost of transferring over the network the total bytes of query result (*per-byte* cost) in all messages over the network. This cost encodes the time needed to communicate the bytes of a given message over the network. This cost is estimated by taking into account the actual transmission speed of the network.

Also, [14] gives a formula to estimate the cost of sending and receiving a message from a data server node to client node for parallel query optimization. Concretely, this formula estimates the time for communicating a message as:

- The cost of constructing a data message before any data is placed onto network (*per-message* cost). This cost is considered a constant and estimated according to *CPU* cycles of used machines.
- The cost of communicating a message over the network (*per-byte* cost). The cost of communicating a message is computed according to the machine power (*CPU* cycles of the used machines).

It is worth noting that our experiments presented in Section 3.2 show that the estimation results returned by communication cost models based on *per-byte* and *per-message* are not very accurate because the used estimation functions do not take into account the round-trips and the pipelining effects.

Recently, in massively parallel systems, several cost models have being proposed in literature [8] in order to optimize system performances. In these systems, the bottleneck is the communication of data between computation nodes. This is due to the fact that query can be evaluated by a large enough number of servers such that the entire data can be kept in the main memory and network speeds in a large clusters are significantly lower than main memory access [5]. The recent research works focused on MapReduce framework has proposed the MapReduce Class ($\mathcal{MRC}$) to optimize the communication cost [1, 2, 5, 24] in massively parallel systems. This model considers the amount of data assigned to a computation node and the number of communication rounds for communicating intermediate results between computation nodes. However, it does not take into account the overhead due to message construction and transfer.

2.2.2 Techniques for optimizing communication cost

Many efforts from both academia and industry focused on developing techniques that minimize the total amount of data that needs to be communicated over the network [1, 2, 5, 14, 17, 24–26, 29, 36], while few research works considered the problem of optimizing the number of communicated messages [14, 25]. New emerging research works propose to use software defined networking for distributed query optimization [43].

Next we review different techniques that allow to minimize the time for communicating query result between computation nodes.

2.2.2.1 Query and data shipping techniques

There are three main approaches to minimize the communication overhead in distributed query processing [25].

- *Query shipping*, consists in shipping the query to the lowest possible level in a hierarchy of sites (database server site). In commercial distributed DBMS_s, the so-called hints are introduced to push operators (e.g., *selection, projection* and JOIN) to remote data nodes, which yield to reduce the communication cost [17].
- *Data shipping*, queries are executed at the client machine, where data is cached in main-memory or on disk at the client node.
- *Hybrid shipping*, is based on the combination of data and query shipping. *Hybrid shipping* approach, which provides a flexibility to execute query operators on client and server nodes,

presents in general better performance than *data* or *query shipping*. This is because *hybrid shipping* enables to exploit client and server as well as intra-query parallelism. However, query optimization is significantly more complex [25].

2.2.2.2 Techniques that reduce the volume of data

The popular techniques used to avoid communicating as less as possible of volume of data between computation nodes, are: filtering relations, fragmenting big tables and compressing data before being communicated over the network. The main ideas of these techniques are given bellow.

Semijoin technique:
[25, 29] consider that the semijoin operation can be used to decrease the total time of join queries. The semijoin acts as a size reducer for a relation much as a selection does. The join of two relations R and S over attribute A, stored at sites 1 and 2, respectively, can be computed by replacing one or both operand relations by a semijoin with the other relation, using the following rules:

- $R \bowtie_A S \Leftrightarrow (R \ltimes_A S) \bowtie_A S$
- $\Leftrightarrow R \bowtie_A (S \ltimes_A R)$
- $\Leftrightarrow (R \ltimes_A S) \bowtie_A (S \ltimes_A R)$

The choice between one of the three semijoin strategies requires estimating their respective costs. The use of the semijoin is beneficial if the cost to produce and send it to the other site is less than the cost of sending the whole operand relation and of doing the actual join.

Bloom-Filter technique: similar to *semijoin* technique, the *Bloom-filter* (also called *Bloom-join*) is a "hashed *semijoin*", in the sense that it filters out tuples that have no matching tuples in a JOIN operation [25, 29]. The main idea of *Bloom-filter* technique is to communicate as less data as possible from one site to another.

2.2.2.3 Reducing the number of communicated messages

Row-blocking (batching) technique

The main idea of this technique is to ship tuples, from one site to another via network, in block-wise fashion, rather than every tuple individually. This approach, implemented in commercial distributed DBMS, is obviously much cheaper than the naive approach of sending one tuple at a time because the data is communicated into fewer messages [25]. The technical documentations of popular DBMS drivers (e.g., JDBC [40], ODBC [15], and proprietary *middleware*: Oracle Net services [6, 35] and Distributed Relational Database Architecture: DRDA of DB2 [21]) emphasize the performance improvement obtained by communicating many rows at one time, which reduce the number of *round-trips* from client node to DBMS node. But, this is at the price of resource consumption, e.g., buffers and network bandwidth [40].

Despite the performance advantage provided by the *row-blocking* technique, to the best of our knowledge that there is not cost model proposed in the literature that takes into account the impact of *row-blocking* (*batching*).

In addition, other classical techniques have been used to reduce communicated volume of data and messages, such as vertical and horizontal partitioning for big relations [3, 31] and data compression [28].

2.2.2.4 Adaptive optimization

Recent research work [43] proposes to use the software-defined networking to provide an optimal network bandwidth. Precisely, this work points out that:

- Each query needs a particular network bandwidth to communicate efficiently its query result from DBMS node to client node.
- DBMS optimizer can tune external parameters, such that network bandwidth, which can be controlled via *Software Defined Network* (SDN) to fix the necessary bandwidth for each query [43].

We are orthogonal on such approach since it tunes the network bandwidth needed to communicate a query result, whereas we focus on tuning the *middleware* parameters.

Furthermore, we point out that our research work is in the spirit of the research line on DBMS self-tuning [9, 27], parametric optimization [23, 41] and query-driven tuning [38]. However, we are complementary to such approaches as we allow the DBMS to tune external parameters (i.e., from the *middleware* level), which falls outside the scope of existing distributed DBMS.

2.3 Discussion

At the end of this chapter, it is important to stress that despite the developed approaches and techniques (Section 2.2), minimizing the communication time in distributed query processing remains an open research domain. This is motivated by the fact that:

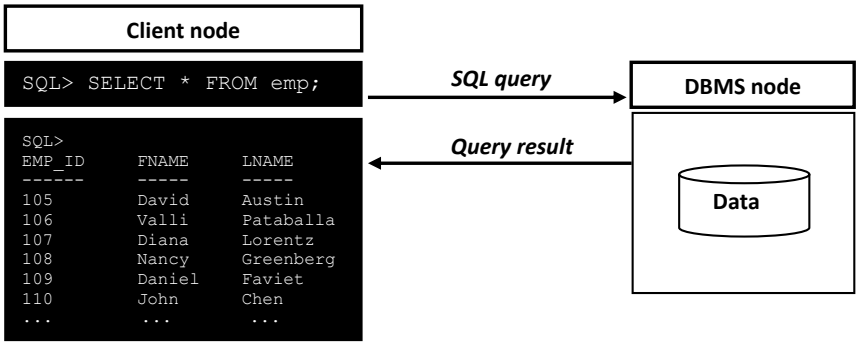


Figure 2.2: Communicating query result in a simplified *client-server* architecture.

- Cost models proposed in literature [5, 14, 29, 36], which are based on the *per-message* and/or *per-byte* cost components are not suited to tune the middleware parameters F and M because they do not take into account the round-trip and pipeline communication effects. In Section 3.2.4, we give experiment results that strengthen this point.
- Distributed query processing mainstream literature [1, 2, 5, 14, 17, 24–26, 29, 36] does not take into account the interaction between the DBMS and the middleware layer in distributed query processing. It typically focuses on designing distributed query plans that minimize the communicated volume of data and number of messages. The designed query plans do not take into account *how* query results are communicated between computation

nodes. However, we argue that it is important to look inside the middleware black-box to understand what happens. For instance, the execution of a distributed query in Figure 2.2 raises several questions, such as:

- How data is communicated from DBMS node to client node (*tuple-per-tuple*, *whole query result at once* or *batch by batch*)?
- How SQL middleware allocates memory in both DBMS and client nodes to manage the query result?
- How client node processes the communicated query result (in streaming manner, wait until receiving the whole query result, computing result batch per batch or with other manner)?
- What is the influence of the communication layers (e.g., middleware, *network protocol*, etc.) in the improvement of distributed query execution?
- What is the parameters that impact the time for communicating query result and how they can be suited to provide a good communication time?
- Whether the classical cost models *per-message* and *per-byte* components are suited to estimate the time for communicating query result between computation nodes?

In this thesis we focus on all these questions and we take a new look to the problem of optimizing the time for communicating query results in a distributed architectures, by investigating the relationship between the communication time and the middleware configuration.

Chapter 3

DBMS-tuned Middleware

In this chapter, we recall the main functionalities provided by SQL middleware (in Section 3.1) and then we present our experimental study that emphasizes the crucial impact of the middleware configuration on the time for communicating query results over the network (in Section 3.2).

3.1 Middleware in distributed query processing

The term middleware refers to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages [12].

In distributed DBMS architectures, the middleware (e.g., JDBC [40], ODBC [15], or proprietary middleware: *Oracle-Net-services* [6, 35] and Distributed Relational Database Architecture: DRDA of DB2 [21]) is a layer on top of a network protocol that determines, among others, *how* data is divided into batches and messages before being communicated from DBMS node to client node over the network (see Figure 3.1).

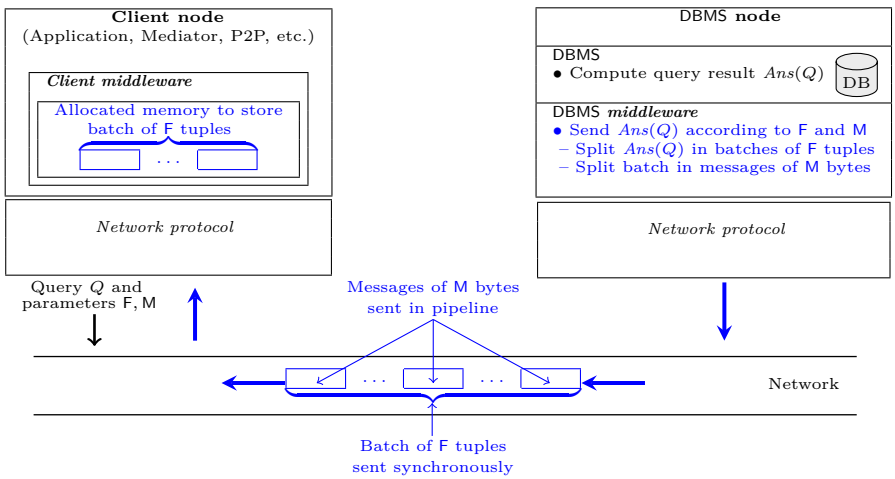


Figure 3.1: Communication model in simplified *Client-Server* architecture.

3.1.1 Middleware functionalities

The main functionalities provided by the SQL middleware in distributed DBMS_s architectures are:

- Establishing and managing connection between client and DBMS nodes. Indeed, the middleware is responsible to establish and keep alive connection between application and DBMS nodes throughout the query execution.
- Hiding heterogeneity problems (e.g., data types, synthetic variables of the query language, etc.) between client and DBMS nodes. That means providing a maximum of interoperability, in the sense that an application can access different heterogeneous DBMS_s with a single program code.

  For example, most SQL middleware implements SQL Call Level Interface (CLI) i.e., an API which provides standard functions to mainly send SQL statements to the DBMS and gather query results into application node. The goal of this interface is to increase the portability of applications by enabling them to become independent from particular DBMS [11].
- Determining *how* data is divided into batches of F tuples and messages of M bytes before being communicated over the network from DBMS node to client node, such as illustrated in Figure 3.1.
- Receiving and caching data into buffers before being processed by client application.

3.1.2 Communication model

We focus on a simplified distributed architecture (cf. Figure 3.1), where client node sends a query Q to DBMS node, which in its turn sends back to client node the result of a query Q, assuming that the result of Q is needed by some application from client node. Such an architecture is at the core of several distributed architectures presented in Section 2.1 such as:

- The *Client-Server*, where client node is a client and sends its query to the server at DBMS node, which sends back the query result.
- The *mediator*, where client node is a mediator, having as role to compute a distributed query plan and decomposing Q into subqueries; then, it asks distributed data nodes such as DBMS node to compute subquery results.
- The *peer-to-peer*, where every node can play at the same time the roles of client and server, etc.

A simplified architecture as in Figure 3.1 allows us to stress test the communication time by requiring to communicate over the network the entire query result that is computed in DBMS node. We focus on the impact of the middleware configuration on the time for communicating query result from DBMS node to client node. In particular, the middleware of DBMS node is in charge of splitting query result in batches of F tuples and then splitting each batch in messages of M bytes. The values of middleware parameters F and M can be tuned in all standard or DBMS-specific middleware [7, 15, 16, 30, 40].

In the rest of this section, we describe the standard behaviour of these parameters. An important point is that *batches and messages are communicated differently over the network*.

Synchronous communication of batches

When DBMS node sends a batch of F tuples to client node, the communication is done *synchronously* i.e., client node needs to wait an ack signal and a request for a next batch from the part of client node to be able to send the next batch of F tuples. Moreover, client node is able to process a batch only after receiving all messages composing that batch and it is blocked while waiting all such messages. After client node receives and processes an entire batch, it sends to DBMS node a request for a next batch.

The number of batches that are used to communicate a query result is known as the number of *round-trips* [40]. Whereas the size of a batch F is usually given in tuples, it is sometimes important to quantify the actual size in bytes of a batch, that we denote by F^B (this can be computed simply by multiplying F with the size in bytes of a tuple). For instance, client node needs to have F^B bytes of available heap memory to store an entire batch while receiving it.

Pipeline communication of messages

To send a batch of F tuples over the network, the middleware of DBMS node splits it into messages, each message having M bytes. The messages of M bytes are sent in *pipeline* from DBMS node to client node (assuming that a batch of F tuples has more than one message of M bytes).

More precisely, the middleware at DBMS node has a buffer of M bytes that it continuously filled with tuples of the current batch; each time the buffer is full, the middleware of DBMS node sends a message of M bytes over the network, until the whole requested batch is achieved. This means that DBMS node sends messages over the network without waiting for ack signals from client node, hence several messages of a same batch can be communicated over the network at a specific moment.

Low-level network communication

The messages of M bytes are further split into network packets by a low-level network protocol such as TCP/IP [39], that we do not represent in Figure 3.1 for simplicity of presentation and because such low-level protocols are out of the scope of our work. The technical documentation of some state-of-the-art DBMS [6, 7] recommends using a middleware message that fits in a network packet to avoid the overhead of fragmenting a middleware message of M bytes in several network packets. However, our extensive empirical study shows that such a strategy never gives the best communication time in practice, which suggests that the number of $round - trips$ and the network latency have the most dominant impact on the communication time.

3.1.3 Memory management

In this section, we focus on *how* middleware allocates and manages the query results communicated by DBMS node. Recall that the DBMS middleware communicates synchronously query results in one or several batches (i.e., many tuples that are communicated at once from DBMS node to client node. The number of tuples is fixed by a middleware parameter F). The middleware in client node is in charge to allocate a sufficient memory (buffers) to store an entire batch while receiving it, as depicted in Figure 3.1.

The popular JDBC middleware [20, 33] allocates buffers that should store query result or batch of F tuples according to:

- The number of columns and their types for a given query (to compute an approximate size per row, where each column type has a its size).
- The size of the query result (if the whole query result is gathered at once) or batch (if query result is split into several batches).It is recommended that the size of a batch should be configured carefully because it has an enormous impact on memory consumption and system performance.

The memory allocation in client middleware is very sensitive, in the sense that:

- Small memory size implies batch of small sizes (i.e., a small F) which lead to poor performance in communication time because the client node needs many *round-trips* to process the entire query result [40]. In this case, the main advantage is that the middleware consumes a less memory resource, which does not impact system performance in client node.
- Large memory size for a batch of F tuples, in presence of enough bandwidth network, can improve considerably the communication time. The drawback in this case is that the middleware consumes a large memory, which impacts the system performance at client node and can generate an errors when the available memory is not enough to store a batch of F tuples.

To the best of our knowledge, no existing DBMS middleware is able to find the best value of the middleware parameter F that provides a good trade-off between communication time and memory consumption.

Furthermore and in addition to the memory consumption in client node, it is important to stress that middleware parameters F has an important impact on the network bandwidth and I/O disk access. We report in Appendix A the consumption of network bandwidth and I/O disk access according to the values of middleware parameters F and M during query execution.

Next we present the current practices used in commercial DBMS_s to set the middleware parameters F and M.

3.1.4 Current practices for configuring middleware parameters

We focus on commercial distributed DBMS_s, namely: Oracle [34, 35], DB2 [19, 21, 22], PostgreSQL [16], SQL Server [1] and MySQL [2] to analyse how the middleware parameters F and M are configured.

These DBMS_s use different strategies to set the values of the middleware parameters F and M. These strategies are summarized in the following points:

- Leave default values set by the DBMS middleware.
- Leave F as default value and set M to maximum value.
- Set F to maximum value and leave M as default value.
- Set both parameters F and M to maximum values.

[1] https://technet.microsoft.com/en-us/library/aa342344%28SQL.90%29.aspx
[2] http://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-implementation-notes.html

- Set M to maximum value and set F such that all tuples in a batch fit in a single message.

Next we give more detail on the current practices used in these DBMS_s to set the values of the middleware parameters F and M.

Oracle

Batch Size F*:* Oracle JDBC middleware [34] sets the default F at 10 tuples and can be setted to maximum size that can be handled in the heap memory. Oracle ODBC[3] middleware allows to configure the amount of memory to contain a batch of F tuples. In ODBC, the default size to handle a batch is fixed to 64 KB. For an application $SQL - Plus$ command line, which uses a proprietary middleware (Oracle Net-Services), it sets a default F to 15 tuples and a maximum value to 5K tuples. Oracle considers that determining the best value of F is not obvious [4].

Message Buffer Size M*:* Oracle Net-Services [35] uses a session parameter, namely session data unit (SDU, which is the M in our case). This parameter defines the size of data message that can be used for communicating query results between DBMS and client nodes. The size of message is negotiated between client and DBMS nodes at the connection time. The message size can range from $0.5KB$ to $65KB$. The default message size is $8KB$, and rarely larger than $8KB$ bytes. When large amounts of data are being transmitted, increasing the message size can improve performance and network throughput. However, technical documentations [6, 7] consider that the M should be less than the network packet limited by the parameter Maximum Transmission Unit MTU.

DB2

DB2 has two strategies to set the middleware parameters F and M [19, 21, 22]. The first, called a *limited block fetch*, which is a default strategy. This strategy recommends to set the values of the middleware parameter F such that it fits in a one data message of M bytes. The second strategy, called *continuous block fetch*, which consists of setting the middleware parameter F such that it fits in several messages of M bytes. More detail on both strategies is given bellow.

Limited block fetch: consists of setting F such that it fits in one message of M bytes. The sizes of the F and M are fixed via parameter RQRIOBLK. The default value is $32KB$ and uses range of values from $4KB$ to $65KB$. This strategy consumes less resources, but provides a worst communication time when a large query result is communicated. This is due to the fact that a large number of communicated batches (*round-trips*) is done synchronously (*batch per batch*) between DBMS and client nodes, since each batch of F tuples is communicated in one message of M bytes.

Continuous block fetch: consists of setting F such that it can be fitted in several messages of M bytes. This strategy shows a good communication time due to the pipelined communication, of messages in the same batch, between DBMS and client nodes. This is due to the fact that a DBMS node does not wait a signal from client node to send next messages of a current batch. However, DB2 client can receive multiple messages of M bytes from DBMS node for each batch of F tuples. This strategy needs a large memory allocation in client node. The number of messages of M bytes (called extra blocks in IBM documentation) is fixed by the parameter EXTRA BLOCKS SRV in server node and MAXBLKEXT in client node. The maximum messages of M bytes that can be communicated for a batch of F tuples is limited at 100 messages.

[3] https://docs.oracle.com/cd/B28359_01/server.111/b32009/app_odbc.htm#UNXAR403
[4] http://docs.oracle.com/cd/B25221_04/web.1013/b13593/optimiz011.htm

SQL Server

Batch Size F*:* the default configuration of Microsoft JDBC Driver [5] is to retrieve the whole query result from DBMS node at once. If a large query result is gathered SQL Server can provide an adaptive buffering, but this can show an OutOfMemoryError in the JDBC application for the queries that produce very large results.

Message Buffer Size M*:* Sql Server [6] uses a default packet size of $4KB$. The message size M is very sensitive in Sql Server DBMS, since it can only tuned by an experienced database administrator or certified Sql Server technician.

PostgreSQL

Batch Size F*:* the default configuration of F in PostgreSQL [7] is to collect all query results at once when the memory can handle it. For a large query results, the middleware (e.g., JDBC) can extract only a small sizes of F tuples from DBMS node.

Message Buffer Size M*:* PostgreSQL uses a message-based protocol [8] for communicating query result from DBMS node to client node. In this kind of protocol, the size of message M is not fixed at session time, but each message contains its size in message-header.

MySQL

Batch Size F*:* by default MySQL [9] retrieves all query result at once, which is stored in client middleware buffers. For queries handling a large query results, the middleware can extract small batches from DBMS node (e.g., JDBC driver uses *setFetchSize* method to set the F).

At the end of this section, we stress that:

- The commercial DBMS_s do not provide methods and techniques to efficiently set middleware parameters F and M.
- The middleware parameters F and M are configured in commercial DBMS independently, in the sense that there is not a clear correlation (relationship) between the values of middleware parameters F and M, since a batch of F tuples is communicated in one or several messages of M bytes.
- The commercial DBMS_s tune the values of middleware parameters F and M such that they are used for all queries and network environments. This approach can not be useful because there is not a unique optimal configuration of middleware parameters F and M that fits all queries and network environments, because these parameters are *query-dependent* and *network-dependent*.

[5] https://technet.microsoft.com/en-us/library/aa342344%28SQL.90%29.aspx
[6] https://technet.microsoft.com/en-us/lib rary/ms177437.aspx
[7] https://jdbc.postgresql.org/documentation/head/query.html
[8] https://www.postgresql.org/docs/9.3/static/protocol.html
[9] http://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-implementation-notes.html

3.2 Analysing the impact of middleware parameters

In this section, we present an experimental study emphasizing that the middleware configuration has a crucial impact on the time of communicating query results. We present the considered distributed architecture in Section 3.2.1, the experimental setup in Section 3.2.2, the trade-off between performance and resource consumption in Section 3.2.3 and we discuss our empirical observations in Section 3.2.4.

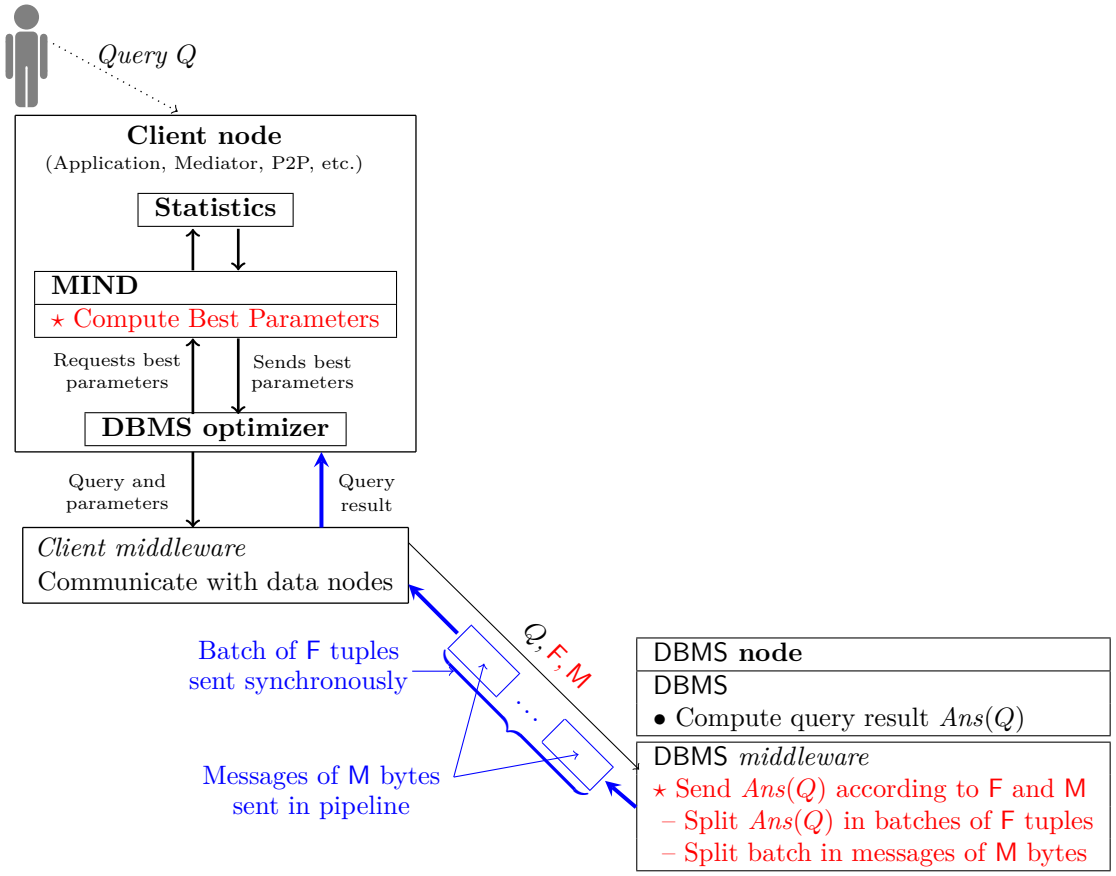


Figure 3.2: Architecture for DBMS-tuned Middleware framework (called MIND).

3.2.1 Architecture

We focus on a simplified distributed architecture (cf. Figure 3.2), which is at the core of several distributed paradigms [25, 36] such as: the *Client-Server* (where client node is a client and sends its query to the server at DBMS node, which sends back the query result), the *mediator* (where client node is a mediator, having as role to compute a distributed query plan and decomposing Q into subqueries; then, it asks distributed data nodes such as DBMS node to compute subquery results), the *peer-to-peer* (where every node can play at the same time the roles of client and server), etc.

Id	*Query*	*Result size (# of tuples)*	*Tuple size (bytes)*	*Result size (bytes)*
Q_1	`SELECT * FROM source`	165M	$205B$	$32GB$
Q_2	`SELECT objectid, sourceid, ra, decl, taimidpoint, psfflux FROM source`	165M	$54B$	$9GB$
Q_3	`SELECT objectid, taimidpoint, psfflux FROM source`	165M	$27B$	$4.5GB$
Q_4	`SELECT * FROM object`	4M	$450B$	$2GB$
Q_5	`SELECT objectid, ra_ps, decl_ps FROM object`	4M	$29B$	$0.1GB$
Q_6	`SELECT objectid, taimidpoint, psfflux FROM source WHERE objectid=433327840429162`	55	$27B$	$1.5KB$

Figure 3.3: Six real-world astronomical queries.

The components annotated with $\star$ in Figure 3.2 correspond to the middleware tuning features proposed by our framework that we detail in Chapter 4 and Chapter 5.

A simplified architecture as in Figure 3.2 allows us to stress test the communication time by requiring to communicate over the network the entire query result that is computed in a DBMS node. We focus on the impact of the middleware on communicating the query result $Ans(Q)$ from DBMS node to client node. In particular, the middleware of DBMS node is in charge of splitting $Ans(Q)$ in batches of F tuples and then splitting each batch in messages of M bytes.

Recall that the middleware parameters F and M can be tuned in virtually all standard or DBMS-specific middleware [7, 15, 16, 30, 40]. We recall also that the standard behaviour of communicating query results in batches of F tuples and messages of M bytes is described in Section 3.1.2.

3.2.2 Experimental environment

Data and queries

We focus on data and queries provided by researchers from the astronomical domain. More precisely, the dataset consists of astronomical data from LSST[10] (Large Synoptic Survey Telescope). The dataset has $\sim 34GB$ spread over two tables: `OBJECT` of objects detected in the sky ($\sim 2GB$, $\sim$4M tuples, 227 attributes) and `SOURCE` of sources used in the detection of these objects ($\sim 32GB$, $\sim$165M tuples, 92 attributes). More information on the schema of these tables can be found online[11]. Moreover, we focus on six queries (given in Figure 3.3) that we extracted from existing workloads that astro-physicists frequently run on astronomical data[12]. The only criteria that we took into account when choosing these queries is that they have diverse selectivities (their query results span from a few bytes to $32GB$). We are not interested in the actual query plan that computes their query results as we are interested only in how their query results are communicated over the network.

Database

Commercial distributed relational database system DBMS is used to store and manage dataset in tables.

[10] `http://www.lsst.org/lsst/`
[11] `https://lsst-web.ncsa.illinois.edu/schema/index.php?sVer=PT1_1`
[12] `https://dev.lsstcorp.org/trac/wiki/db/queries/ForPerfTest`

System environment

Our machines have CPU Xeon E5-2630 at 2.4 *GHz*, 8*GB* of memory, 250*GB* of hard drive (at 10K revolutions per minute), and run Ubuntu 12.04.

Network configurations

We use two configurations: (i) *high-bandwidth* (10*Gbit/s*) and (ii) *low-bandwidth* (50*Mbit/s*). In both cases, we have an Ethernet MTU (Maximum Transmission Unit) jumbo frame of 8.95*KB* (that is the size limit of a network packet that can be sent over our networks).

F and M configurations

We consider a set of $629 = 17 \times 37$ configurations as follows:

- For M we have 17 values: 1.5, and from 2 to 32*KB* with a step of 2. We have a minimum value of 1.5*KB* to simulate a standard Ethernet MTU that is usually fixed to 1.5*KB*.

  Moreover, we have a maximum value 32*KB* for M because this is the maximum supported by the middleware of the commercial DBMS that we used.

- For F we have 37 values:

  - 9 values from 110 to 990 tuples with a step of 110,
  - 9 values from 1.1K to 9.9K tuples with a step of 1.1K,
  - 9 values from 11K to 99K tuples with a step of 11K, and
  - 10 values from 110K to 1.1M tuples with a step of 110K.

  We considered multiples of 110 tuples for F because 1.1K tuples can be communicated in one message of 32*KB* corresponding to the maximum M for query Q_3 (that we consider to be the "average" query since both its result size of 165M tuples and its tuple size of 27 bytes are the most frequent among all queries; Q_3 is actually the running example query throughout the thesis).

  We have a maximum value of 1.1M tuples for F because this occupies the maximum heap space allowed for a middleware batch by our system (that is 30*MB*). We also point out that for the queries Q_1, Q_2, Q_4 having visibly larger tuples compared to Q_3 only a subset of the 629 configurations have been run successfully (since the largest batches that can be used for the three aforementioned queries are of 110K, 550K, 66K tuples, respectively).

Measures

We report the time needed to communicate the result of a query Q between two nodes, for a given network environment, a given F, and a given M. Each number reported for the high-bandwidth configuration is obtained as the average over ten runs, whereas for the low-bandwidth configuration over three runs. The reported communication times are provided by the DBMS trace files, which additionally provide information on the query execution time, number of transferred messages, etc.

In Table 3.1, we zoom on the main information gathered from the DBMS trace files in each query execution. The meaning of each column is explained bellow:

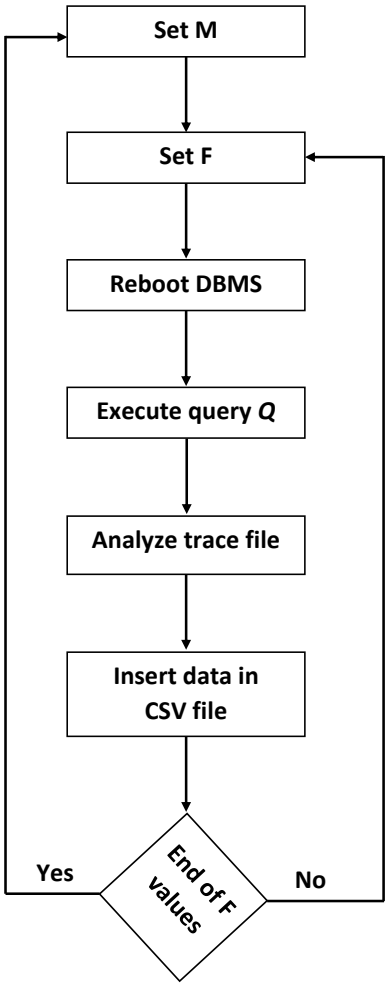


Figure 3.4: How query is executed in experimental process?

(i) Trace file name: reports details on how query execution is done;

(ii) M value: the size in bytes of middleware buffer, which corresponds to the amount of data that can be communicated at once from the middleware to the network;

(iii) F value: the number of tuples in a batch that is communicated at once;

(iv) Elapsed time: the query execution time in seconds;

(v) CPU time: the consumed time, in seconds, by CPU in the query execution;

(vi) Number of fetch operations (called also in the literature $round - trips$);

(vii) Number of rows in query result;

(viii) Number of first messages communicated;

Table 3.1: Zoom on information gathered from DBMS trace files, according to experimental process described in Figure 3.4.

(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	...
...	...	...	...	...	...	...	...	...	...	...	...
00.trc	32767	110	1325.75	454.54	1499907	164989583	1499907	944.47			...
81.trc	32767	220	888.02	351.12	749954	164989583	749954	576.49			...
84.trc	32767	330	681.64	306.15	499970	164989583	499970	403.64			...
45.trc	32767	440	595.93	295.84	374978	164989583	374978	324.27			...
99.trc	32767	550	531.79	269.36	299983	164989583	299983	281.26			...
68.trc	32767	660	530.43	273.86	249986	164989583	249986	273.65			...
08.trc	32767	770	504.40	270.43	214274	164989583	214274	249.79			...
47.trc	32767	880	483.59	262.58	187490	164989583	187490	234.39			...
80.trc	32767	990	464.04	258.10	166658	164989583	166658	219.13			...
18.trc	32767	1100	440.02	252.18	149992	164989583	149992	198.87			...
69.trc	32767	2200	320.27	220.29	74997	164989583	74997	104.80	74995	1.14	...
86.trc	32767	3300	257.06	193.55	49998	164989583	49998	65.85	99992	1.99	...
92.trc	32767	4400	209.85	164.32	37499	164989583	37499	44.93	112327	3.71	...
86.trc	32767	5500	202.03	159.70	30000	164989583	30000	38.76	104739	6.04	...
86.trc	32767	6600	187.25	152.80	25000	164989583	25000	31.81	112281	5.07	...
78.trc	32767	7700	167.31	139.42	21429	164989583	21429	25.11	117656	4.94	...
72.trc	32767	8800	161.47	137.28	18750	164989583	18750	21.98	121632	4.21	...
64.trc	32767	9900	157.54	132.93	16667	164989583	16667	21.40	123887	5.28	...
56.trc	32767	11000	153.32	129.98	15001	164989583	15001	18.70	119715	6.49	...
49.trc	32767	22000	134.66	119.26	7501	164989583	7501	10.02	126506	6.99	...
38.trc	32767	33000	129.55	116.75	5001	164989583	5001	6.89	127155	7.39	...
26.trc	32767	44000	131.45	115.97	3751	164989583	3751	5.40	128640	6.53	...
15.trc	32767	55000	128.10	117.18	3001	164989583	3001	4.30	128543	7.88	...
01.trc	32767	66000	125.80	115.58	2501	164989583	2501	3.59	129134	7.83	...
11.trc	32767	77000	124.90	117.12	2144	164989583	2144	3.08	129039	5.70	...
00.trc	32767	88000	123.55	114.18	1876	164989583	1876	2.67	129206	7.96	...
94.trc	32767	99000	125.77	118.31	1668	164989583	1668	2.46	129261	6.04	...
80.trc	32767	110000	122.60	114.56	1501	164989583	1501	2.20	129325	6.98	...
72.trc	32767	220000	128.09	122.75	751	164989583	751	1.16	129739	5.08	...
58.trc	32767	330000	130.91	123.74	501	164989583	501	0.80	129826	7.16	...
46.trc	32767	440000	125.73	123.36	376	164989583	376	0.62	129918	2.45	...
35.trc	32767	550000	125.61	124.05	301	164989583	301	0.50	129933	1.50	...
22.trc	32767	660000	122.84	120.56	251	164989583	251	0.42	129913	2.60	...
10.trc	32767	770000	121.12	118.89	216	164989583	216	0.37	130012	2.31	...
80.trc	32767	880000	121.36	118.90	189	164989583	189	0.33	129941	2.48	...
86.trc	32767	990000	122.22	119.46	168	164989583	168	0.30	130004	3.21	...
73.trc	32767	1100000	121.98	116.93	151	164989583	151	0.27	129934	3.74	...
...	...	...	...	...	...	...	...	...	...	...	...

(ix) Time, in seconds, for communicating all first messages in all batches;

(x) Number of not first messages in all batches;

(xi) Time, in seconds, for communicating all not first messages in all batches.

For an extensive study of the influence of the middleware parameters F and M, we develop a shell program on Linux OS that executes and analyses continuously for different values of middleware parameters F and M. The mains tasks of this program are shown in Figure 3.4.

3.2.3 Trade-off between performance and resource consumption

Unsurprisingly, an as it can be observed in our intensive experimentation (cf. 3.2.2), the cost of shipping query results is very sensitive to the network and middleware settings. Poor configurations of these two layers can increase significantly the communication cost.

We discuss the trade-off between performance and resource consumption for extreme cases of F and M:

- Large value of F minimizes the number of $round-trips$ (which minimizes the communication time in a network having a sufficiently large bandwidth), but increases the memory footprint (since all messages of a same batch need to be stored before processing them), and also increases the waiting time in Client node (since it is blocked while waiting for the entire batch);
- Small value of F does not consume large resources (and also avoids potential out-of-memory errors), but has the disadvantage of a large number of $round-trips$ (which increases the communication time);
- Large value of M gives empirically good results, particularly when it is combined with a large F because this implies multiple messages in a batch that are communicated in pipeline, which reduces the communication time, but at the price of increasing memory footprint in both nodes and increasing network usage;
- Small value of M does not consume large resources and usually yields a pipelining implying good communication times, but sometimes implies an overhead due to splitting batches in a large number of messages in DBMS node and reconstructing the batches in Client node. Our experiments confirm the intuitions outlined in this paragraph.

3.2.4 Empirical analysis

The goal of this section is to emphasize the impact of the middleware parameters F and M on the time of communicating a query result from DBMS node and client node. In particular, we show that:

- Two different middleware configurations can yield very different communication times;
- For a fixed network configuration, each query has a different best combination of F and M (i.e., the best middleware configuration is *query-dependent*);
- For a fixed query, each network configuration has a different best combination of F and M (i.e., the best middleware configuration is *network-dependent*);
- There is no current strategy of middleware tuning that allows to find the best parameters.

We next detail all these points.

Impact of the middleware configuration

To emphasize the impact of the middleware configuration on the communication time of a query result, we report in Figure 3.5 the best and worst communication times among 629 combinations of F and M parameters, six queries, and two networks (cf. Section 3.2.2). The difference between the best and the worst configuration is particularly visible for the queries having a large result. For example, for the queries Q_1, Q_2, and Q_3 that return the largest results, the difference is of two orders of magnitude in the high-bandwidth network and an order of magnitude in the low-bandwidth network. Moreover, we observe in Figure 3.6 that for the first five queries the communication times for the different combinations of F and M are not clustered neither around the best case nor around the worst case, but are rather spread over the entire space between them.

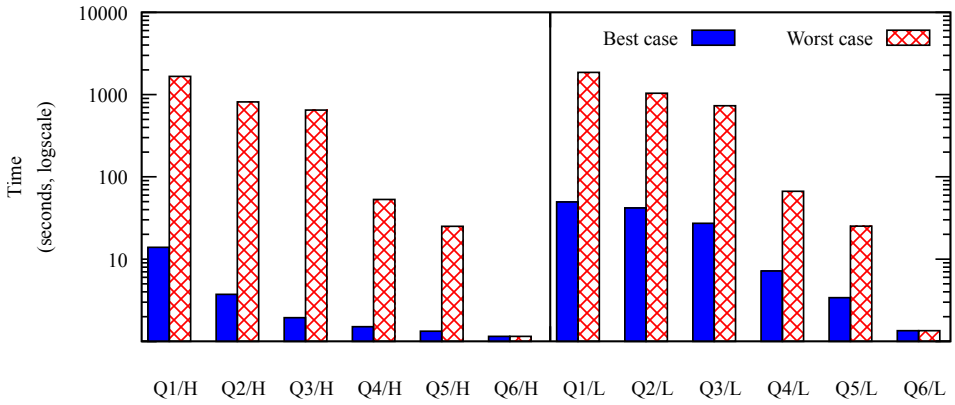


Figure 3.5: Best and worst communication times in (H)igh and (L)ow bandwidth networks for six queries (cf. Section 3.3).

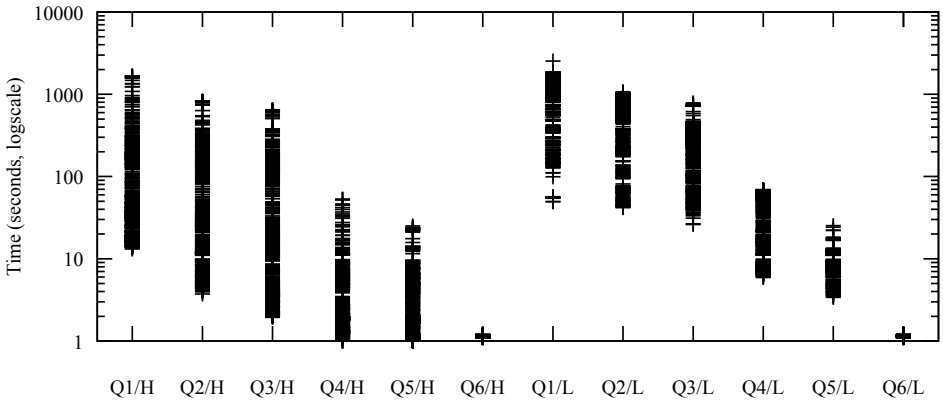


Figure 3.6: Repartition of communication times for all configurations (F and M) and for six queries (cf. Section 3.3).

Query-dependency

We present in Figure 3.7 the resources consumed by the best and worst cases from Figures 3.5, 3.6. In particular, we observe that all queries have pairwise distinct combinations of F and M that yield the best communication time. This observation holds for both network configurations. For example, if we look at Q_3 and Q_5 in the high-bandwidth network, we notice that both best cases use M=32KB, but the query Q_3 that has a much larger result also needs a larger F to obtain its best time (it needs F=880K tuples compared to Q_5 that needs F=330K tuples).

Network-dependency

In the same Figure 3.7, we observe that, for the first five queries, the two network configurations have different combinations of F and M values that give the best result. For example, for Q_3 we observe that the low-bandwidth network requires a much smaller F (880K vs 22K tuples) due to the network latency.

	High-bandwidth network				Low-bandwidth network			
	Best case		Worst case		Best case		Worst case	
	F	M	F	M	F	M	F	M
Q_1	110K	32	110	22	22K	32	4.4K	2
Q_2	550K	32	110	8	7.7K	32	33K	4
Q_3	880K	32	110	10	22K	24	110	4
Q_4	66K	32	110	32	1.1K	30	66K	2
Q_5	330K	32	110	6	3.3K	22	110	10
Q_6	110	1.5	110	1.5	110	1.5	110	1.5

Figure 3.7: Resources consumed by the bars in Figure 3.5.

Limitations of current strategies

As already mentioned in this chapter, to the best of our knowledge, the database research community does not have well-established strategies for middleware tuning. However, the documentation of the state-of-the-art DBMS e.g., [7, 40] puts forward some recommendations. We next point out that none of these recommended strategies is able to capture the aforementioned query- and network-dependency in order to find optimal values for the middleware parameters F and M.

In this experiment, we zoom on queries Q_3 and Q_5 (that have different query result sizes), and we report the communication times for four scenarios, obtained by crossing the two queries with the two considered network configurations (high- and low-bandwidth). We discuss the following five strategies, introduced in Section 3.1.4, for setting the values of middleware parameters F and M.

(i) Leave default values set by the middleware of our DBMS (in our case F=15 tuples and M=8KB).

(ii) Set M to maximum value (32KB) and leave F as default.

(iii) Set F to maximum value and leave M as default.

(iv) Set both parameters to maximum values.

(v) Set M to maximum value and set F such that all tuples in a batch fit in a single message.

In our case, we have for both queries Q_3 and Q_5 the same F=1.1M tuples in (iii) and (iv), and the same F=1.1K tuples for (v) because the two queries have similar tuple sizes (cf. Figure 3.3). The five aforementioned strategies correspond, in order, to the first five bars of each plot from Figure 3.8. The default M (used in the strategies depicted in the first and third bar of each plot) is set such that a middleware message of M bytes fits in one network packet (whose size is limited by the MTU, which is set to 8.95KB in our system as mentioned in Section 3.2.2). Such a default choice is suggested as being "optimal" by a state-of-the-art DBMS [7] to avoid the overhead due to splitting the middleware messages into network packets.

We observe that:

- There is no strategy that imposes itself as the best choice in all scenarios.

- The best combination of F and M as found after an exhaustive search over 629 combinations (i.e., the sixth bar of each plot) is always better than the best competing strategy.
- For each scenario, the worst performance is obtained by the strategies using a default (small) F value because this implies a very large number of $round-trips$ (cf. Section 3.2.1); this suggests that the commercial DBMS that we use has chosen such a default F in order to sacrifice the performance in the favor of less resource consumption (and to avoid out-of-memory errors).
- For the high-bandwidth network, the combination of the parameters F and M that yields the best communication time consumes less resources than the best competing strategy for Q_3 and Q_5.
- Setting the middleware message M such that it fits in one network packet never gives the best communication time (this last point is in fact noticeable in Figure 3.7 for all combinations of queries and networks).

Limitations of classical communication cost models

As mentioned in Section 2.2, state-of-the-art cost models in distributed data management area are based on *per-byte cost* and *per-message cost* components to estimate the communication cost [14, 29, 36]. Recall that intuitively, the per-message cost encodes the time needed to construct a message before actually communicating it over the network, whereas the per-byte cost encodes the time needed to actually communicate the bytes of a given message over the network. Our experiments show that such models are not precise enough to accurately estimate the communication time. We stress in Example 3.1 the limitation of this cost model for tuning middleware parameters F and M.

Example 3.1. We zoom on query Q_3 and we report in Figure 3.9(a) the communication times and in Figure 3.9(b) the number of communicated messages for four middleware configurations, namely F=1.1K, 2.2K, 3.3K and 4.4K tuples, with M=32KB. Figure 3.9(b) depicts the total number of messages with black bar, the total number of 1^{st} messages in batches with slash bar, and the total number of non first messages (i>1) in batches with backslash bar.

Observe that the four configurations transfer the same amount of data (4.5GB corresponding the result size of Q_3) using the same number of messages (150K messages as shown by the black bar of Figure 3.9(b)). In this case, classical *per-message* and *per-byte* cost models [14, 29, 36] tend to derive the same communication cost for the four configurations which is contradicted by the measures reported at Figure 3.9(a). The reason behind this discrepancy comes from the fact that existing cost models do not take into account the round-trip effect while, as it can be observed at Figure 3.9(b), the communication time is tightly correlated to the number of 1^{st} messages communicated in all batches (which corresponds to the number of round-trips). □
◊

Network overhead

Recall that the middleware messages of M bytes are further split into network packets by a low-level network protocol such as TCP/IP [39]. The documentation of some state-of-the-art DBMS [6, 7] puts forward a recommendation that suggests using a middleware message of M bytes that fits in a network packet. The goal of such an approach is to reduce the overhead due to the fragmentation of a middleware message in several network packets. However, our experiments

show that such a recommendation never gives the best communication time in practice, which suggests that the number of *round-trips* (that is equivalent to the *number of first messages*) has the most dominant impact on the communication time. In the experiment that we present in Example 3.2, we analyze the impact of the number of round-trips and the pipeline communication of middleware messages of M bytes, and the effect of network overhead due to fragmentation of M into network packets. Recall that the maximum size of a network packet that can be sent over a network is limited by the MTU parameter (whose size is limited to $8.95KB$ in our network).

Example 3.2. Take query Q_3 in the high-bandwidth network, using ten configurations obtained by crossing values of F (1.1K and 110K tuples), and M (2, 4, 8, 16, and $32KB$). We report the communication times in Figure 3.10(a), the number of communicated messages for F=1.1K in Figure 3.10(b) and for F=110K tuples in Figure 3.10(c). Moreover, Figure 3.10(d) and Figure 3.10(e) present the number of fragmentation operations of middleware messages of M bytes into network packets, which is defined as $\lceil \frac{\mathsf{M}}{\mathrm{MTU}} \rceil - 1$.

In this example, we make two important points: (i) the effect of the fragmentation of middleware messages of M bytes into network packets, and (ii) the subtle interaction between the values of the middleware parameters F and M.

Effect of fragmenting M *into network packets.* We show below that the overhead due to the fragmentation of middleware messages into network packets is not a dominant cost.

- In configurations using F=110K tuples, the best communication time is obtained with M=$32KB$, as depicted in Figure 3.10(a). Note that this case (F=110K tuples and M=$32KB$) presents a maximum network fragmentation operations (see Figure 3.10(e)).
- In configurations using F=1.1K tuples, the best communication time is obtained with M=$2KB$, as depicted in Figure 3.10(a). The documentation of some state-of-the-art DBMS [6, 7] recommends setting the middleware message size M such that it fits in a single network packet. Such documentation claims that this approach is optimal because setting $\mathsf{M} \leqslant$ MTU avoids the overhead due to splitting the middleware messages into network packets. However, we observe that such a recommendation does not fully capture the practical behavior. More precisely, we consider three values of M i.e., 2, 4, $8KB$ that are smaller than the MTU. In all three cases, although there is no cost associated to the fragmentation of M into network packets (cf. Figure 3.10(d)), the overall communication time varies from 64.98 seconds (for M=$2KB$) to 78.87 seconds (for M=$8KB$).

Subtle interaction between the values of F *and* M. Observe that there is a subtle interaction between the values of F and M. For example, with F=110K tuples, the best communication time is obtained using M=$32KB$, which is rather natural: we consume maximum resources of F and M, and we observe low communication time. On the other hand, for F=1.1K, the best communication time is obtained with M=$2KB$, which may appear unnatural because we use small message size, which increases the total number of messages. The rational behind such a behavior is that when smaller messages are sent through network of a large enough bandwidth, the data can be communicated in pipeline (i.e., several messages can be at the same time in the pipeline between DBMS node and client node) and, in addition, the cost of transferring the first message is cheaper when M is small. Recall that we detailed the pipeline phenomenon in Section 3.1.2. □ ◊

Conclusions of experimental study and research problem statement. The experiments detailed in this section show that there is no single combination of F and M that is optimal

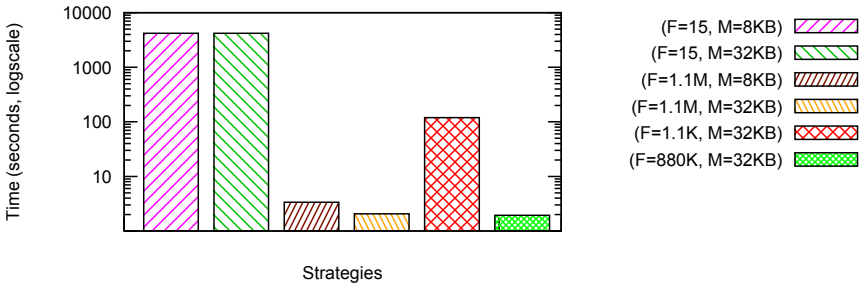
for all the considered queries and networks (even when large resources are consumed). This motivates our work on the following research problem: *given a query result size and a network environment, what is the best trade-off between the middleware parameters* F *and* M *in order to minimize the communication time of transferring the query result over the network?* We present our techniques to solve this problem in the following chapters.

3.3 Discussion

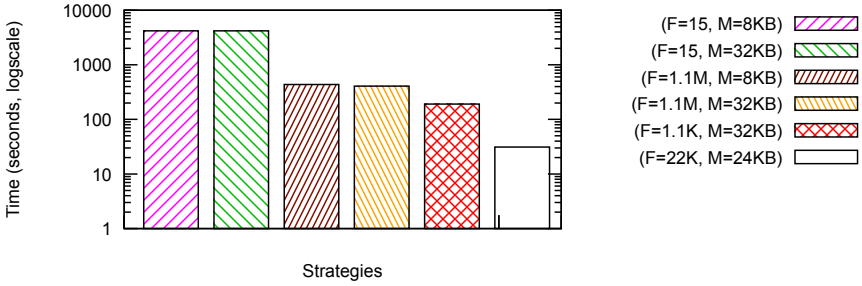
At the end of this chapter, we stress that:

- Tuning the middleware parameters F and M is non trivial problem because the optimal values of the parameters are *query-dependent* (that may vary in terms of selectivity) and *network-dependent* (that may vary in terms of bandwidth).
- The classical communication cost models (based on *per-byte* and *per-message* cost components) can not estimate accurately the time for communicating query results between distributed computation nodes. This is because they do not capture the practical behavior of communicating query results over the network. Precisely, they do not take into account the *round-trip* and *pipeline communication* effects.
- To the best of our knowledge, no existing DBMS is able to automatically tune the middleware parameters F and M, nor is able to adapt to different queries (that may vary in terms of selectivity) and network environments (that may vary in terms of bandwidth). It is currently the task of the database administrators and programmers to manually tune the middleware to improve the system performance.
- The database research community does not have well-established strategies for middleware tuning. However, existing technical documentations e.g., [7, 40] put forward some recommendations, none of which being query- and network-dependent. Our intensive experimental study shows that these strategies do not usually yield the best communication time in practice, even when they consume large resources.

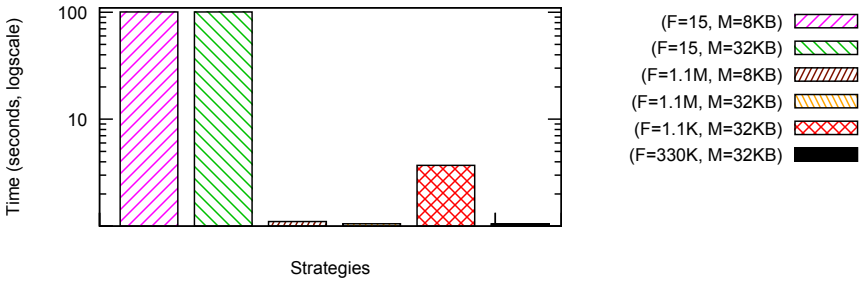
We present next our solution that is based on MIND framework, which allows to find the good values of middleware parameters F and M that minimize the communication time for a given query and network environment.



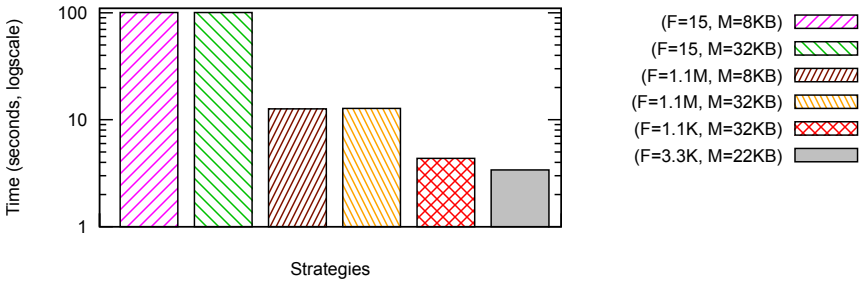
(a) Q_3 in high-bandwidth network configuration.



(b) Q_3 in low-bandwidth network configuration.

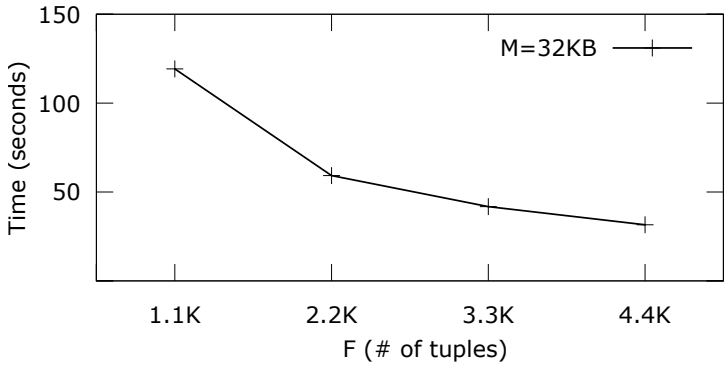


(c) Q_5 in high-bandwidth network configuration.

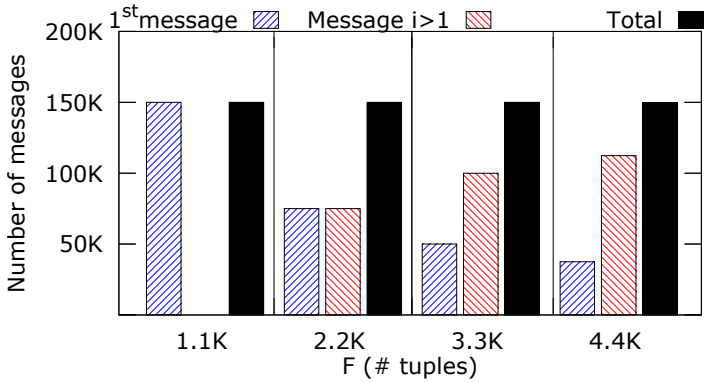


(d) Q_5 in low-bandwidth network configuration.

Figure 3.8: Communication times in different strategies for Q_3 and Q_5 in high- and low-bandwidth networks.

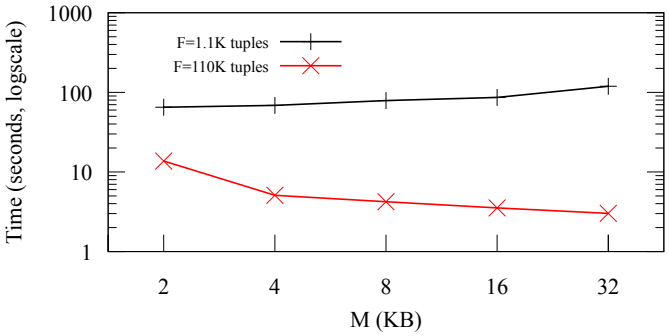


(a) Communication time for each configuration.

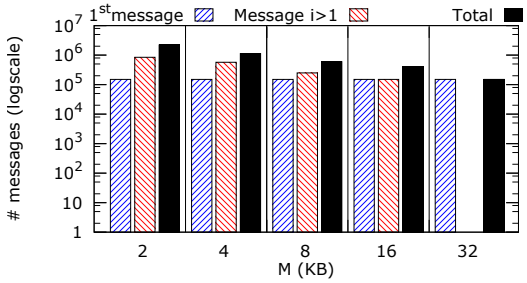


(b) Data messages communicated from data node to client node.

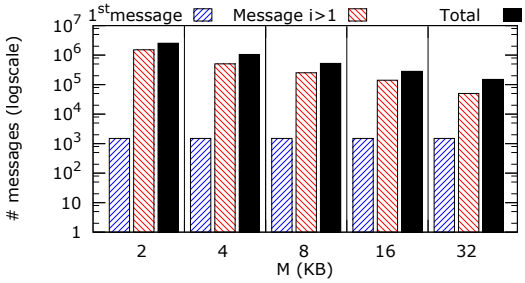
Figure 3.9: Zoom on Q_3 and four configurations to show the influence of first and non first messages (i>1).



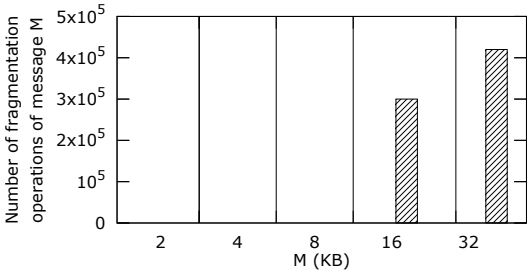
(a) Zoom on communication times for query Q_3.



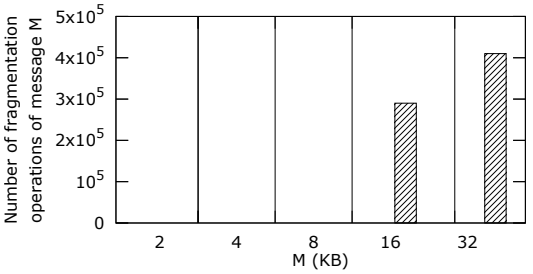
(b) Number of messages of size M communicated for F=1.1K tuples.



(c) Number of messages of size M communicated for F=110K tuples.



(d) Number of fragmentation operations on M to produce a network packet of MTU size ($8.95KB$) for F=1.1K tuples.



(e) Number of fragmentation operations on M to produce a network packet of MTU size ($8.95KB$) for F=110K tuples.

Figure 3.10: Zoom on Q_3 using ten configurations introduced in Example 3.2.

Chapter 4

MIND framework

We propose the MIND framework, which tunes the *middleware* parameters F and M, in order to minimize the communication time, while adapting to different queries and networks. The key ingredients of MIND are a *communication time estimation function* that we present in this chapter and an *iterative optimization algorithm*, proposed in Chapter 5.

More precisely, in this chapter we present the intuition behind the estimation function of the MIND framework (Section 4.1) and the model used to estimate the communication cost (Section 4.2). And the parameter calibration algorithm (Section 4.3). We analyse the accuracy of the proposed estimation function (Section 4.4). And demonstrate the sensitivity of calibrated parameters to network environment (in Section 4.5).

4.1 Intuition

Before entering into the detail of the proposed communication cost model, we stress the fact that in our query workload the communication time is dominant as shown in Figure 4.1. This is due to the fact that we focus our study on optimizing the communication cost and we do not consider the additional costs related to local processing and I/O operations.

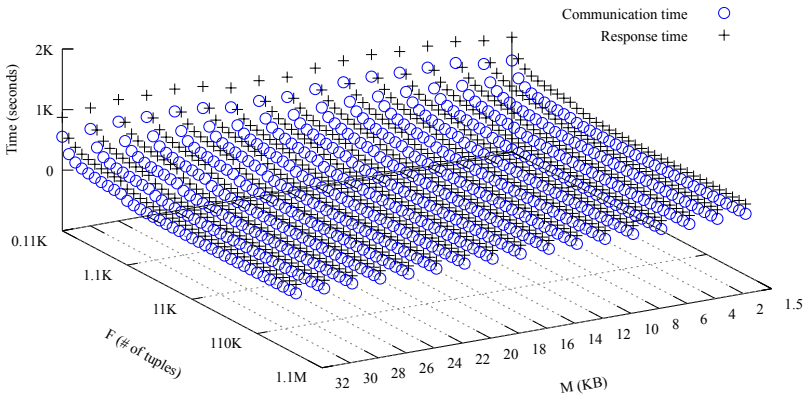


Figure 4.1: Elapsed versus communication times using Q_3 in high-bandwidth network.

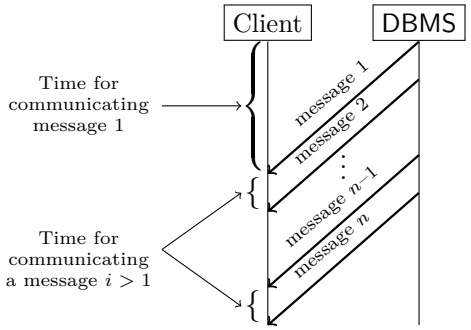


Figure 4.2: Pipeline communication of a batch of n messages between DBMS and Client nodes.

We recall two crucial observations, presented in Section 3.1.2, that allow us to develop an effective cost model for estimating the communication time:

- The batches are communicated synchronously over the network, whereas the messages in a batch (assuming that a batch has more than a message) are communicated in pipeline, hence it is possible to exploit the pipelining for minimizing the communication time, and
- Due to network latency and pipelining, it is more expensive to communicate the first message in a batch compared to a message that is not the first in its batch.

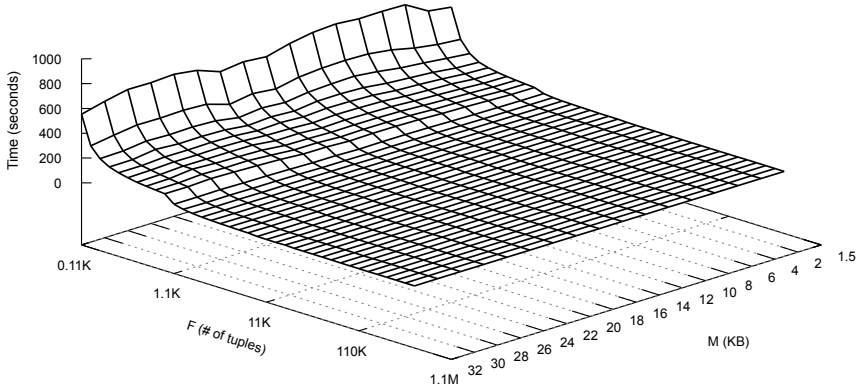
We next detail the first observation that we have already introduced in Section 3.2.1. On the one hand, when DBMS node sends a batch of F tuples to client node (cf. Figure 3.2), the communication is done *synchronously* i.e., client node needs to wait to receive the whole batch before it can start processing the batch and request the next batch. On the other hand, the messages of M bytes are sent in *pipeline* from DBMS node to client node i.e., DBMS node can send messages over the network without waiting for ack signals from client node. More precisely, messages are sequentially sent over the network once the buffer is filled by the *middleware*, which means that several messages of a same batch can be pipelined in the network at a specific moment. Figure 3.2 presents an illustration of a pipeline communication of a batch. In Figure 4.2, we zoom on pipeline communication of n messages between DBMS node and client node.

As for the second observation, we argue that it is more expensive to communicate the first message of a batch compared to the time needed to communicate any other message.

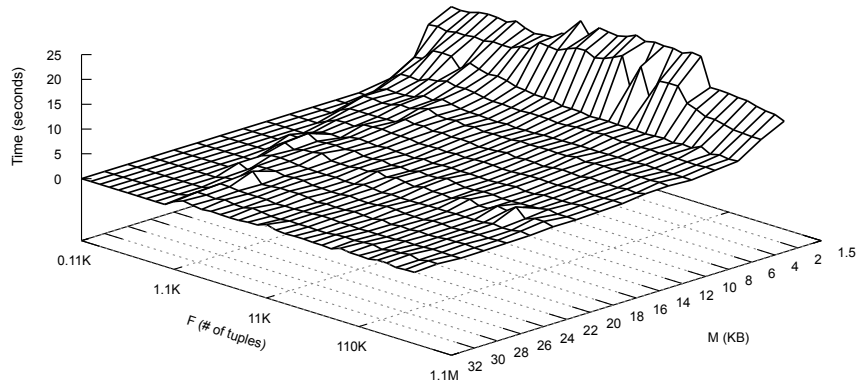
We illustrate via Example 4.1 that the described intuition effectively matches the practical behaviour. To the best of our knowledge, there are no related works in the literature that exploit such observations to optimize the communication time.

Example 4.1. Take query Q_3 (cf. Figure 3.3; recall that it has a result set of $\sim 4.5\,GB$). In the high-bandwidth network, the average time consumed by a first message is of 10^{-3} seconds, whereas the average time consumed by a non first messages (messages $i > 1$ in Figure 4.2) is of 3×10^{-5} seconds. The aforementioned numbers were obtained as an average for all 629 configurations of F and M (cf. Section 3.2.2).

Moreover, we report in Figure 4.3(a) and Figure 4.3(b) the total time of all first messages and the total time of all non first messages, respectively, for the same query Q_3 in the high-bandwidth network. Notice that:



(a) Time for communicating first messages.



(b) Time for communicating non first messages.

Figure 4.3: Time for communicating first messages vs not first messages in its batch using Q_3 in high-bandwidth network.

- The time of all first messages can go up to a thousand seconds (Figure 4.3(a)), whereas for all non first messages the time can go up to only tens of seconds 4.3(b), and
- The time of all first messages reach the highest values for small F (because this implies a large number of $round-trips$ i.e., a large number of batches, hence a large number of first messages, which are more expensive)

We observe that the aforementioned behaviour occurs in practice for all queries and both network configurations, although it is more visible for the queries Q_1 to Q_5 (which return larger results). □ ◊

As a consequence of this observation, the number of $round-trips$ (first messages) has an important impact on the communication time. As shown in the example 3.1.

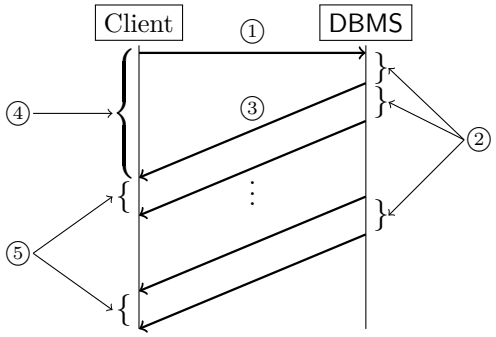


Figure 4.4: Consumed times for communicating a batch of F tuples in messages of M bytes from DBMS node to client node.

4.2 Communication cost model

In this section, before introducing MIND's function for estimating the communication time for a query result, we zoom on the intuition outlined in Section 4.1 and we enumerate in Figure 4.4 the communication steps and times needed for communicating a batch of F in messages of M bytes, which are:

- Step ①: time needed to initiate a request of batch of F tuples from client node to DBMS node.
- Step ②: time needed to initiate a message of M bytes by DBMS node.
- Step ③: time needed for communicating a message of M bytes over the network from DBMS node to client node.
- Step ④: time needed from receiving first message of M bytes in its batch by client node.
- Step ⑤: time needed from receiving not first message of M bytes in its batch by client node.

Relying on the Figure 4.4, MIND's function for estimating the communication time for a query result treats a communicated messages differently depending on whether or not it is the first in its batch since we want to exploit the pipelining. More precisely, given a query Q, we estimate the time C to communicate its result $Ans(Q)$ from DBMS node to client node with the formula:

$$C_{V,\alpha,\beta}(\mathsf{F}^B, \mathsf{M}) = \alpha \times \left\lceil \frac{V}{\mathsf{F}^B} \right\rceil + \beta \times \left\lceil \frac{V}{\mathsf{F}^B} \right\rceil \times \left(\left\lceil \frac{\mathsf{F}^B}{\mathsf{M}} \right\rceil - 1 \right)$$

where:

- F^B is the size in bytes of a batch i.e., the result of multiplying the number F of tuples in a batch and the size in bytes of a tuple in $Ans(Q)$;
- V is the size in bytes of the query result i.e., the result of multiplying the number of tuples in $Ans(Q)$ and the size in bytes of a tuple in $Ans(Q)$;

- The parameters α and β capture the characteristics of the network environment between the two nodes: α captures the time of communicating the first message of each batch (it is the ④ which equal to the sum of of consumed times in steps ①, ② and ③), whereas β captures the time of communicating a message that is not the first in its batch (it is the time consumed in step ⑤);
- The coefficient of α i.e., $\lceil V/\mathsf{F}^B \rceil$ gives the total number of first messages: $Ans(Q)$ has V bytes, which are communicated in batches of F^B bytes, consequently there are at all $\lceil V/\mathsf{F}^B \rceil$ batches, hence $\lceil V/\mathsf{F}^B \rceil$ first messages. As mentioned in Section 3.2.1, the number $\lceil V/\mathsf{F}^B \rceil$ is known as the number of *round-trips* in the literature [40];
- The coefficient of β i.e., $\lceil V/\mathsf{F}^B \rceil \times (\lceil \mathsf{F}^B/\mathsf{M} \rceil - 1)$ gives the total number of messages that are not the first in their batch: $\lceil \mathsf{F}^B/\mathsf{M} \rceil$ is the number of messages in a batch, hence $\lceil \mathsf{F}^B/\mathsf{M} \rceil - 1$ is the number of messages in a batch except the first one, and by multiplying it with the total number of batches $\lceil V/\mathsf{F}^B \rceil$, we obtain the total number of messages that are not the first in their batch.

Our formula for estimating the communication time is *query-dependent* (via the parameter V) and *network-dependent* (via the parameters α and β). In Section 4.3, we show how to calibrate α and β for a given network, whereas in Chapter 5 we develop an optimization algorithm that finds the values of F and M that allow to minimize C.

We end this section with two observations on the presentation of our formula $C_{V,\alpha,\beta}(\mathsf{F}^B, \mathsf{M})$:

- We omit the indices V, α, β when these parameters are clear from the context because the optimization is done for a query result and a network environment (hence these parameters do not change during the optimization), whereas we write F^B and M as function input variables because their values change throughout the iterations.
- We used F^B (the size in bytes of a batch) instead of F (the number of tuples in a batch) in the definition of C because this facilitates the presentation of the linear regression calibration algorithm (where both batch and message size are measured in bytes), which then also plays a role in the optimization algorithm.

4.3 Parameters calibration

As introduced in Section 4.2, α and β measure the time needed to communicate a message that is either the first or not the first in its batch, respectively. The parameter calibration introduced in MIND is in the spirit of the recent line of research on calibrating cost model parameters (for centralized DBMS) to take into account the specificities of the environment [18].

The calibration of α and β is the fundamental pre-processing step of the MIND framework, which allows to capture the network environment as part of the communication time estimation. Precisely, we capture the costs of first message in its batch via α and not first message in its batch via β. The general idea of our calibration algorithm is to iterate over a set of queries and different combinations of the two considered *middleware* parameters, and successively send over the network a query result according to a combination of parameters. Then, we leverage the observed communication times to estimate α and β.

In our calibration, we use the size of a batch in bytes (F^B) rather than the number of tuples in a batch (F) as a normalization of the batch size of all queries used for calibration. Indeed, different queries may have different result tuple size, hence knowing only the number of tuples in

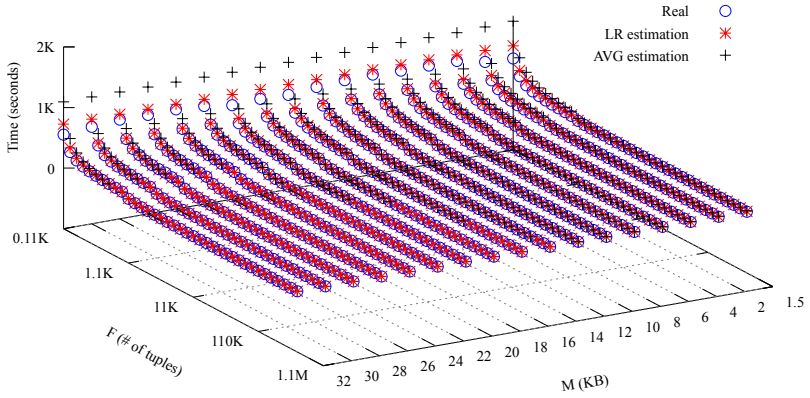


Figure 4.5: Comparison of LR versus AVG estimations using Q_3 in high-bandwidth network.

a batch does not provide enough information about the actual size in bytes that is communicated over the network in a batch.

A naive way to estimate α and β would be to represent them as constant real values that capture the averages of the observed communication times i.e., let α be the average time needed to communicate a message that is the first in its batch and let β be the average time needed to communicate a message that is not the first in its batch. However, such an approach does not capture the natural relationship between F^B, M and the communication time, as we illustrate in Example 4.2.

Example 4.2. Take the query Q_3 and the 629 combinations of F and M values (cf. Section 3.2.2). We plot in Figures 4.6 and 4.7 the times observed for the messages that are the first and not the first in their batch, respectively. The natural dependency between them is a "plane" (i.e., a linear function of F^B and M), whereas the plane given by choosing the calibration parameter α as an average value clearly does not capture this natural dependency. Also, Figure 4.5 shows a difference between average estimation and linear function estimation, where the last one provides an estimations very closer to real communication times. □ ◊

We observed in practice that the behaviour illustrated in Example 4.2 also holds for the other low-bandwidth network, and for other queries. This suggests that the natural way to capture the network environment via the parameters α and β is to define them as linear functions of F^B and M. Such a representation of α and β is very intuitive since the time of communicating a message over the network obviously depends on the actual network environment, the size of the message, and the size of the batch from which the message comes.

Next, we introduce the LR *(linear regression) calibration algorithm* (Algorithm 1), which captures the linear dependency between F^B and M by solving a linear regression based on the observed times. We use a ternary relation $result_1$ that stores three columns on each line: F^B, M, and the observed communication time for a message that is first for α and not first for β. Then, we use a standard ordinary least squares (*ols*) regression to compute α as a function

$$\alpha(\mathsf{F}^B, \mathsf{M}) = a_1 \times \mathsf{F}^B + b_1 \times \mathsf{M} + c_1$$

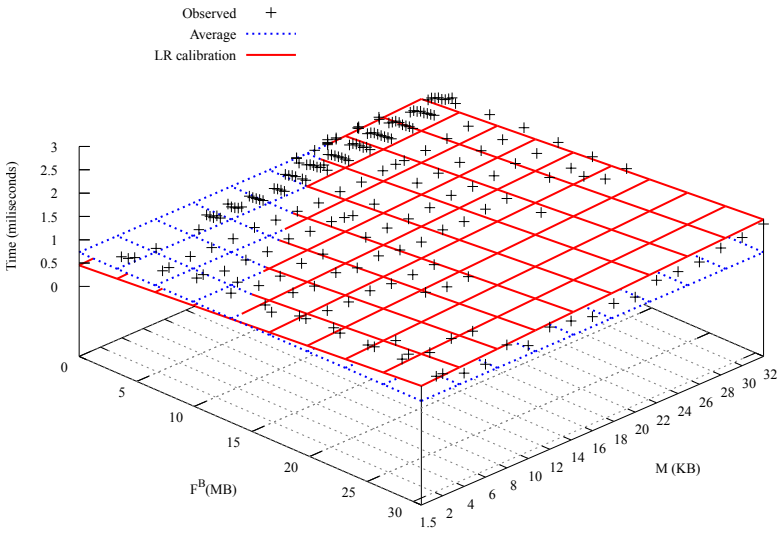


Figure 4.6: Calibration of parameter α in high-bandwidth network.

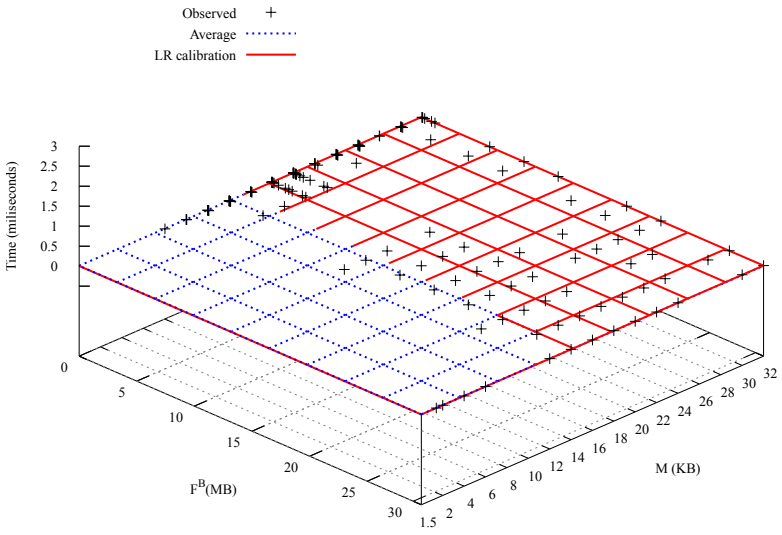


Figure 4.7: Calibration of parameter β in high-bandwidth network.

that captures the time to communicate a message that is the first in its batch. We similarly compute β as a function

$$\beta(\mathsf{F}^B, \mathsf{M}) = a \times \mathsf{F}^B + b \times \mathsf{M} + c$$

that captures the time to communicate a message that is not the first in its batch, by using a similar data structure *result* to store the observed times.

Algorithm 1 LR (linear regression) algorithm for calibrating α and β. We use the index 1 when we refer to a message that is the first in its batch and no index for all other messages.

Input: set $\mathcal{Q}$ of queries, set $\mathcal{F}$ of possible F values, set $\mathcal{M}$ of possible M values
Output: α, β (each of them as a function of F^B and M)
let $result_1$=$\emptyset$, $result$=$\emptyset$
for each $(Q, \mathsf{F}, \mathsf{M}) \in \mathcal{Q} \times \mathcal{F} \times \mathcal{M}$ **do**
  $update_1$ ($result_1$)
  $update$ ($result$)
return ($ols(result_1)$, $ols(result)$)

Example 4.2 (continued). The linear function $\alpha(\mathsf{F}^B, \mathsf{M})$ and $\beta(\mathsf{F}^B, \mathsf{M})$ result of applying a linear regression on the observed points from Figures 4.6 and 4.7 are, respectively:

- $\alpha(\mathsf{F}^B, \mathsf{M}) = 2.02e^{-11} \times \mathsf{F}^B + 1.17e^{-8} \times \mathsf{M} + 4.5e^{-4}$, which captures precisely the plane entitled "LR calibration" in Figure 4.6;
- $\beta(\mathsf{F}^B, \mathsf{M}) = 1.24e^{-14} \times \mathsf{F}^B + 2.73e^{-10} \times \mathsf{M} + 4.35e^{-6}$, which captures precisely the plane entitled "LR calibration" in Figure 4.7.

□

At the end of this section, it is important to note that a new calibration phase is needed when a network environment changes (that may vary in terms of bandwidth).

4.4 Accuracy of communication cost model

To emphasize the accuracy of the communication time estimation on top of the LR algorithm, we present in Figure 4.8 the real and estimated communication times, for all queries (cf. Figure 3.3) in high-bandwidth network (we report averages over the 629 combinations of F and M values cf. Section 3.2.2), using query Q_3 as calibration query. We observe that MIND achieves a very accurate estimation for all six queries, although the majority of them differ from the calibration query Q_3 in terms of both tuple size and/or number of tuples in the query result.

Moreover, we empirically observed that the estimation has the desirable monotonicity property i.e., when the estimated time for a combination of F and M values is smaller than for another combination, then the real communication time for the first combination of F and M values is also smaller than the second one. Such a property is useful because it implies that the combination of values that minimizes the estimation function is also the one minimizing the real communication time.

For instance, we depict in Figure 4.12 the real and estimated communication times for query Q_4 in the high-bandwidth network. Notice that query Q_4 that we present in Figure 4.12 differs from query Q_3 that we used for calibration in terms of both tuple size and number of tuples in the query result, which suggests the pertinence of our estimation method for middleware optimization.

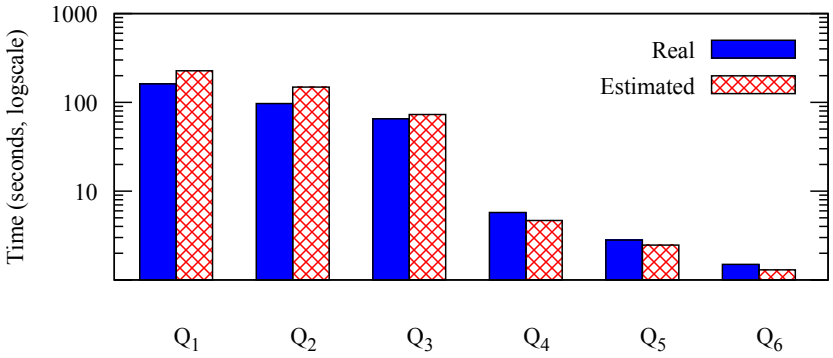


Figure 4.8: Average real and estimated communication times for all 6 queries (cf. Figure 3.3) in high-bandwidth network.

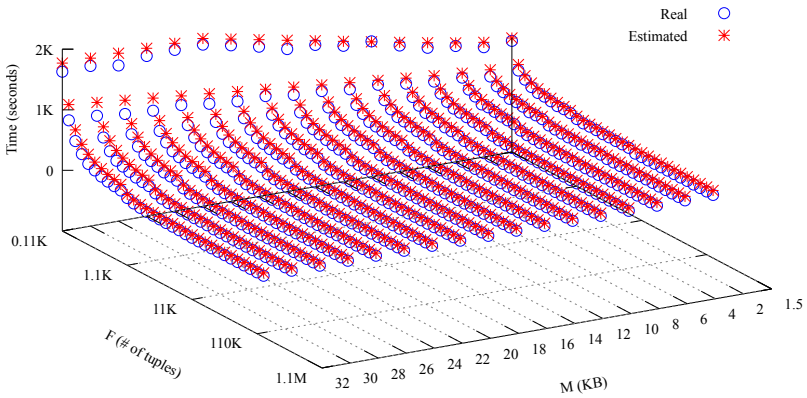


Figure 4.9: Zoom on Q_1 in high-bandwidth network.

We empirically observed similar monotonicity behaviours for all queries and network bandwidths.

4.5 Sensitivity of calibrated parameters to network environment

In this section, we show the sensitivity of calibrated parameters α and β on the network environment characteristics. We experiment only the sensitivity on network bandwidth. We use the network configurations (high and low bandwidth networks) introduced in Section 3.2.2.

We recall that in Section 3.2, we argue that the time for communicating a distributed query result is network-dependency presented.

To show the sensitivity of calibrated parameters α and β, we zoom on the values of these parameters in both network configurations (high-bandwidth (10 $Gbps$) and low-bandwidth (50 $Mbps$)). We report the values of α and β calibrated on the same query Q_3 in Figure 4.15 for high-bandwidth network and in Figure 4.16 for low-bandwidth network, respectively. We notice that:

- The calibration captures well the variation of network environment characteristics, in the

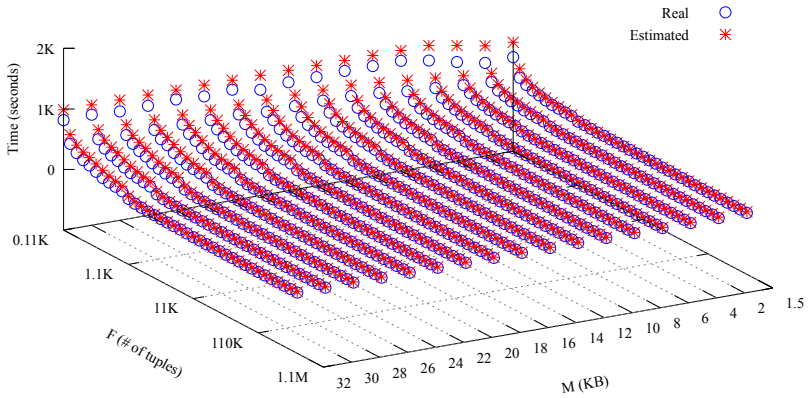


Figure 4.10: Zoom on Q_2 in high-bandwidth network.

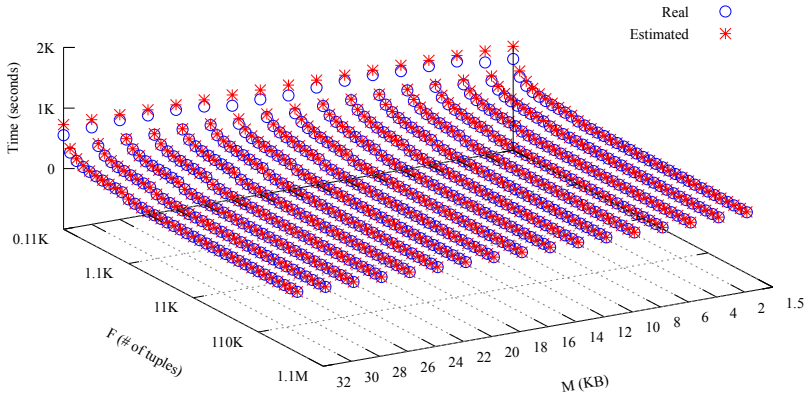


Figure 4.11: Zoom on Q_3 in high-bandwidth network.

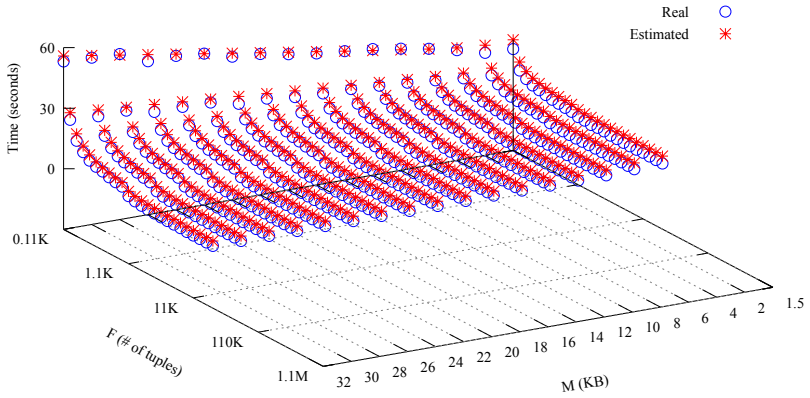


Figure 4.12: Zoom on Q_4 in high-bandwidth network.

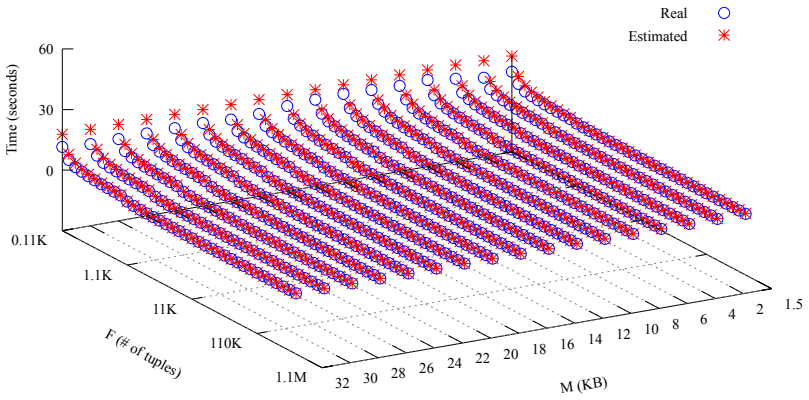


Figure 4.13: Zoom on Q_5 in high-bandwidth network.

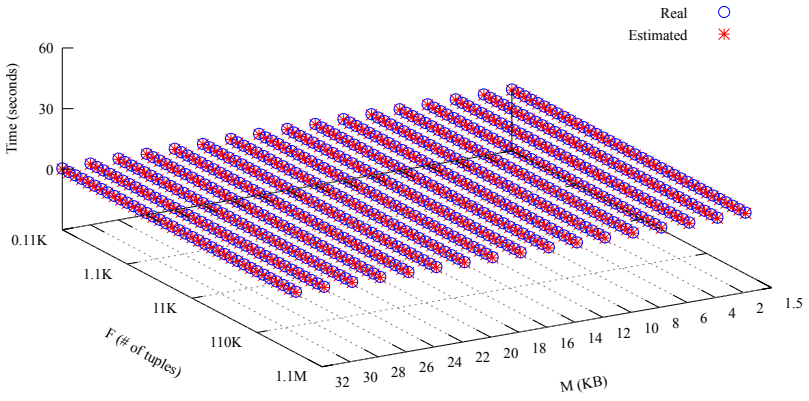
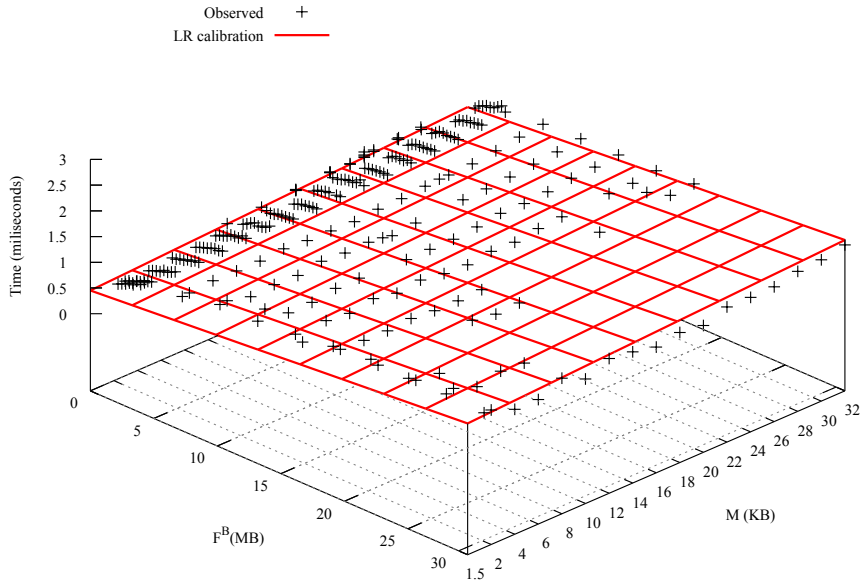


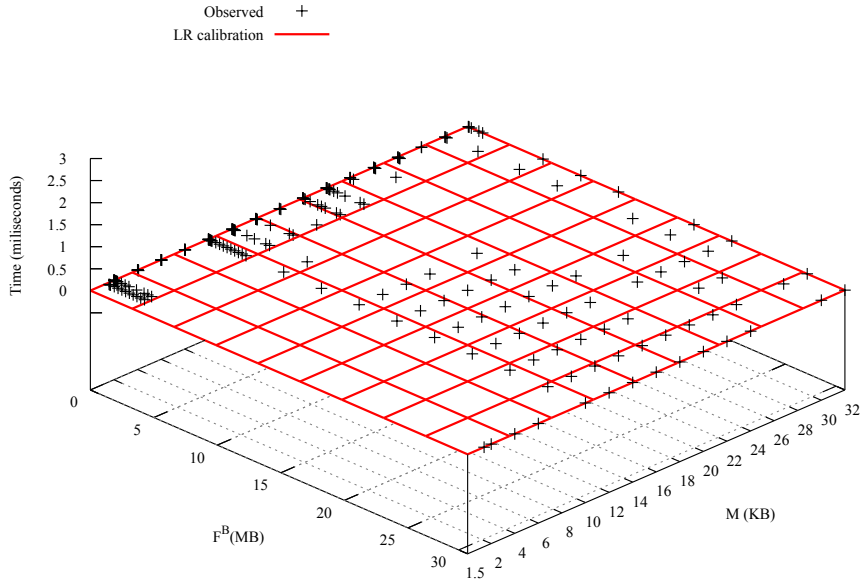
Figure 4.14: Zoom on Q_6 in high-bandwidth network.

sense that in high bandwidth the β values are negligible comparatively to the α values, whereas in low-bandwidth, the β values are not negligible and have an important weight as for the α values. This is because β captures a latency due to the low-bandwidth.

- The α values in high and low bandwidth networks shows a closer estimation time. This is due to the fact that α captures the time for communicating the first message in its batch. However, the first messages can avoid the network latency.
- The β calibration captures the variation of network environment characteristics, in the sense that β captures the time for communicating messages that are not first in their batches. Hence, large F^B communicates an important number of messages in pipeline, which can not be consumed easily and quickly in low-bandwidth network.

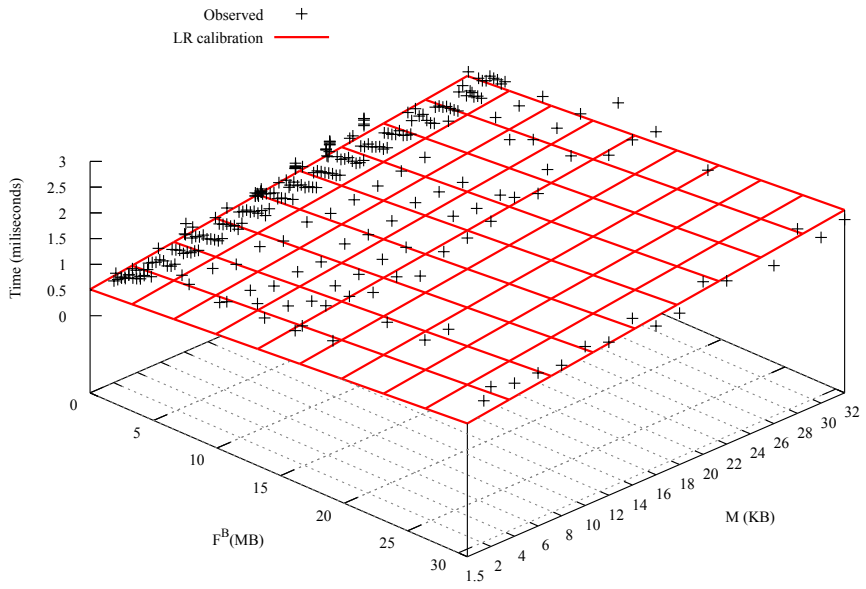


(a) Time consumed for communicating first message in each batch in high bandwidth for Q_3.

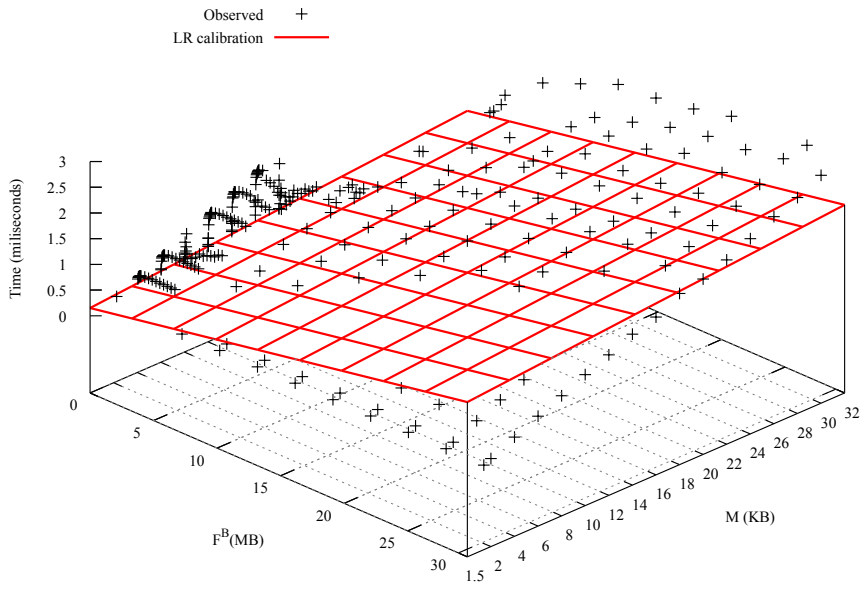


(b) Time consumed for communicating next message that is not the first in its batch in high bandwidth for Q_3.

Figure 4.15: Sensitivity of calibrated parameters α and β in high-bandwidth networks using Q_3 in calibration phase.



(a) Time consumed for communicating first message in each batch in low bandwidth for Q_3.



(b) Time consumed for communicating next message that is not the first in its batch in low bandwidth for Q_3.

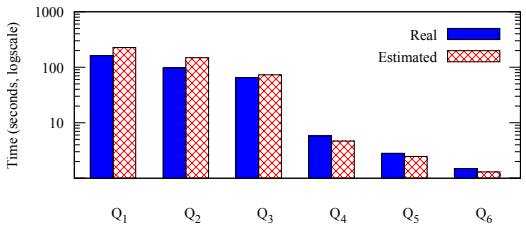
Figure 4.16: Sensitivity of calibrated parameters α and β in low-bandwidth networks using Q_3 in calibration phase.

Concretely, the linear functions $\alpha(\mathsf{F}^B, \mathsf{M})$ and $\beta(\mathsf{F}^B, \mathsf{M})$ result of applying a linear regression on the observed points from Figure 4.15 and Figure 4.16 in high- and low-bandwidth networks, respectively, are:

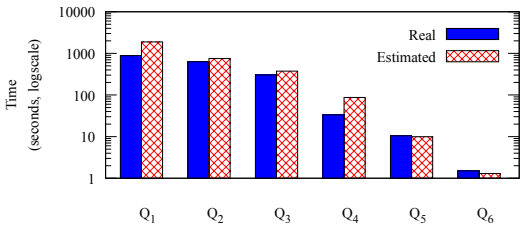
- High-bandwidth network:
  - $\alpha(\mathsf{F}^B, \mathsf{M}) = 2.02e^{-11} \times \mathsf{F}^B + 1.17e^{-8} \times \mathsf{M} + 4.5e^{-4}$, which captures precisely the plane entitled "LR calibration" in Figure 4.15(a). The variance of the values of a_1, b_1 and c_1 are $+/-5.32e^{-13}$ (2.63%), $+/-4.6e^{-10}$ (3.93%) and $+/-8.4e^{-6}$ (1.86%), respectively, and
  - $\beta(\mathsf{F}^B, \mathsf{M}) = 1.24e^{-14} \times \mathsf{F}^B + 2.73e^{-10} \times \mathsf{M} + 4.35e^{-6}$, which captures precisely the plane entitled "LR calibration" in Figure 4.15(b). The variance of the values of a, b and c are $+/-9.12e^{-14}$ (73.58%), $+/-8.06e^{-11}$ (2.94%) and $+/-1.53e^{-6}$ (3.53%), respectively.
- Low-bandwidth network:
  - $\alpha(\mathsf{F}^B, \mathsf{M}) = 1.94e^{-11} \times \mathsf{F}^B + 2.94e^{-8} \times \mathsf{M} + 5.12e^{-4}$, which captures precisely the plane entitled "LR calibration" in Figure 4.16(c). The variance of the values of a_1, b_1 and c_1 are $+/-7.787e^{-13}$ (4.00%), $+/-6.678e^{-10}$ (2.26%) and $+/-1.214e^{-5}$ (2.37%), respectively, and
  - $\beta(\mathsf{F}^B, \mathsf{M}) = 4.51e^{-11} \times \mathsf{F}^B + 2.25e^{-8} \times \mathsf{M} + 1.29e^{-4}$, which captures precisely the plane entitled "LR calibration" in Figure 4.16(d). The variance of the values of a, b and c are $+/-2.41e^{-12}$ (5.34%), $+/-2.11e^{-9}$ (9.39%) and $+/-4.05e^{-5}$ (31.28%), respectively.

At the end of this section, it is important to note that the calibration of parameters α and β preserves the network-dependency feature on the estimation of the communication time, at the price of preprocessing phase to calibrate these parameters to a current network environment. The network-dependency is visible via the parameter β in high- and low-bandwidth networks, which presents a large difference between both network configurations.

We report in Figure 4.17, the estimated and real communications times for all queries (cf. Figure 3.3) the averages communication times over 629 F and M values in both networks (high and low-bandwidth). From this Figure, we notice that our communication cost model presents a good adaptivity to network bandwidth variation (from high:10 *Gbps* to low:50 *Mbps*), in the sense that the estimated communication times are closer to the real communication times, as presented in this figure.



(a) All 6 queries in high-bandwidth network.



(b) All 6 queries in low-bandwidth network.

Figure 4.17: Real and estimated communication times, for all queries (cf. Figure 3.3) in high and low-bandwidth networks.

4.6 Discussion

At the end of this chapter, we stress that:

- Our estimation function (communication cost model) simulates accurately the real communication time, in the sense that it computes a good communication time estimation (cf. Section 4.4).
- This cost model takes into account the pipeline costs, since it considers that the batches of F tuples are communicated synchronously over the network, whereas the messages of M bytes in a batch are communicated in pipeline. Recall that the classical communication cost model (*per-message* and *per-byte*) can not provide a good estimation, because it does not take into a account the pipeline effect(cf. Section 3.2).
- This cost model preserves the *query-dependency* and *network-dependency*.
  - *Query-dependency*, in the sense that the coefficients of α and β, which are the number of first messages and other messages *i>1* respectively, change according to the query result and tuple sizes. Consequently, the estimated communication times (cf. Section 4.4) change according to the given query.
  - *Network-dependency*, since the calibrated parameters α and β are very sensitive to the environment and capture the characteristics of the link between client and DBMS nodes, such as presented in Section 4.5.
- The number of combinations that should be used in calibration phase is an important element, in the sense that a large number of combinations of queries Q, F and M provide a good calibration of parameters α and β but at the price of a large time.

Next we present an iterative optimization algorithm of the MIND framework that allows to find a good *middleware* configuration (F and M) that minimizes the communication cost function and provides a good trade-off of resource consumption (particularly memory space should be reserved to F^B).

Chapter 5

An iterative middleware tuning approach

In this chapter, we present our optimization algorithm that takes as input query result needed to be communicated from DBMS node to client node and effectively compute the values of the middleware parameters F and M that minimize the communication time using the estimation function described in Section 4.2. In Section 5.1, we introduce the optimization problem, in Section 5.2, we develop our optimization algorithm that efficiently finds good values for the middleware parameters, in Section 5.3, we evaluate MIND optimization algorithm, whereas in Section 5.4, we present how MIND optimization algorithm stops iterations via threshold (Δ).

5.1 Optimization problem

We consider the optimization problem as computing F and M that aim to minimize the communication cost function $C(\mathsf{F}^B, \mathsf{M})$ (cf. Section 4.2). We formulate our optimization problem as follows:

Input: Size of the query result (V), the tuple size in query result (T) and the network environment characteristics captured via the parameters α and β obtained from the calibration step.
Output: Parameters F and M that minimize the time for communicating the query result over the network and consume less memory resources.

The values of F and M are integers that can span over the intervals of all possible values $[\mathsf{F}_{\min}, \mathsf{F}_{\max}]$ and $[\mathsf{M}_{\min}, \mathsf{M}_{\max}]$, respectively. Notice that a brute-force algorithm would have to loop over all combinations of possible values from the two sets and see which combination minimizes the estimation of the communication time. It is easy to see that such an algorithm would need to explore a very large number of combinations, as we illustrate via Example 5.1.

Example 5.1. Assuming a system whose memory can handle batches of up to 1.1M tuples (for a tuple size of $27B$), we obtain that $[\mathsf{F}_{\min}, \mathsf{F}_{\max}]$ has 1.1M elements. Moreover, assuming that the same system can handle messages of size between $512B$ and $32KB$, we obtain that $[\mathsf{M}_{\min}, \mathsf{M}_{\max}]$ has more than 32K elements. Hence, there are more than 35 billions of distinct combinations of F and M values. □ ◊

Since it would be highly inefficient to naively explore all possible combinations, we developed an optimization algorithm (detailed in Section 5.2) that efficiently explores the large search space to quickly find the middleware parameters F and M that minimize the communication time estimation.

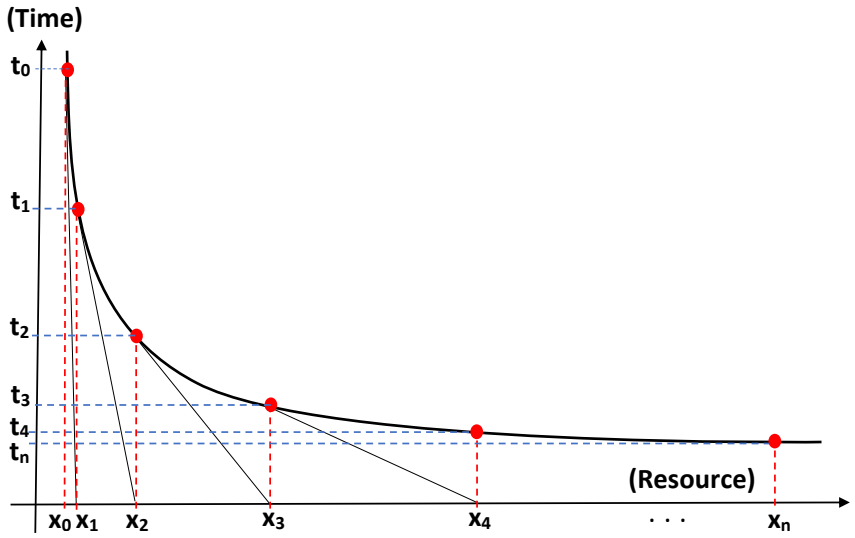


Figure 5.1: Illustration of consecutive iterations in Newton resolution.

5.2 Optimization approach

We present an optimization algorithm based on the *Newton optimization*, which is a popular numerical optimization paradigm [32]. We have chosen to rely on such a paradigm because the optimization is done iteratively, which allows to control the number of iterations. Figure 5.1 illustrates how iterations are done in Newton resolution. In particular, the algorithm that we present in this section, the control is done via a threshold parameter measuring the gain in terms of communication time between two consecutive iterations. This allows to stop the algorithm when the gain is insignificant and comes at the price of large consumed resources, as illustrated in Figure 5.1. Our approach translates in practice to a good trade-off between low communication time and low resource consumption.

Recall that we introduced the communication time estimation function in Section 4.2 and the calibration algorithm for network-dependent parameters α and β in Section 4.3. The function that we optimize is (after replacing the calibration parameters in the estimation function):

$$C(\mathsf{F}^B, \mathsf{M}) = (a_1 \times \mathsf{F}^B + b_1 \times \mathsf{M} + c_1) \times \frac{V}{\mathsf{F}^B} + (a \times \mathsf{F}^B + b \times \mathsf{M} + c) \times \frac{V}{\mathsf{F}^B} \times \left(\frac{\mathsf{F}^B}{\mathsf{M}} - 1\right).$$

We present the pseudo-code of the optimization algorithm in Algorithm 2, where we write x and y as simplified notations for F^B and M, respectively. The input consists of:

- Two parameters related to the query: the result size in bytes V (it is not visible later on in the pseudo-code, but it is a necessary parameter of the estimation function C and its derivatives) and the tuple size in bytes T (which is used at the end of the algorithm to convert the batch size in bytes as computed by the algorithm in a batch size in number of tuples, as usually used in a DBMS);

- Six parameters related to the network environment (a_1, b_1, c_1, a, b and c that capture the coefficients of the linear functions that are result of the calibration, cf. Section 4.3);
- The maximum size $\mathsf{F}^B_{\max}$ of a batch in bytes, that we set as the maximum heap memory size allowed by the *middleware* for a batch (hence we can compute $\mathsf{F}_{\max}$ as the largest number of tuples that can fit in a batch that consumes the entire heap memory; we always assume $\mathsf{F}_{\min} = 1$ tuple);
- The minimum and maximum values for the message size allowed by system: $\mathsf{M}_{\min}$ and $\mathsf{M}_{\max}$, respectively;
- Threshold Δ used to measure whether the gain of the time estimation between two consecutive iterations is small enough (insignificant) to stop the algorithm.

Our algorithm is iterative in the sense that it starts with initial (small) values of x and y, and iterates to improve the estimation by updating the initial values. In particular, we configure the initial x_0 to contain a number of tuples that needs at least two messages of initial message size y_0=$\mathsf{M}_{\min}$ (to exploit the pipeline communication of messages). It is important to note that *Newton optimization* does not impose a particular rule to define the initial configuration.

Algorithm 2 MIND optimization algorithm (x and y are simplified notations for F^B and M, respectively).

Input: V, T, a_1 b_1, c_1, a, b, c, $\mathsf{F}^B_{\max}$, $\mathsf{M}_{\min}$, $\mathsf{M}_{\max}$, Δ
Output: F and M minimizing C (cf. Section 4.2)
let $y_0 = \mathsf{M}_{\min}$
let $x_0 = 2 \times y_0$
let $k = 0$
while true do
$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} - \left(\mathbf{H}(x_k, y_k)\right)^{-1} \times \begin{pmatrix} g_1(x_k, y_k) \\ g_2(x_k, y_k) \end{pmatrix}$$
$x_{k+1} = \min(x_{k+1}, \mathsf{F}^B_{\max})$
$y_{k+1} = \min(y_{k+1}, \mathsf{M}_{\max})$
if $C(x_k, y_k) - C(x_{k+1}, y_{k+1}) \leq \Delta$ **or**
$(x_{k+1} = \mathsf{F}^B_{\max}$ **and** $y_{k+1} = \mathsf{M}_{\max})$ **then**
return $(\lfloor \frac{x_{k+1}}{T} \rfloor, y_{k+1})$
else
$k := k + 1$

For every loop, we update the values of the *middleware* parameters F and M as follows. We rely on g_1 and g_2 that are the *gradient functions* of the *Newton optimization* i.e., the first-order partial derivatives of the communication time estimation function C w.r.t. x and y, respectively:

$$g_1(x, y) = \frac{\partial C(x, y)}{\partial x},$$
$$g_2(x, y) = \frac{\partial C(x, y)}{\partial y}.$$

After applying standard differentiation rules, we obtain:

$$g_1(x,y) = \frac{V \times ((c-c_1)+(b-b_1)\times y)}{x^2} + \frac{a \times V}{y},$$
$$g_2(x,y) = \frac{V \times (b_1-b)}{x} - \frac{V \times (a \times x + c)}{y^2}.$$

We also rely on the *Hessian matrix* $\mathbf{H}$ i.e., a square matrix of second-order partial derivatives of C. The four values of $\mathbf{H}$ are obtained by differentiating each of g_1 and g_2 w.r.t. x and y, respectively.

$$\mathbf{H}(x,y) = \begin{pmatrix} \frac{\partial g_1(x,y)}{\partial x} & \frac{\partial g_1(x,y)}{\partial y} \\ \frac{\partial g_2(x,y)}{\partial x} & \frac{\partial g_2(x,y)}{\partial y} \end{pmatrix}.$$

After applying standard differentiation rules, we obtain:

$$\mathbf{H}(x,y) = \begin{pmatrix} \frac{-2\times V\times((c-c_1)+(b-b_1)\times y)}{x^3} & \frac{V\times(b-b_1)}{x^2} - \frac{a\times V}{y^2} \\ \frac{V\times(b-b_1)}{x^2} - \frac{a\times V}{y^2} & \frac{2\times V\times(a\times x+c)}{y^3} \end{pmatrix}.$$

We compute the new values x_{k+1} and y_{k+1} based on the current values x_k and y_k, the Hessian matrix, and the gradient functions applied on x_k and y_k. If any of the new values surpasses the corresponding maximum value $\mathsf{F}^B_{\max}$ and $\mathsf{M}_{\max}$, respectively, we normalize it as the maximum value. The algorithm stops when either

- The gain in terms of communication time estimation between two consecutive iterations is below the threshold Δ (this is the halt condition that occurred in all our experiments), or
- Both *middleware* parameters attain the maximum values ($\mathsf{F}^B_{\max}$ and $\mathsf{M}_{\max}$). The algorithm returns the last values of x (converted to number of tuples) and y.

The algorithm 2 works in practice because the gradient functions g_1 and g_2, and the Hessian matrix $\mathbf{H}$ exist on the entire domain of the two input variables of the estimation function C (in other words the function C is twice differentiable). This happens because in our framework, x and y (that appear as denominators in g_1, g_2, and all four elements of $\mathbf{H}$) can never be zero (since the size in bytes of a batch $x > 0$ and the size in bytes of a message $y > 0$).

This algorithm allows us to quickly find (always in small fraction of a second) values of the *middleware* parameters F and M for which the improvement in terms of communication time estimation between two consecutive iterations is insignificant. In practice, this translates to a good trade-off between low resource consumption and low communication time.

We end this section by noting that the proposed optimization algorithm can be easily incorporated inside the DBMS. Indeed, when a DBMS node is required to compute the result of a specific query (e.g., as part of a distributed query plan), MIND can help the DBMS by computing the best parameters F and M (based on the query and the network configuration; these parameters are subsequently used when sending the query result over the network).

5.3 Evaluation of MIND framework

In this section, we present an evaluation of the MIND framework as an end-to-end solution. In particular, we point out the improvement that we obtain over the current strategies for *middleware* tuning (in terms of communication time and/or resource consumption), the query- and network-adaptivity of MIND, and how the time estimation and the two *middleware* parameters F and M change during the iterations. Moreover, recall that we have already shown an experiment for the accuracy of the MIND estimation function in Section 4.4.

Throughout this section, we rely on the experimental setup introduced in Section 3.2.2 when we motivated our study on the impact of the *middleware* configuration. In particular, we use the same *data* (the astronomical dataset of $\sim 34GB$), *queries* (six queries of diverse selectivity cf. Figure 3.3), and *network configurations* (high- and low-bandwidth networks).

Middleware tuning strategies

We benchmark MIND against the same five strategies introduced in Section 3.2.4. In the remainder of this section, by MIND we denote the combination of F and M returned by the MIND optimization algorithm (cf. Figure 2), with $[\mathsf{M}_{\min}, \mathsf{M}_{\max}]= [512B, 32KB]$, the maximum heap memory size allowed by the *middleware* for a batch $\mathsf{F}^{B}_{\max}$=$30MB$, the threshold Δ set to a second, and network parameters calibrated with the LR algorithm (with input the query Q_3 and the 629 combinations of F and M cf. Section 3.2.2).

We also recall the five strategies from Section 3.2.4, which correspond to recommendations found in technical documentations for tuning the *middleware* parameters e.g., [6, 7, 40]:

(i) Default (default values set by the *middleware* of our DBMS),

(ii) Max M (set M to maximum value and leave F as default),

(iii) Max F (set F to maximum value i.e., all tuples in a batch consume the entire heap memory allowed by the *middleware* for a batch, and leave M as default),

(iv) Max F/M (set both parameters to maximum values),

(v) F in Max M (set M to maximum value and set F such that all tuples in a batch fit in a single message).

Comparison of middleware tuning strategies

We report in Figures 5.3, 5.4 and 5.2 the times and the resources needed to communicate over the network the result of each query, using each strategy and each network i.e., high- and low-bandwidth.

The only query for which all strategies yield the same communication time is Q_6, which has the smallest query result. Nonetheless, we observe that MIND consumes less resources to achieve this communication time i.e., messages of only $0.5KB$. Next, we make some important points that hold for the first five queries, which have large results i.e., between $\sim 0.1GB$ and $\sim 32GB$.

MIND gives particularly good results in the low-bandwidth network, where it is always the best among all strategies. In the high-bandwidth network, the two strategies using maximum F values (i.e., Max F and Max F/M) obtain slightly smaller communication times, but at the price of consuming much larger resources (since they consume the entire heap memory allowed by the

	Default		Max M		Max F		Max F/M		F in Max M		MIND			
	Common for both networks										High-bandwidth		Low-bandwidth	
	F	M	F	M	F	M	F	M	F	M	F	M	F	M
Q_1	15	8	15	32	110K	8	110K	32	150	32	110K	32	5K	32
Q_2	15	8	15	32	550K	8	550K	32	550	32	88K	21	12K	32
Q_3	15	8	15	32	1.1M	8	1.1M	32	1.1K	32	44K	22	24K	32
Q_4	15	8	15	32	66K	8	66K	32	70	32	1K	6	2K	32
Q_5	15	8	15	32	1.1M	8	1.1M	32	1.1K	32	1K	2	4K	31
Q_6	15	8	15	32	55	8	55	32	55	32	36	0.5	36	0.5

Figure 5.2: Resources needed for each query by the strategies from Figures 5.3 and Figures 5.4 (F in tuples, M in *KB*).

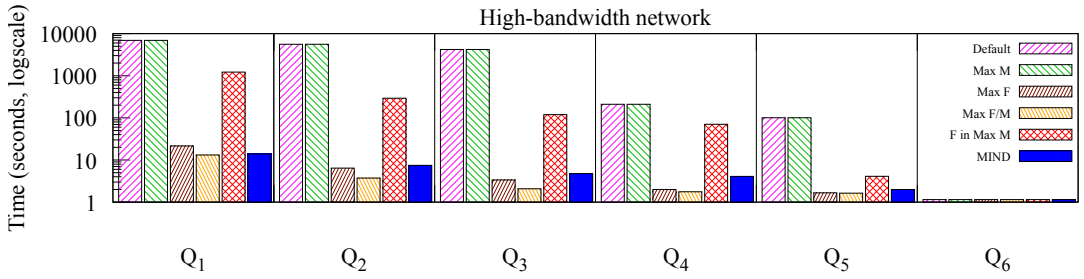


Figure 5.3: Communication times of six *middleware* tuning strategies, in high bandwidth network.

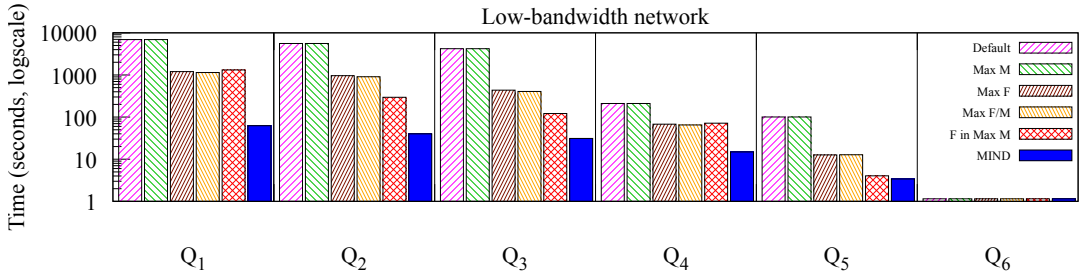


Figure 5.4: Communication times of six *middleware* tuning strategies, in low bandwidth network.

middleware for a batch). For example, for Q_1, Max F and Max F/M use batches of 110K tuples, whereas MIND needs batches of only 37K tuples to achieve a comparable communication time, etc.

Moreover, the good communication times of the two strategies using maximum F values are not confirmed in the low-bandwidth network since its latency precludes the pipelining of large batches. We recall that MIND is always the best strategy in the low-bandwidth network, which suggests another advantage of MIND that is the network-adaptivity. On the other hand, the existing strategies use the same *middleware* configuration regardless the network environment.

We also point out that the strategies using small values of F (i.e., Default and Max M) always give the worst communication times due to the overhead implied by a large number of $round-trips$ (first messages), confirming the discussion from Section 3.2.4

The limitations of the current strategies for *middleware* tuning that we have discussed in this section are particularly interesting since all five competing strategies are recommended by

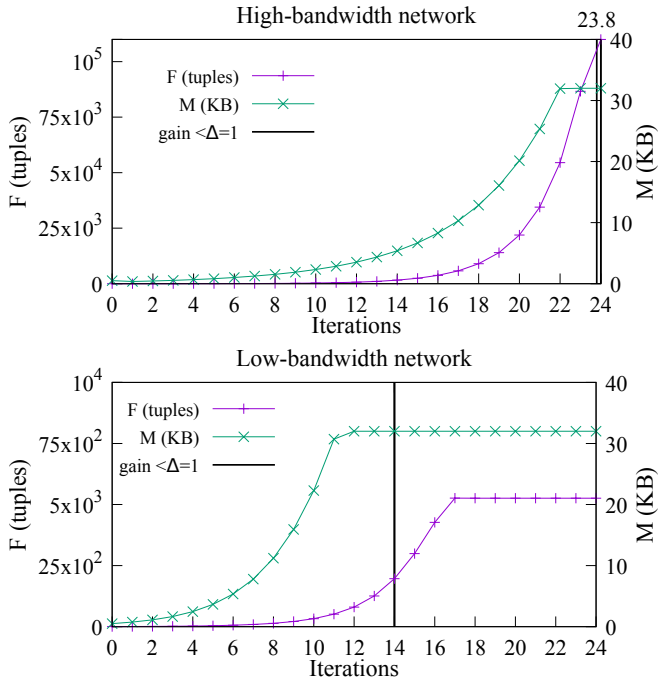


Figure 5.5: Evolution of the values of F and M using Q_1 throughout the iterations of the MIND algorithm.

technical documentations e.g., [6, 7, 40]. On the other hand, the *middleware* tuning achieved by MIND allows to find a good trade-off between a low communication time and low resource consumption by *middleware* parameters F and M.

Query-adaptivity of MIND

We next point out that the values of F and M returned by MIND change from query to query (cf. Figure 5.2). For both networks, all queries have pairwise distinct values for F and M. We recall that MIND always yields the best communication time in the low-bandwidth network, whereas in the high-bandwidth network it achieves comparable communication times to the strategies Max F and Max F/M that consume larger resources.

None of the competing strategies is query-adaptive. Indeed, every such strategy always assumes that the message size M is fixed for all queries, whereas the batch size F is fixed to the default value (strategies Default and Max M), fixed such that a batch occupies the entire allowed heap memory (strategies Max F and Max F/M), or fixed such that a batch occupies a maximum message size (strategy F in Max M). In particular, the approach of fixing the batch size such that it consumes the entire heap memory is not sustainable in practice (since all the memory is reserved for a single query).

In Tables 5.1 and 5.2, we show how MIND optimization algorithm changes the values of *middleware* parameters F and M over iterations. We focus on queries Q_3 and Q_5 that both queries having the same tuple size but differ in query result size.

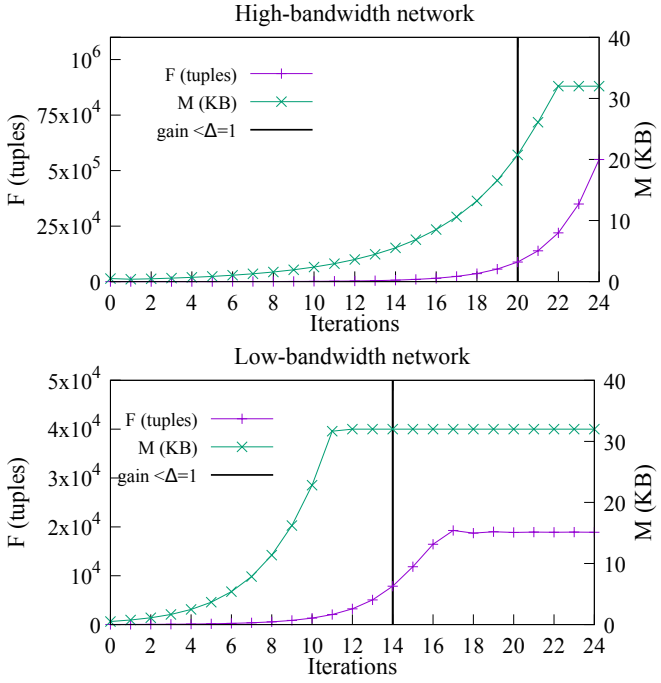


Figure 5.6: Evolution of the values of F and M using Q_2 throughout the iterations of the MIND algorithm.

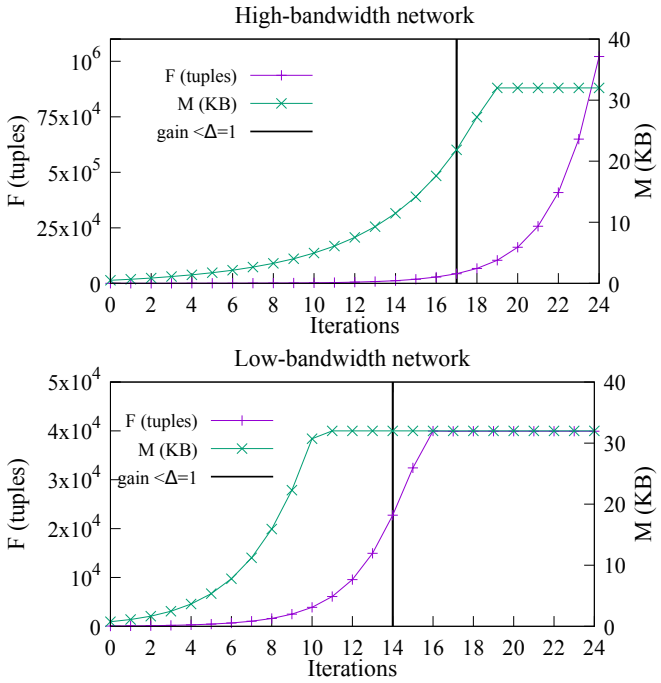


Figure 5.7: Evolution of the values of F and M using Q_3 throughout the iterations of the MIND algorithm.

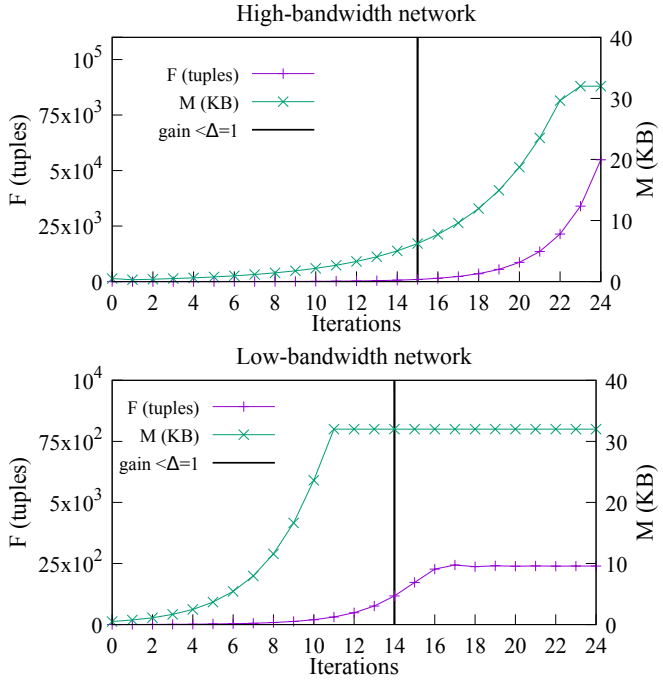


Figure 5.8: Evolution of the values of F and M using Q_4 throughout the iterations of the MIND algorithm.

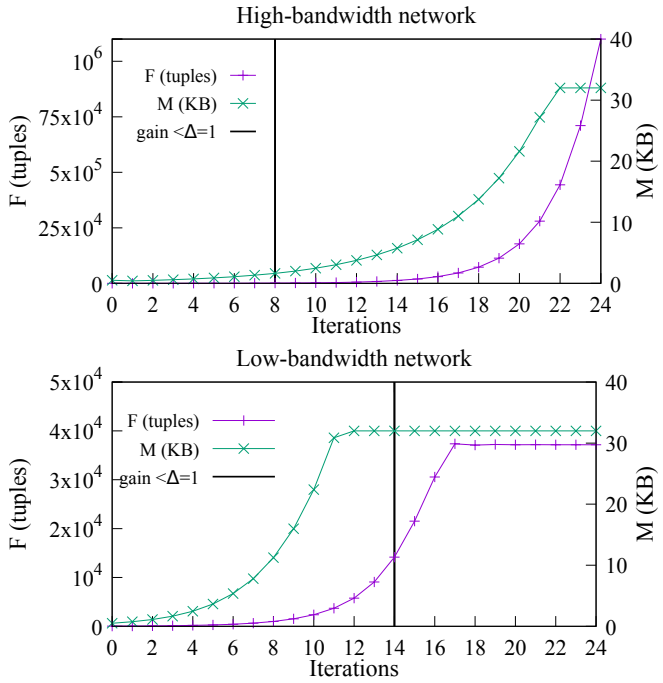


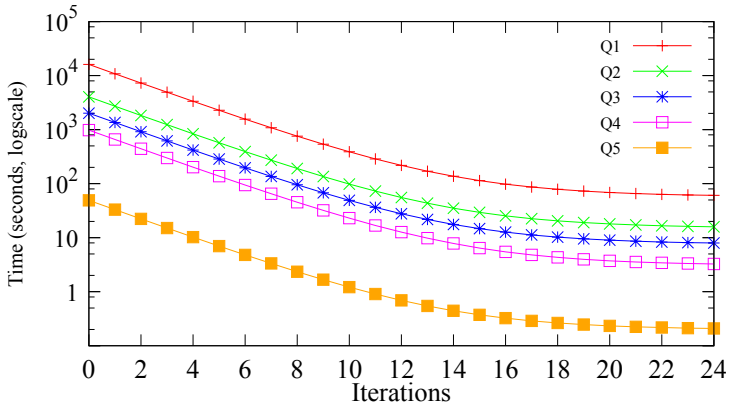
Figure 5.9: Evolution of the values of F and M using Q_5 throughout the iterations of the MIND algorithm.

Table 5.1: Consumed resources by F and M over MIND iterations, and gained time obtained between iterations for query Q_3 in high-bandwidth network ($10Gbps$).

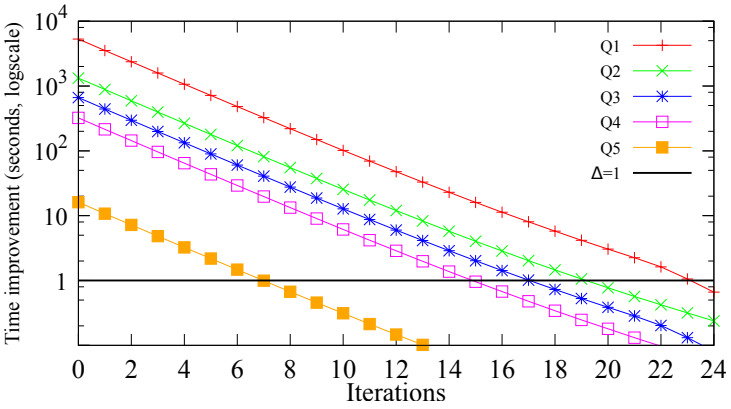
It.	F^B	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	1024	40	512	2 003.700		
1	1 538	57	682	2 003.700	1 353.000	650.740
2	2 311	86	889	1 353.000	905.340	447.660
3	3 474	129	1 135	905.340	612.200	293.140
4	5 224	193	1 428	612.200	410.300	201.890
5	7 859	291	1 779	410.300	278.010	132.290
6	11 829	438	2 203	278.010	188.640	89.370
7	17 817	660	2 719	188.640	128.230	60.413
8	26 860	995	3 349	128.230	88.232	39.998
9	40 535	1 501	4 122	88.232	60.280	27.952
10	61 250	2 269	5 074	60.280	42.158	18.122
11	92 691	3 433	6 249	42.158	29.318	12.840
12	140 520	5 205	7 702	29.318	20.905	8.413
13	213 490	7 907	9 502	20.905	14.997	5.908
14	325 150	12 043	11 740	14.997	10.941	4.057
15	496 640	18 394	14 528	10.941	8.196	2.745
16	761 060	28 187	18 012	8.196	6.245	1.951
17	**1 170 600**	**43 355**	**22382**	**6.245**	**4.878**	**1.367**
18	1 807 900	66 958	27 883	4.878	3.899	0.979
19	2 804 500	103 870	32 767	3.899	3.219	0.680
20	4 385 300	162 420	32 767	3.219	2.745	0.474
21	6 937 100	256 930	32 767	2.745	2.446	0.299
22	11 033 000	408 630	32 767	2.446	2.254	0.191
23	17 536 000	649 460	32 767	2.254	2.151	0.104
24	27 583 000	1 021 600	32 767	2.151	2.088	0.063

Table 5.2: Consumed resources by F and M over MIND iterations, and gained time obtained between iterations for query Q_5 in high bandwidth network ($10Gbps$).

It.	F^B	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	1 024	40	512	52.338		
1	1 538	53	682	52.338	35.341	16.997
2	2 311	80	889	35.341	23.648	11.693
3	3 474	120	1 135	23.648	15.991	7.657
4	5 224	180	1 428	15.991	10.717	5.273
5	7 859	271	1 779	10.717	7.262	3.456
6	11 829	408	2 203	7.262	4.928	2.334
7	17 817	614	2 719	4.928	3.350	1.578
8	**26 860**	**926**	**3 349**	**3.350**	**2.305**	**1.044**
9	40 535	1 398	4 122	2.305	1.575	0.730
10	61 250	2 112	5 074	1.575	1.101	0.473
11	92 691	3 196	6 249	1.101	0.766	0.335
12	140 520	4 846	7 702	0.766	0.547	0.220
13	213 490	7 362	9 502	0.547	0.392	0.154
14	325 150	11 212	11 740	0.392	0.286	0.107
15	496 640	17 125	14 528	0.286	0.215	0.071
16	761 060	26 243	18 012	0.215	0.163	0.052
17	1 170 600	40 365	22 382	0.163	0.128	0.035
18	1 807 900	62 340	27 883	0.128	0.103	0.025
19	2 804 500	96 706	32 767	0.103	0.085	0.018
20	4 385 300	151 220	32 767	0.085	0.073	0.012
21	6 937 100	239 210	32 767	0.073	0.065	0.008
22	11 033 000	380 450	32 767	0.065	0.061	0.003
23	17 536 000	604 670	32 767	0.061	0.059	0.002
24	27 583 000	951 130	32 767	0.059	0.064	0.005



(a) Estimated time – high-bandwidth network.



(b) Estimated time gain – high-bandwidth network.

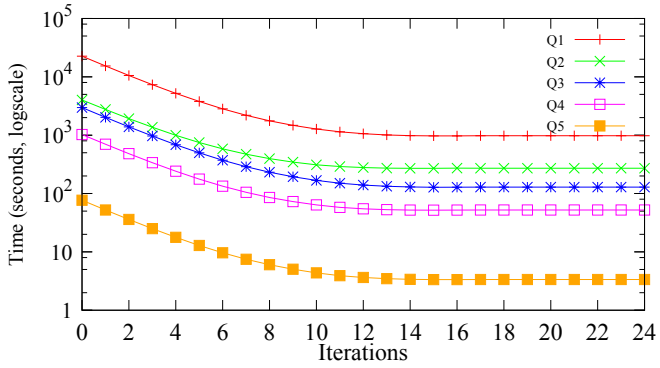
Figure 5.10: Estimated time and corresponding gain computed with the MIND optimization algorithm, for queries Q_1 to Q_5 in high-bandwidth network. We do not plot Q_6, which always needs only one iteration.

Network-adaptivity of MIND

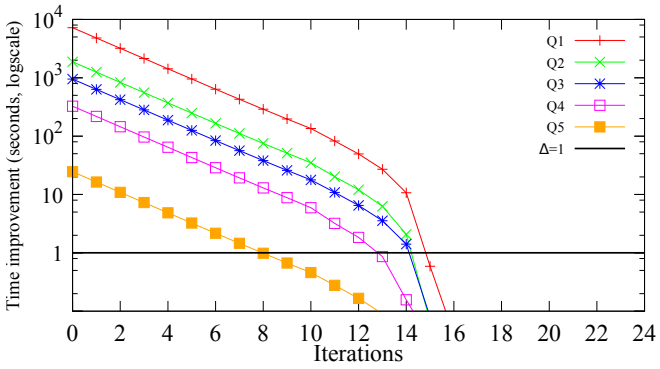
To point out that MIND adapts to the network environment, we discuss how the values of F and M returned by MIND change from a network to another (cf. Figure 5.2). As already pointed out, none of the competing strategies adapts to the network environment.

For each of the first five queries, we observe that MIND returns values of F and M that differ between the high- and the low-bandwidth networks. For instance, we observe that for the first three queries (which have the largest results), MIND favors large values of F in the high-bandwidth network (to exploit the pipelining of the messages from a large batch). The fact that the values of F increase throughout the iterations (particularly fast for the high-bandwidth network) can be noticed in Figures 5.5, 5.6, 5.7, 5.8 and 5.9. On the other hand, in the low-bandwidth network, MIND tends to quickly attain the maximum M value during the first iterations to compensate the choice of a smaller F compared to the high-bandwidth (this happens because the low-bandwidth network has more latency, hence MIND avoids large batches).

In Tables 5.1 and 5.3 we show how MIND optimization algorithm changes the values of



(a) Estimated time – low-bandwidth network.



(b) Estimated time gain – low-bandwidth network.

Figure 5.11: Estimated time and corresponding gain computed with the MIND optimization algorithm, for queries Q_1 to Q_5 in low-bandwidth network. We do not plot Q_6, which always needs only one iteration.

middleware parameters F and M over iterations for the same query Q_3 in high- and low-bandwidth networks, respectively.

Impact of iterations in MIND

We present in Figure 5.10(a) the estimated communication time over MIND iterations and in Figure 5.10(b) the corresponding gain between two consecutive iterations of the MIND optimization algorithm for five queries (Q_1-Q_5, except Q_6 which has a very small result set and the MIND optimization algorithm converge in one iteration). The estimated gain between two consecutive iterations is defined by the formula

$$C(\mathsf{F}_k^B, \mathsf{M}_k) - C(\mathsf{F}_{k+1}^B, \mathsf{M}_{k+1}).$$

Also, we focus on queries Q_3 and Q_5 and we report in Tables 5.1 and 5.2 the successive iterations done by MIND optimization algorithm.

We observe that with a threshold Δ set to a second, the optimization algorithm always stops after less than thirty iterations and the total time needed to compute these iterations is always

Table 5.3: Consumed resources by F and M over MIND iterations, and gained time obtained between iterations for query Q_3 in low-bandwidth network ($50Mbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	40	512	3 068.2		
1	60	766	2 929	1 993.05	935.95
2	90	1 143	1 993.13	1 369.1	624.03
3	136	1 704	1 369.1	952.96	416.1
4	206	2 532	952.96	675.43	277.53
5	311	3 745	675.43	490.24	185.19
6	473	5 507	490.24	366.54	123.7
7	721	8 034	366.54	283.75	82.784
8	1 107	11 600	283.75	228.17	55.586
9	1 710	16 542	228.17	190.64	37.528
10	2 659	23 261	190.64	165.1	25.539
11	4 159	32 234	165.1	147.54	17.563
12	6 539	32 767	147.54	137.32	10.221
13	10 310	32 767	137.32	131.22	6.098
14	16 124	32 767	131.22	127.98	3.24
15	**24 487**	**32 767**	**127.98**	**126.82**	**1.16**
16	34 505	32 767	126.82	127.05	0.235
17	41 324	32 767	127.05	127.61	0.555
18	40 523	32 767	127.61	127.53	0.075
19	40 940	32 767	127.53	127.57	0.039
20	40 735	32 767	127.57	127.55	0.019
21	40 839	32 767	127.55	127.56	0.01
22	40 787	32 767	127.56	127.56	0.005
23	40 813	32 767	127.56	127.56	0.002
24	40 800	32 767	127.56	127.56	0.001

less than a small fraction of a second. Next (in Section 5.4) we present an other alternative that can be used to control the halt condition in MIND optimization algorithm.

To illustrate how the values of F and M change during the iterations, we zoom on queries Q_1-Q_5 and show in Figures 5.5, 5.6, 5.7, 5.8 and 5.9 all intermediate values considered by the algorithm. More detail on the values and iterations of each query is reported in Appendix C.

A first observation is that MIND converges faster in the low-bandwidth network. This happens because, as already explained in the network-adaptivity paragraph, for such a network MIND quickly arrives at the maximum value of M and then chooses a value of F that is enough to obtain a gain below the threshold Δ. Moreover, we show in Figures 5.5, 5.6, 5.7, 5.8 and 5.9 the values of F and M, for queries Q_1, Q_2, Q_3, Q_4 *and* Q_5, respectively, that are considered by the MIND's optimization algorithm beyond the point where the gain is below Δ. For both networks, we observe that the resource consumption continues to increase. Since this increasing resource consumption implies only a small gain (quantified via comparison with threshold Δ), MIND decides to stop the iterations, and returns the current values of the two *middleware* parameters F and M.

5.4 Halt condition of the optimization algorithm

As introduced in previous section, MIND optimization algorithm is controlled via a threshold parameter that measures the improvement (gain of time) between two consecutive iterations. The halt condition could be improved by taking into account additional criteria. The general problem can be viewed as a multi-criteria optimization problem, in the sense that two parameters are worth to consider:

- The estimated improvement in communication time.
- The cost of consumed resources which can be expressed in terms of the cost of memory needed to store a batch of tuples (i.e., F^B) or as a message of M bytes to communicate data over the network.

In Example 5.2 we consider two approaches and we illustrate the practical behaviour of both approaches (*approach 1* and *approach 2*). For simplicity of presentation, we zoom only on the consumed resource by *middleware* parameter F over MIND iterations.

- *Approach 1* is exactly the algorithm defined in Section 5.2. Recall that the halt condition in this algorithm consists of optimizing the gained communication time i.e., the MIND optimization algorithm stops when the improvement given by $C(\mathsf{F}^B_k, \mathsf{M}_k) - C(\mathsf{F}^B_{k+1}, \mathsf{M}_{k+1}) \leq \Delta$, where Δ is a given threshold (in seconds).
- *Approach 2* is the same algorithm as defined in Section 5.2, except the halt condition. Concretely, we replace the halt condition by

$$\frac{\mathsf{F}^B_{k+1} \; - \; \mathsf{F}^B_k}{C(\mathsf{F}^B_k, \mathsf{M}_k) - C(\mathsf{F}^B_{k+1}, \mathsf{M}_{k+1})} \geq \Delta,$$

  where Δ is a given threshold (in KB/second). This halt condition focuses on the price in bytes to pay for a gained second between two consecutive iterations i.e., the MIND optimization algorithm stops when the price to pay for a gained second is greater than a given threshold.

Example 5.2. Take queries Q_3 (having a relative large query result of $4.5GB$) and Q_5 (having a relative small query result of $0.1GB$) (cf. Figure 3.3) in high-bandwidth network. We report in Table 5.4 and Table 5.5 the obtained information over MIND iterations for Q_3 and Q_5, respectively. In both tables, we report:

- Consumed F^B in the iterations k and $k+1$ in columns F^B_k and F^B_{k+1}, respectively.
- The difference of consumed F^B between two consecutive iterations (k and $k+1$) in column $(\mathsf{F}^B_{k+1} - \mathsf{F}^B_k)$.
- Communication time improvement between two consecutive iterations in column $C_k - C_{k+1}$. The formula is defined as

$$C(\mathsf{F}^B_k, \mathsf{M}_k) - C(\mathsf{F}^B_{k+1}, \mathsf{M}_{k+1})$$

  .
- Cost in KB for a gained second between two consecutive iterations (k and $k+1$) in column Cost (KB/Second). The formula is defined as

$$\frac{\mathsf{F}^B_{k+1} \; - \; \mathsf{F}^B_k}{C(\mathsf{F}^B_k, \mathsf{M}_k) - C(\mathsf{F}^B_{k+1}, \mathsf{M}_{k+1})}$$

From these tables, we stress that:

Table 5.4: Cost of consumed resource F versus gained time over MIND iterations using Q_3 in high bandwidth network ($10Gbps$).

It.	F^B_k (KB)	F^B_{k+1} (KB)	F^B_{k+1} - F^B_k (KB)	$C_k - C_{k+1}$	**Cost** $(KB$**/Second)**
1	1.00	1.50	0.50	650.740	0.0008
2	1.50	2.26	0.75	447.660	0.0017
3	2.26	3.39	1.14	293.140	0.0039
4	3.39	5.10	1.71	201.890	0.0085
5	5.10	7.67	2.57	132.290	0.0194
6	7.67	11.55	3.88	89.370	0.0434
7	11.55	17.40	5.85	60.413	0.0968
8	17.40	26.23	8.83	39.998	0.2208
9	26.23	39.58	13.35	27.952	0.4778
10	39.58	59.81	20.23	18.122	1.1163
11	59.81	90.52	30.70	12.840	2.3913
12	90.52	137.23	46.71	8.413	5.5518
13	137.23	208.49	71.26	5.908	12.0616
14	208.49	317.53	109.04	4.057	26.8804
15	**317.53**	**485.00**	**167.47**	**2.745**	**61.0094**
16	485.00	743.22	258.22	1.951	132.3540
17	**743.22**	**1 143.16**	**399.94**	**1.367**	**292.6115**
18	1 143.16	1 765.53	622.36	0.979	635.8756
19	1 765.53	2 738.77	973.24	0.680	1 431.9756
20	2 738.77	4 282.52	1 543.75	0.474	3 255.4829
21	4 282.52	6 774.51	2 491.99	0.299	8 327.1810
22	6 774.51	10 774.41	3 999.90	0.191	20 892.6735
23	10 774.41	17 125.00	6 350.59	0.104	61 210.4669
24	17 125.00	26 936.52	9 811.52	0.063	155 523.6964

- Using the *Approach 1* with threshold parameter (Δ_1=1 second), the MIND optimization algorithm stops at the iteration 17 with memory cost=292.61KB/second and memory consumption F^B=1 143.16KB. Whereas for query Q_5, the MIND optimization algorithm stops at the iteration 8 with memory cost=8.45$KB/second$ and memory consumption F^B=26.23KB.

- For *Approach 2*, MIND provides a different values of F and M comparatively to *Approach 1*. For instance, for threshold (Δ_2=100KB/second), MIND stops at iteration 15 for query Q_3 (cf. Table 5.4), with memory consumption F^B=485KB. Whereas it stops at iteration 11 for query Q_5 (cf. Table 5.5), with memory consumption F^B=90.52KB.

It is worth noting that both approaches can be combined, in the sense that MIND optimization algorithm stops when the first halt condition (Δ_1 or Δ_2) is satisfied.

□ ◊

5.5 Discussion

At the end of his chapter, we stress that:

- *Middleware* parameters F and M belong to a large research space (as explained in Section 5.1 where a naive approach needs to enumerate more than 35 billions of distinct combinations of F and M).

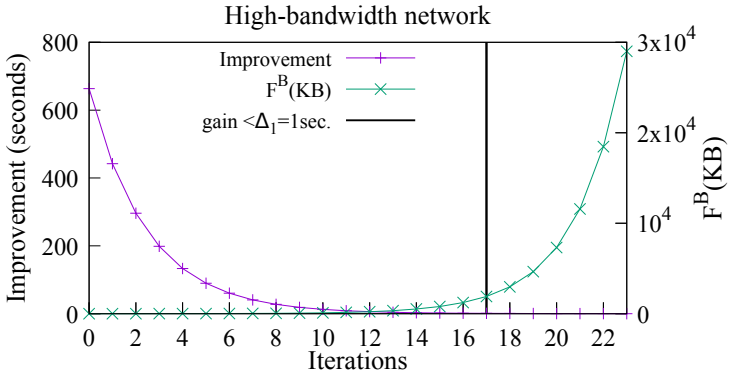
- MIND framework presents a good trade-off between communication time and consumed resource comparatively to the five strategies recommended by commercial DBMS_s.

Table 5.5: Cost of the resource consumed versus gained time over iterations done by MIND optimization algorithm using Q_5 in high bandwidth network ($10Gbps$).

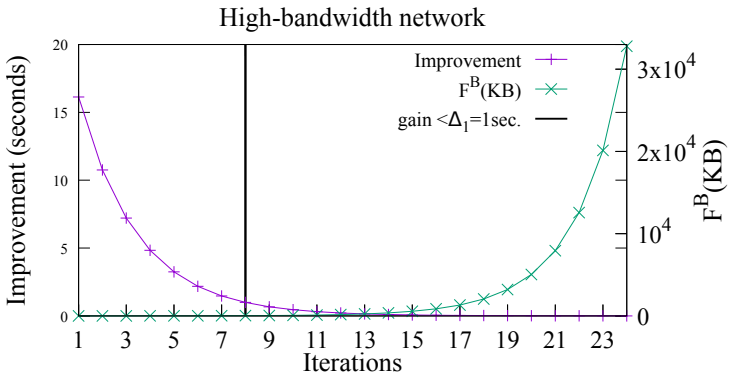
It.	F_k^B (KB)	F_{k+1}^B (KB)	F_{k+1}^B - F_k^B (KB)	$C_k - C_{k+1}$	**Cost** $(KB$**/Second)**
1	1.00	1.50	0.50	16.9970	0.030
2	1.50	2.26	0.75	11.6930	0.065
3	2.26	3.39	1.14	7.6569	0.148
4	3.39	5.10	1.71	5.2732	0.324
5	5.10	7.67	2.57	3.4556	0.745
6	7.67	11.55	3.88	2.3340	1.661
7	11.55	17.40	5.85	1.5784	3.705
8	**17.40**	**26.23**	**8.83**	**1.0443**	**8.456**
9	26.23	39.58	13.35	0.7304	18.284
10	39.58	59.81	20.23	0.4734	42.730
11	**59.81**	**90.52**	**30.70**	**0.3351**	**91.618**
12	90.52	137.23	46.71	0.2195	212.764
13	137.23	208.49	71.26	0.1543	461.976
14	208.49	317.53	109.04	0.1065	1023.493
15	317.53	485.00	167.47	0.0712	2353.406
16	485.00	743.22	258.22	0.0515	5013.740
17	743.22	1143.16	399.94	0.0350	11411.248
18	1143.16	1765.53	622.36	0.0253	24554.694
19	1765.53	2738.77	973.24	0.0177	54920.275
20	2738.77	4282.52	1543.75	0.0121	127151.800
21	4282.52	6774.51	2491.99	0.0083	300750.937
22	6774.51	10774.41	3999.90	0.0033	1218479.405
23	10774.41	17125.00	6350.59	0.0023	2708485.494
24	17125.00	26936.52	9811.52	0.0054	1819138.488

- Values of *middleware* parameters F and M provided by MIND are sensitive to query and network environment, in the sense that they are query- and network-dependant parameters.
- MIND optimization algorithm converges quickly (in small fraction of a second) to the optimal region of *middleware* parameters F and M.
- Execution of MIND is controlled via a threshold parameter, which can be tuned to find in order to find trade-off between the improvement in the communication time and the consumed resources.

All these elements point out the effectiveness of MIND framework that aims to minimize the time for communicating query result from DBMS node to client node with a rational consumption of resource.

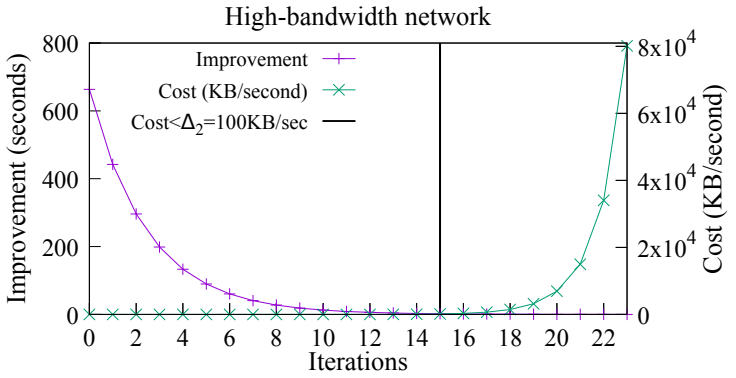


(a) Improvement communication times and resource consumption by F^B over iterations for Q_3

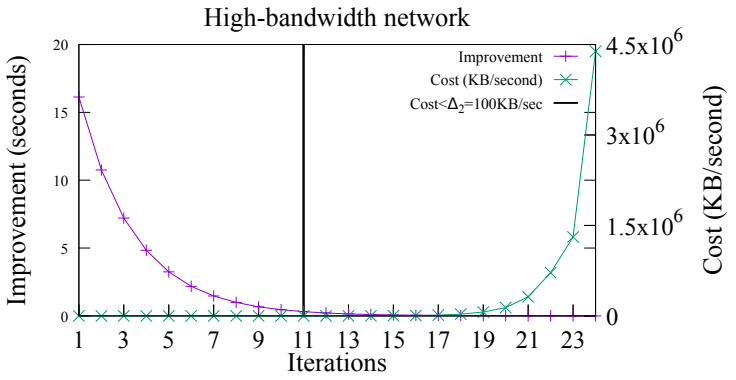


(b) Improvement communication times and resource consumption by F^B over iterations for Q_5

Figure 5.12: MIND using threshold according to communication time improvement (Δ=1 second in our case) using Q_3 and Q_5 in high-bandwidth network.

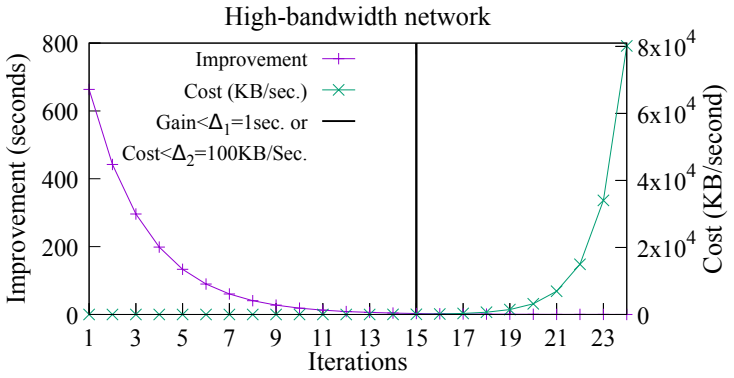


(a) Improvement communication times and cost of consumed resource F for Q_3 over iterations.

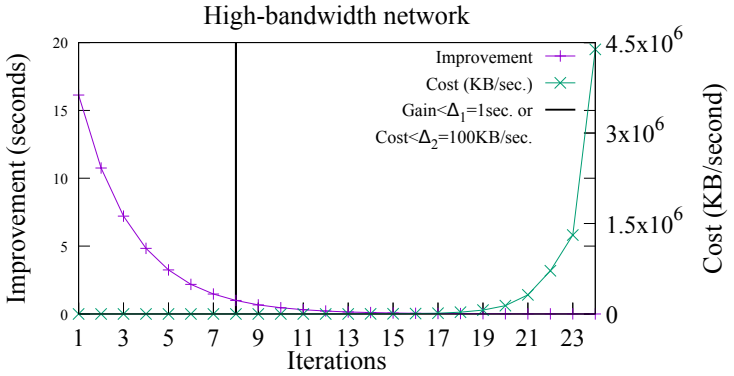


(b) Improvement communication times and cost of consumed resource F for Q_5 over iterations.

Figure 5.13: MIND using threshold according to memory cost (Δ=100KB/second in our case) using Q_3 and Q_5 in high-bandwidth network.



(a) Improvement communication times and cost of consumed resource F for Q_3 over iterations.



(b) Improvement communication times and cost of consumed resource F for Q_5 over iterations.

Figure 5.14: MIND using threshold according to the improvement (Δ_1=1 second in our case) and the memory cost (Δ_2=100KB/second in our case) in each improvement over iterations using Q_3 and Q_5 in high-bandwidth network.

Chapter 6

Conclusions

In our research work, we have taken a complementary look to the problem of optimizing the time for communicating query results in a distributed query processing, by focusing on *how data* is communicated over the network. To achieve this goal, we have investigated the relationship between the communication time and the *middleware* configuration. We have focused on two middleware parameters that are manually tuned by database administrators or programmers: the *fetch size*: F (i.e., the number of tuples that are communicated at once) and the *message size*: M (i.e., the size of the buffer at the middleware level).

We first motivated our research by presenting an intensive experimental study that emphasizes the impact of the middleware parameters F and M on the communication time. Also, we stress that tuning the middleware parameters is a non-trivial problem because the optimal values are *query-dependent* and *network-dependent*. Then, we designed MIND framework, which tunes the aforementioned middleware parameters, while adapting to different queries (that may vary in terms of selectivity) and networks (that may vary in terms of bandwidth). At the heart of the MIND framework, there is a cost model that estimates accurately the communication time. This cost model takes into account the middleware parameters F and M, together with the network environment and the volume of data to transfer. The MIND framework includes a calibration step that enables to tune the network dependent parameters. The cost model is exploited by an optimization algorithm that allows to compute for a given query and a network environment the F and M values while providing a good trade-off between optimized communication time and low resource consumption.

Looking ahead to future work, there are many directions for further investigation. We plan, in short term, to implement MIND framework inside a query optimizer in open-source DBMS. This is due to the fact that DBMS is the right place where the relevant information are available, in particular statistics and query plans.

Then, we project to refine the calibration process by investigating other approaches that allow *dynamic* calibration of a network-dependant parameters α and β. The *dynamic* calibration is important in the sense that (i) network-dependent parameters (α and β) are the core of the accuracy of the estimation cost function and (ii) the workload may vary in time (e.g., the number of concurrent queries may raise and the network characteristics may change).

Also, current MIND optimization algorithm uses a basic iterative optimization method namely the Newton method, we plan to explore more sophisticated optimization techniques using sub-gradient methods (e.g., Volume method [4]) which is more powerful for controlling the iterations steps and halt conditions, in order to ensure that the function value is decreasing over iterations.

In addition, such method can be more easily implemented than the Newton method because it does not require calculation of the Hessian matrix and its inverse.

We also project in medium-term to explore how MIND framework can be used to built an optimized query plan that minimizes the time for communicating query results between computation nodes. More precisely, our goal is to be able to optimize communication time between the operators inside a distributed query plan (c.f. Figure 6.1). To achieve this goal, we need to be capable to incorporate in our cost model the production and consumption rates of query plan operators and design a more sophisticated optimization algorithm that is apt to handle such a complex cost model.

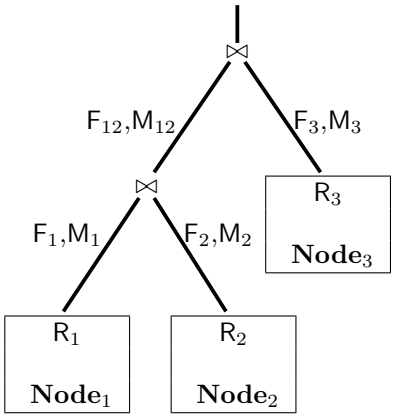


Figure 6.1: Simplified distributed query plan.

Bibliography

[1] Foto Afrati, Manas Joglekar, Christopher Re, Semih Salihoglu, and Jeffrey Ullman. Gym: A multiround join algorithm in mapreduce and its analysis.

[2] Foto N. Afrati and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, pages 99–110, 2010.

[3] Peter M. G. Apers. Data allocation in distributed database systems. *ACM*, 13:263–304, 1988.

[4] Francisco Barahona and Ranga Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, page 385–399, 2000.

[5] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013.

[6] Gamini Bulumulle, Sun Professional Services, and Sun Blueprints. Oracle middleware layer net8 performance tuning utilizing underlying network protocol, 2002.

[7] Donald K. Burleson. *Oracle Tuning: The Definitive Reference*. 2011.

[8] Duncan K.G. Campbell. A survey of models of parallel computation. Technical report, 1997.

[9] Surajit Chaudhuri and Vivek R. Narasayya. Self-tuning database systems: A decade of progress. In *VLDB*, pages 3–14, 2007.

[10] Alvin Cheung, Samuel Madden, and Armando Solar-Lezama. Sloth: Being lazy is a virtue (when issuing database queries). In *ACM SIGMOD*, pages 931–942, 2014.

[11] X/Open Company. Data management: Sql call level interface (x/open sql cli), 1995.

[12] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. 5th edition, 2011.

[13] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., 1st edition, 2012.

[14] Sumit Ganguly, Akshay Goel, and Avi Silberschatz. Efficient and accurate cost models for parallel query optimization (extended abstract). In *ACM SIGACT-SIGMOD-SIGART*, pages 172–181, 1996.

[15] Kyle Geiger. *Inside ODBC*. Microsoft Press, 1995.

[16] PostgreSQL Global Development Group. *PostgreSQL 9.4.0 Documentation*. 2014.

[17] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *VLDB*, pages 276–285, 1997.

[18] Hakan Hacigumus, Yun Chi, Wentao Wu, Shenghuo Zhu, Junichi Tatemura, and Jeffrey F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *IEEE ICDE*, pages 1081–1092, 2013.

[19] IBM. *Performance Monitoring and Tuning Guide*. 2010.

[20] IBM. *Application Programming Guide and Reference for Java*. 2013.

[21] IBM. *DB2 Connect User's Guide*. 2013.

[22] IBM. *Managing Performance*. 2015.

[23] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric query optimization. *The VLDB Journal*, pages 132–151, 1997.

[24] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, pages 938–948, 2010.

[25] Donald Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, pages 422–469, 2000.

[26] Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234, 2011.

[27] Mong-Li Lee, Masaru Kitsuregawa, Beng Chin Ooi, Kian-Lee Tan, and Anirban Mondal. Towards self-tuning data placement in parallel database systems. In *SIGMOD*, pages 225–236, 2000.

[28] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. *ACM Comput. Surv.*, 19:261–296, 1987.

[29] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *VLDB*, pages 149–159, 1986.

[30] Microsoft. Tabular data stream protocol, 2016. `https://msdn.microsoft.com/en-us/library/dd304523.aspx`.

[31] Shamkant B. Navathe and Mingyoung Ra. Vertical partitioning for database design: A graphical algorithm. In *ACM SIGMOD*, pages 440–450, 1989.

[32] J. Nocedal and S. J. Wright. *Numerical optimization, Second Edition*. Springer, 2006.

[33] Oracle. Oracle jdbc memory management, 2009. `https://http://www.oracle.com/technetwork/database/enterprise-edition/memory.pdf`.

[34] Oracle. *Oracle Database, JDBC Developer's Guide, version 11g Release2 (11.2)*. 2011.

[35] Oracle. Oracle database, net services administrator's guide, version 11g release2 (11.2). 2013.

[36] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer New York, 3rd edition, 2011.

[37] Karthik Ramachandra and S. Sudarshan. Holistic optimization by prefetching query results. In *SIGMOD*, pages 133–144, 2012.

[38] Kai-Uwe Sattler, Ingolf Geist, and Eike Schallehn. Quiet: Continuous query-driven index tuning. In *VLDB*, pages 1129–1132, 2003.

[39] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis. Automatic tcp buffer tuning. *Computer Communication Review*, pages 315–322, 1998.

[40] Jack Shirazi. *Java performance tuning*. O'Reilly, 2003.

[41] Immanuel Trummer and Christoph Koch. Multi-objective parametric query optimization. *SIGMOD Rec.*, pages 24–31, 2016.

[42] Eric Weigle and Wu-chun Feng. A comparison of tcp automatic tuning techniques for distributed computing. In *IEEE*, pages 265–, 2002.

[43] Pengcheng Xiong, Hakan Hacigumus, and Jeffrey F. Naughton. A software-defined networking based approach for performance management of analytical queries on distributed data stores. In *ACM SIGMOD*, pages 955–966, 2014.

Appendix A

Resource consumption according to middleware parameters

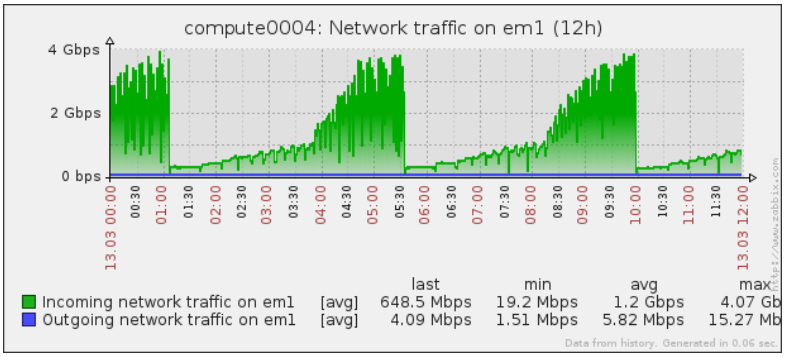


Figure A.1: Hard drive throughput of I/O operations in the execution of query Q_3 in DBMS node. One cycle (e.g., from 01:00 to 05:30) presents hard drive throughput of 37 configurations of F in high-bandwidth network (cf. Section 3.2.2).

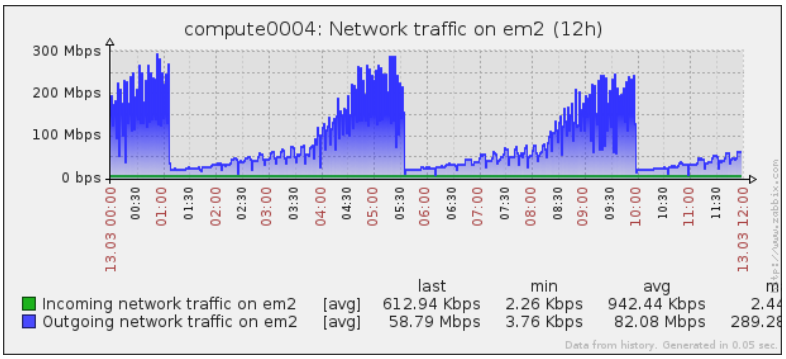


Figure A.2: Network traffic sent in the execution of query Q_3 by DBMS node. One cycle (e.g., from 01:00 to 05:30) presents consumed network bandwidth for 37 configurations of F in high-bandwidth network (cf. Section 3.2.2).

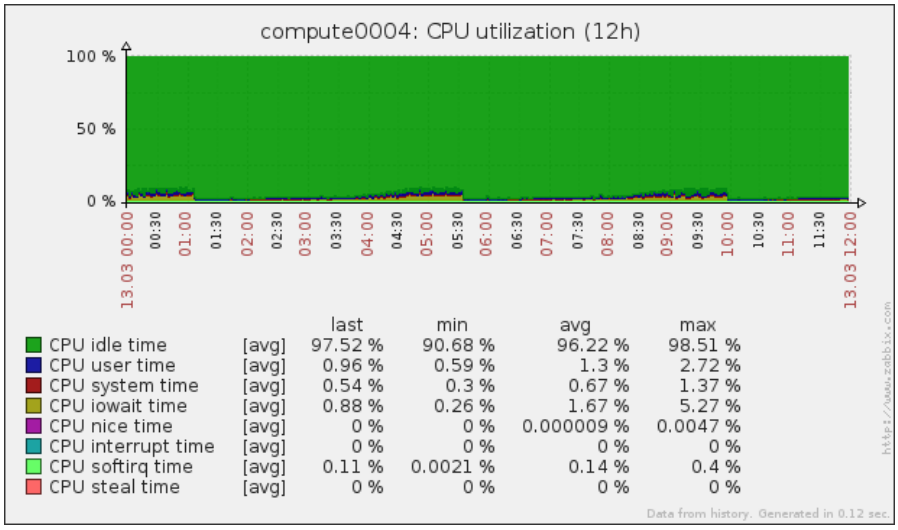


Figure A.3: *CPU* usage in the execution of query Q_3 in DBMS node. One cycle (e.g., from 01:00 to 05:30) presents *CPU* consumption of 37 configurations of F in high-bandwidth network (cf. Section 3.2.2).

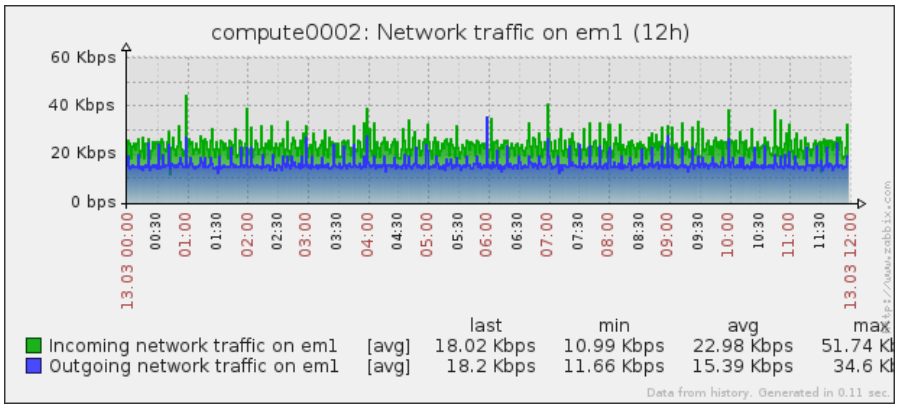


Figure A.4: Hard drive throughput of I/O operations in the execution of query Q_3 in client node. This figure presents hard drive throughput different configurations of F in high-bandwidth network (cf. Section 3.2.2).

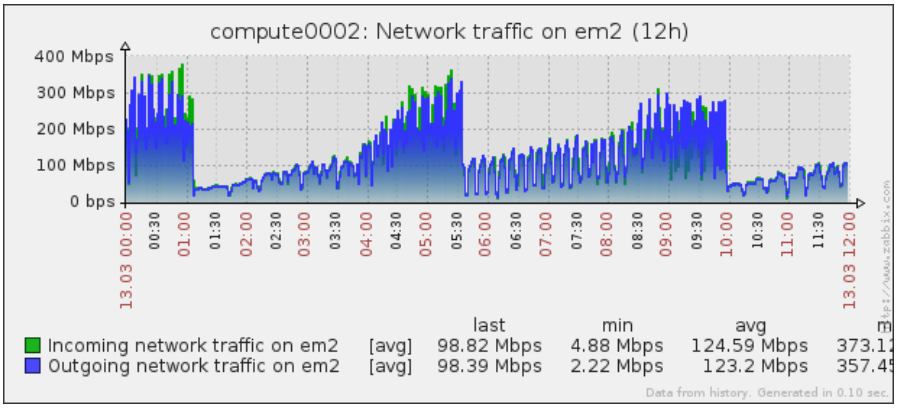


Figure A.5: Network traffic received by client node in the execution of query Q_3. One cycle (e.g., from 01:00 to 05:30) presents consumed network bandwidth for 37 configurations of F in high-bandwidth network (cf. Section 3.2.2).

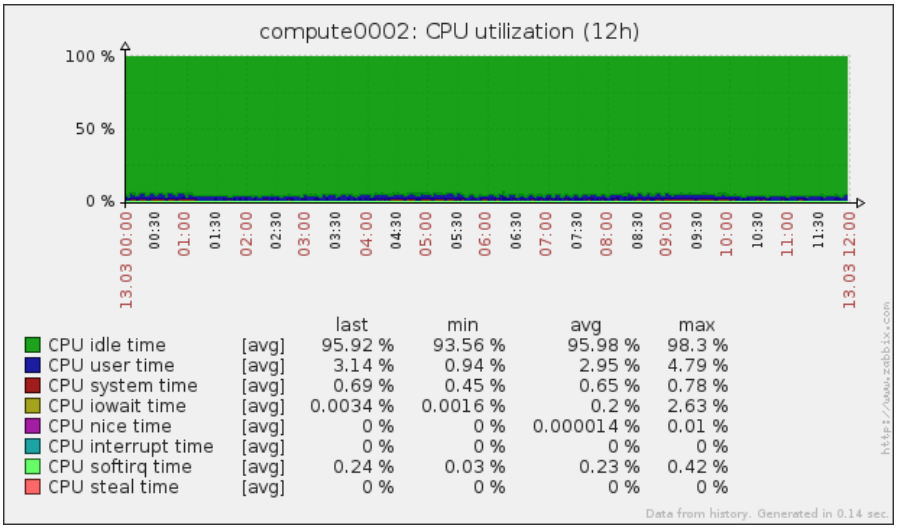


Figure A.6: *CPU* usage in the execution of query Q_3 in client node. This figure presents *CPU* consumption of different configurations of F in high-bandwidth network (cf. Section 3.2.2).

Appendix B

Real and estimated communication times

Table B.1: Real communication times using query Q_1 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	833.15	809.72	911.23	963.11	1081.4	1231.5	1240.1	1336.7	1348.9	1470.9	1574.3	1667.2	1662.5	1643.3	1568.3	1641.3	1628.6
220	399.97	382.74	446.20	461.77	509.58	572.55	579.92	632.32	649.36	712.44	706.38	748.42	807.99	797.55	824.20	876.25	882.84
330	307.26	279.86	305.36	319.55	345.76	376.12	379.41	418.38	418.77	460.31	485.20	504.01	533.81	519.15	546.47	575.17	597.74
440	240.75	237.71	247.79	241.44	259.28	279.71	287.74	311.17	305.38	340.46	353.15	366.25	386.43	386.50	401.30	404.08	439.71
550	199.72	183.71	174.57	186.87	196.31	209.43	211.24	242.94	253.59	268.15	272.65	288.32	300.11	298.74	318.77	304.76	334.76
660	195.98	185.49	182.92	176.30	189.31	194.15	193.57	217.40	226.13	235.92	231.64	257.32	257.24	256.76	274.57	265.33	282.08
770	175.60	169.34	164.50	157.73	162.09	169.04	173.54	183.61	191.19	200.40	201.88	221.11	218.36	217.49	236.82	233.22	237.30
880	153.03	133.44	142.02	137.59	128.95	148.56	145.45	164.29	163.07	174.34	176.38	190.56	193.35	194.32	199.23	196.03	210.85
990	158.54	139.23	137.07	128.43	121.30	138.74	138.23	155.42	144.38	164.84	159.34	179.22	176.29	173.22	188.53	178.97	191.22
1 100	148.95	131.69	129.58	114.95	109.70	123.04	124.41	142.97	131.48	146.66	145.10	158.24	165.12	159.96	174.21	168.35	176.49
2 200	108.00	84.05	80.24	77.01	73.31	79.88	79.02	84.43	80.13	89.67	88.84	94.95	97.43	99.23	104.46	103.95	108.88
3 300	93.58	73.03	62.44	58.65	55.10	58.45	57.17	59.10	60.25	64.31	65.13	67.63	68.14	70.22	72.59	72.81	76.52
4 400	95.44	64.48	53.96	50.83	46.98	48.83	47.73	49.49	48.97	52.20	52.82	54.49	56.20	57.28	58.04	58.00	58.72
5 500	94.70	57.76	47.36	45.33	41.94	42.25	42.09	42.90	42.34	43.22	44.58	46.05	46.31	47.75	48.13	48.45	49.26
6 600	85.02	59.43	43.52	40.96	37.67	37.85	37.30	37.47	37.53	39.28	39.81	40.14	40.45	42.00	42.33	42.51	42.75
7 700	88.85	50.44	37.56	38.51	47.08	33.67	34.27	33.94	33.53	35.29	35.05	35.93	36.64	38.14	37.80	37.94	38.90
8 800	86.10	59.17	36.14	36.54	32.43	32.18	31.50	31.21	30.99	32.74	32.61	33.42	33.48	35.02	34.19	34.12	35.15
9 900	86.38	55.15	36.49	34.93	31.38	30.42	29.47	29.99	29.41	30.92	30.17	31.49	31.06	32.44	31.60	31.76	32.22
$11x10^3$	85.77	55.70	34.62	33.96	30.14	29.54	28.38	28.56	27.74	29.43	28.67	30.00	28.77	29.73	29.70	29.79	30.22
$22x10^3$	45.84	39.08	26.47	27.65	23.88	23.21	20.73	21.24	19.73	20.99	20.54	20.54	20.19	20.37	20.27	19.88	20.48
$33x10^3$	45.44	37.26	25.36	25.43	21.08	20.79	17.92	18.38	17.02	18.97	17.21	18.09	16.75	17.21	16.90	16.42	16.52
$44x10^3$	46.54	36.95	25.34	24.68	21.01	20.78	18.12	18.48	16.46	18.76	17.18	17.38	16.10	16.33	16.16	15.76	15.56
$55x10^3$	48.09	37.48	25.75	24.41	21.08	20.90	18.33	18.79	16.93	18.95	17.19	17.66	16.49	16.15	15.71	15.22	14.84
$66x10^3$	47.83	38.49	25.44	24.49	20.87	20.73	18.69	18.92	17.56	18.76	17.16	17.65	16.46	16.34	15.63	15.09	14.51
$77x10^3$	47.31	38.98	25.36	24.12	20.72	20.45	18.57	19.07	17.21	18.73	17.15	17.29	16.42	15.87	15.25	14.85	14.05
$88x10^3$	46.65	39.24	25.50	24.05	20.99	20.58	18.60	18.90	17.71	18.58	17.11	17.43	16.35	16.08	15.06	14.43	14.27
$99x10^3$	45.88	38.56	25.54	23.62	21.05	20.63	18.79	18.98	18.09	18.37	17.23	17.92	16.14	16.03	15.09	14.52	13.93
$11x10^4$	46.71	36.06	25.62	23.65	21.01	20.43	19.18	19.26	17.80	18.63	17.13	17.82	16.09	15.86	14.83	14.04	13.90
$22x10^4$	42.69	35.20	25.70	23.61	20.74	20.77	19.43	19.82	18.67	19.86	17.98	17.58	16.04	15.92	14.56	14.19	14.03
$33x10^4$	42.05	33.93	26.38	23.90	21.45	21.76	19.43	20.24	18.27	20.42	18.27	17.94	15.87	15.93	14.43	14.13	13.81
$44x10^4$	41.06	33.27	26.86	23.62	21.09	22.34	19.78	20.41	18.65	20.25	17.81	18.00	16.43	15.78	14.49	14.52	13.51
$55x10^4$	41.26	33.04	26.20	24.14	21.54	21.99	19.85	21.82	19.39	20.15	17.85	17.15	16.75	15.40	15.17	14.18	13.16

Table B.2: Estimated communication times using query Q_1 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	880.21	890.10	961.19	1045.9	1134.1	1223.7	1313.9	1404.6	1495.4	1586.5	1677.6	1768.9	1769.9	1769.9	1769.9	1769.9	1769.9
220	497.74	493.82	519.09	558.05	600.44	644.19	688.63	733.46	778.54	823.77	869.13	914.56	960.06	1005.6	1051.1	1096.8	1142.4
330	370.25	361.72	371.73	395.42	422.54	451.02	480.19	509.75	539.56	569.52	599.61	629.77	660.00	690.27	720.59	750.93	781.28
440	306.51	295.67	298.04	314.10	333.58	354.44	375.97	397.90	420.07	442.40	464.85	487.38	509.97	532.61	555.29	578.00	600.72
550	268.26	256.04	253.83	265.31	280.21	296.48	313.44	330.78	348.37	366.13	383.99	401.94	419.95	438.01	456.11	474.24	492.38
660	242.77	229.63	224.36	232.79	244.63	257.85	271.75	286.04	300.58	315.28	330.09	344.98	359.94	374.95	389.99	405.07	420.16
770	224.55	210.75	203.31	209.55	219.22	230.25	241.97	254.08	266.44	278.95	291.59	304.30	317.08	329.90	342.76	355.66	368.57
880	210.89	196.60	187.52	192.13	200.16	209.56	219.64	230.11	240.83	251.71	262.71	273.79	284.93	296.12	307.34	318.60	329.88
990	200.27	185.59	175.24	178.57	185.33	193.46	202.27	211.47	220.92	230.53	240.25	250.06	259.92	269.84	279.79	289.78	299.78
1 100	191.77	176.79	165.42	167.73	173.47	180.58	188.37	196.56	204.98	213.58	222.28	231.07	239.92	248.82	257.75	266.72	275.71
2 200	153.52	137.16	121.21	118.94	120.10	122.63	125.84	129.44	133.29	137.30	141.43	145.63	149.90	154.22	158.58	162.96	167.37
3 300	140.78	123.95	106.47	102.68	102.31	103.31	105.00	107.07	109.39	111.88	114.47	117.15	119.90	122.69	125.52	128.37	131.25
4 400	134.40	117.34	99.10	94.55	93.42	93.65	94.57	95.89	97.44	99.16	101.00	102.91	104.89	106.92	108.99	111.08	113.20
5 500	130.58	113.38	94.68	89.67	88.08	87.86	88.32	89.18	90.27	91.54	92.91	94.37	95.89	97.46	99.07	100.71	102.37
6 600	128.03	110.74	91.73	86.42	84.52	83.99	84.15	84.70	85.49	86.45	87.52	88.68	89.89	91.16	92.46	93.79	95.14
7 700	126.21	108.85	89.63	84.09	81.98	81.23	81.17	81.51	82.08	82.82	83.67	84.61	85.61	86.65	87.73	88.85	89.98
8 800	124.84	107.44	88.05	82.35	80.07	79.16	78.94	79.11	79.52	80.09	80.78	81.56	82.39	83.27	84.19	85.14	86.11
9 900	123.78	106.34	86.82	80.99	78.59	77.55	77.20	77.24	77.53	77.98	78.54	79.18	79.89	80.64	81.44	82.26	83.11
$11x10^3$	122.93	105.46	85.84	79.91	77.40	76.27	75.81	75.75	75.94	76.28	76.74	77.28	77.89	78.54	79.23	79.95	80.70
$22x10^3$	119.10	101.49	81.42	75.03	72.07	70.47	69.56	69.04	68.77	68.65	68.66	68.74	68.89	69.08	69.31	69.58	69.86
$33x10^3$	117.83	100.17	79.94	73.40	70.29	68.54	67.48	66.80	66.38	66.11	65.96	65.89	65.89	65.93	66.01	66.12	66.25
$44x10^3$	117.19	99.51	79.21	72.59	69.40	67.57	66.43	65.69	65.18	64.84	64.61	64.47	64.39	64.35	64.36	64.39	64.45
$55x10^3$	116.81	99.11	78.76	72.10	68.86	66.99	65.81	65.01	64.46	64.08	63.80	63.61	63.49	63.41	63.36	63.35	63.36
$66x10^3$	116.55	98.85	78.47	71.78	68.51	66.61	65.39	64.57	63.99	63.57	63.26	63.04	62.89	62.78	62.70	62.66	62.64
$77x10^3$	116.37	98.66	78.26	71.55	68.25	66.33	65.09	64.25	63.64	63.21	62.88	62.64	62.46	62.32	62.23	62.17	62.13
$88x10^3$	116.23	98.52	78.10	71.37	68.06	66.12	64.87	64.01	63.39	62.93	62.59	62.33	62.14	61.99	61.88	61.79	61.74
$99x10^3$	116.13	98.41	77.98	71.24	67.92	65.96	64.70	63.82	63.19	62.72	62.37	62.10	61.89	61.72	61.60	61.51	61.44
$11x10^4$	116.04	98.32	77.88	71.13	67.80	65.84	64.56	63.67	63.03	62.55	62.19	61.90	61.69	61.51	61.38	61.28	61.20
$22x10^4$	115.66	97.93	77.44	70.64	67.26	65.26	63.93	63.00	62.31	61.79	61.38	61.05	60.79	60.57	60.39	60.24	60.11
$33x10^4$	115.53	97.79	77.29	70.48	67.09	65.06	63.72	62.78	62.07	61.53	61.11	60.77	60.49	60.25	60.06	59.89	59.75
$44x10^4$	115.47	97.73	77.22	70.40	67.00	64.97	63.62	62.66	61.95	61.41	60.97	60.62	60.34	60.09	59.89	59.72	59.57
$55x10^4$	115.43	97.69	77.17	70.35	66.94	64.91	63.56	62.60	61.88	61.33	60.89	60.54	60.25	60.00	59.79	59.62	59.46

Table B.3: Real communication times using query Q_2 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	549.37	535.72	632.85	738.39	815.31	810.82	811.11	791.31	788.10	744.30	828.11	798.95	830.58	806.97	795.60	822.90	815.58
220	266.00	275.98	307.65	340.30	369.75	424.05	453.33	459.85	451.73	438.33	477.12	464.72	473.90	465.61	462.02	476.05	480.32
330	206.13	194.60	231.17	240.32	264.62	293.80	317.07	318.33	328.31	356.78	375.93	368.27	376.00	369.62	374.41	377.72	385.72
440	169.54	160.96	177.20	191.01	200.56	230.29	240.20	239.04	254.85	267.89	290.15	300.37	318.65	326.65	314.11	321.77	324.38
550	145.74	132.11	151.71	157.44	169.04	187.05	199.59	203.71	209.93	216.96	232.59	239.51	253.88	258.00	277.98	286.95	292.65
660	119.82	116.02	123.38	129.49	137.93	152.08	161.97	168.77	172.55	180.60	186.69	195.47	207.49	212.54	223.69	235.87	243.95
770	100.68	93.24	107.13	111.57	116.62	127.39	136.19	142.47	143.25	152.22	162.97	168.45	175.19	177.05	188.18	198.03	198.59
880	95.46	87.79	95.17	99.44	105.46	112.75	118.47	126.29	127.83	132.45	142.25	146.57	154.16	158.49	166.40	177.49	180.31
990	88.91	82.29	91.49	94.20	96.63	103.70	111.11	114.26	116.93	122.50	130.69	133.26	142.96	144.71	151.58	160.17	166.42
1 100	86.82	80.18	85.25	87.23	86.36	98.41	101.71	102.77	108.09	113.96	119.41	124.82	129.88	129.39	136.77	146.20	149.31
2 200	50.34	47.87	53.16	52.27	51.38	52.19	56.48	56.65	59.90	64.50	63.83	65.06	71.13	68.78	71.43	75.60	80.49
3 300	51.11	37.48	39.88	40.88	42.21	41.28	42.23	43.78	43.72	46.84	48.72	45.35	49.41	51.32	51.45	52.53	56.05
4 400	45.23	37.76	33.71	34.18	33.67	33.99	36.18	34.87	32.99	37.81	40.97	40.92	37.58	41.14	43.32	44.10	46.73
5 500	39.85	29.40	31.27	29.26	27.19	29.79	31.40	31.19	27.09	32.65	33.64	33.42	35.78	33.38	34.95	35.25	37.23
6 600	42.68	30.38	29.45	27.42	30.62	28.02	29.03	27.10	28.48	30.73	29.24	30.59	28.72	33.76	32.01	33.16	39.05
7 700	40.69	28.71	29.79	24.73	24.65	25.94	24.96	24.81	23.03	24.97	28.85	29.56	27.45	26.34	30.09	28.16	30.05
8 800	41.05	27.29	26.06	24.69	26.09	25.71	24.66	22.85	22.00	25.96	27.08	25.54	22.80	25.35	29.63	30.33	29.29
9 900	38.40	28.38	26.39	23.18	22.42	24.86	22.05	24.53	21.06	26.77	25.06	29.26	26.41	24.36	25.12	24.49	25.06
$11x10^3$	37.69	27.56	27.99	20.97	24.00	22.54	18.65	24.00	23.04	23.76	22.82	22.85	20.96	25.09	25.08	25.16	25.46
$22x10^3$	35.10	26.38	20.35	18.91	14.95	15.29	15.78	16.35	18.17	15.03	18.37	17.35	18.75	17.98	14.50	15.85	18.61
$33x10^3$	37.00	19.81	18.83	17.33	14.43	14.85	13.81	14.46	12.05	16.04	14.54	12.60	11.10	12.81	14.20	15.66	12.39
$44x10^3$	32.98	17.33	16.81	13.69	12.30	14.14	10.85	14.82	10.93	10.15	13.72	11.44	11.86	15.03	14.62	11.14	12.60
$55x10^3$	34.55	20.11	16.43	12.09	12.42	14.56	11.75	11.89	12.00	12.23	12.54	13.46	8.89	11.20	12.00	11.67	9.71
$66x10^3$	28.54	16.38	15.56	11.50	10.40	10.92	11.21	9.42	11.20	10.94	10.79	10.00	9.24	11.15	9.21	10.77	10.47
$77x10^3$	28.95	17.01	13.52	12.05	11.98	12.75	13.19	11.38	8.65	10.31	9.39	10.79	9.34	8.42	11.26	12.27	8.16
$88x10^3$	24.94	20.38	12.94	12.84	10.75	10.64	9.32	8.52	9.93	9.38	7.96	7.96	9.10	7.38	8.59	8.45	6.93
$99x10^3$	22.61	17.79	12.42	12.08	10.28	14.40	10.35	12.09	11.98	9.25	7.43	9.36	8.53	8.50	10.91	8.79	8.43
$11x10^4$	24.05	16.19	13.59	10.40	10.71	10.58	9.72	9.70	10.36	7.90	7.03	8.13	8.55	7.82	8.85	6.84	6.23
$22x10^4$	17.57	12.70	8.54	9.20	8.81	8.87	7.79	6.87	6.45	7.08	5.58	5.89	5.25	5.08	5.79	6.24	5.53
$33x10^4$	16.37	11.62	7.63	8.12	6.67	7.08	6.42	5.86	6.35	5.86	4.78	5.15	4.89	4.51	4.54	4.69	3.96
$44x10^4$	15.28	10.63	7.93	7.32	6.98	6.97	6.11	6.33	5.57	5.28	5.03	5.46	4.73	4.25	5.31	4.26	3.60
$55x10^4$	14.75	10.28	7.55	7.09	6.48	6.64	6.17	5.40	6.17	5.27	5.07	5.33	4.61	4.51	4.55	4.37	3.73
$66x10^4$	14.21	10.38	7.26	6.83	6.52	6.50	6.09	6.00	5.62	5.08	4.86	4.72	4.73	4.20	4.60	4.32	3.56
$77x10^4$	13.11	9.77	7.25	7.20	6.41	6.13	5.94	5.29	5.25	5.15	4.86	4.63	4.39	4.58	4.16	4.21	3.54
$88x10^4$	12.92	9.58	7.22	6.42	6.79	6.51	5.85	5.45	5.54	5.41	4.77	4.55	4.48	4.25	4.47	4.10	3.12
$99x10^4$	12.59	9.71	7.25	6.65	6.39	5.98	5.97	5.23	5.60	5.08	4.82	4.70	4.81	4.66	4.83	4.39	3.44
$11x10^5$	13.10	9.45	6.92	6.34	6.43	6.11	5.85	5.27	5.46	5.09	4.98	4.65	4.50	4.18	4.42	4.03	3.15

Table B.4: Estimated communication times using query Q_2 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	549.37	535.72	632.85	738.39	815.31	810.82	811.11	791.31	788.10	744.30	828.11	798.95	830.58	806.97	795.60	822.90	815.58
220	266.00	275.98	307.65	340.30	369.75	424.05	453.33	459.85	451.73	438.33	477.12	464.72	473.90	465.61	462.02	476.05	480.32
330	206.13	194.60	231.17	240.32	264.62	293.80	317.07	318.33	328.31	356.78	375.93	368.27	376.00	369.62	374.41	377.72	385.72
440	169.54	160.96	177.20	191.01	200.56	230.29	240.20	239.04	254.85	267.89	290.15	300.37	318.65	326.65	314.11	321.77	324.38
550	145.74	132.11	151.71	157.44	169.04	187.05	199.59	203.71	209.93	216.96	232.59	239.51	253.88	258.00	277.98	286.95	292.65
660	119.82	116.02	123.38	129.49	137.93	152.08	161.97	168.77	172.55	180.60	186.69	195.47	207.49	212.54	223.69	235.87	243.95
770	100.68	93.24	107.13	111.57	116.62	127.39	136.19	142.47	143.25	152.22	162.97	168.45	175.19	177.05	188.18	198.03	198.59
880	95.46	87.79	95.17	99.44	105.46	112.75	118.47	126.29	127.83	132.45	142.25	146.57	154.16	158.49	166.40	177.49	180.31
990	88.91	82.29	91.49	94.20	96.63	103.70	111.11	114.26	116.93	122.50	130.69	133.26	142.96	144.71	151.58	160.17	166.42
1 100	86.82	80.18	85.25	87.23	86.36	98.41	101.71	102.77	108.09	113.96	119.41	124.82	129.88	129.39	136.77	146.20	149.31
2 200	50.34	47.87	53.16	52.27	51.38	52.19	56.48	56.65	59.90	64.50	63.83	65.06	71.13	68.78	71.43	75.60	80.49
3 300	51.11	37.48	39.88	40.88	42.21	41.28	42.23	43.78	43.72	46.84	48.72	45.35	49.41	51.32	51.45	52.53	56.05
4 400	45.23	37.76	33.71	34.18	33.67	33.99	36.18	34.87	32.99	37.81	40.97	40.92	37.58	41.14	43.32	44.10	46.73
5 500	39.85	29.40	31.27	29.26	27.19	29.79	31.40	31.19	27.09	32.65	33.64	33.42	35.78	33.38	34.95	35.25	37.23
6 600	42.68	30.38	29.45	27.42	30.62	28.02	29.03	27.10	28.48	30.73	29.24	30.59	28.72	33.76	32.01	33.16	39.05
7 700	40.69	28.71	29.79	24.73	24.65	25.94	24.96	24.81	23.03	24.97	28.85	29.56	27.45	26.34	30.09	28.16	30.05
8 800	41.05	27.29	26.06	24.69	26.09	25.71	24.66	22.85	22.00	25.96	27.08	25.54	22.80	25.35	29.63	30.33	29.29
9 900	38.40	28.38	26.39	23.18	22.42	24.86	22.05	24.53	21.06	26.77	25.06	29.26	26.41	24.36	25.12	24.49	25.06
$11x10^3$	37.69	27.56	27.99	20.97	24.00	22.54	18.65	24.00	23.04	23.76	22.82	22.85	20.96	25.09	25.08	25.16	25.46
$22x10^3$	35.10	26.38	20.35	18.91	14.95	15.29	15.78	16.35	18.17	15.03	18.37	17.35	18.75	17.98	14.50	15.85	18.61
$33x10^3$	37.00	19.81	18.83	17.33	14.43	14.85	13.81	14.46	12.05	16.04	14.54	12.60	11.10	12.81	14.20	15.66	12.39
$44x10^3$	32.98	17.33	16.81	13.69	12.30	14.14	10.85	14.82	10.93	10.15	13.72	11.44	11.86	15.03	14.62	11.14	12.60
$55x10^3$	34.55	20.11	16.43	12.09	12.42	14.56	11.75	11.89	12.00	12.23	12.54	13.46	8.89	11.20	12.00	11.67	9.71
$66x10^3$	28.54	16.38	15.56	11.50	10.40	10.92	11.21	9.42	11.20	10.94	10.79	10.00	9.24	11.15	9.21	10.77	10.47
$77x10^3$	28.95	17.01	13.52	12.05	11.98	12.75	13.19	11.38	8.65	10.31	9.39	10.79	9.34	8.42	11.26	12.27	8.16
$88x10^3$	24.94	20.38	12.94	12.84	10.75	10.64	9.32	8.52	9.93	9.38	7.96	7.96	9.10	7.38	8.59	8.45	6.93
$99x10^3$	22.61	17.79	12.42	12.08	10.28	14.40	10.35	12.09	11.98	9.25	7.43	9.36	8.53	8.50	10.91	8.79	8.43
$11x10^4$	24.05	16.19	13.59	10.40	10.71	10.58	9.72	9.70	10.36	7.90	7.03	8.13	8.55	7.82	8.85	6.84	6.23
$22x10^4$	17.57	12.70	8.54	9.20	8.81	8.87	7.79	6.87	6.45	7.08	5.58	5.89	5.25	5.08	5.79	6.24	5.53
$33x10^4$	16.37	11.62	7.63	8.12	6.67	7.08	6.42	5.86	6.35	5.86	4.78	5.15	4.89	4.51	4.54	4.69	3.96
$44x10^4$	15.28	10.63	7.93	7.32	6.98	6.97	6.11	6.33	5.57	5.28	5.03	5.46	4.73	4.25	5.31	4.26	3.60
$55x10^4$	14.75	10.28	7.55	7.09	6.48	6.64	6.17	5.40	6.17	5.27	5.07	5.33	4.61	4.51	4.55	4.37	3.73
$66x10^4$	14.21	10.38	7.26	6.83	6.52	6.50	6.09	6.00	5.62	5.08	4.86	4.72	4.73	4.20	4.60	4.32	3.56
$77x10^4$	13.11	9.77	7.25	7.20	6.41	6.13	5.94	5.29	5.25	5.15	4.86	4.63	4.39	4.58	4.16	4.21	3.54
$88x10^4$	12.92	9.58	7.22	6.42	6.79	6.51	5.85	5.45	5.54	5.41	4.77	4.55	4.48	4.25	4.47	4.10	3.12
$99x10^4$	12.59	9.71	7.25	6.65	6.39	5.98	5.97	5.23	5.60	5.08	4.82	4.70	4.81	4.66	4.83	4.39	3.44
$11x10^5$	13.10	9.45	6.92	6.34	6.43	6.11	5.85	5.27	5.46	5.09	4.98	4.65	4.50	4.18	4.42	4.03	3.15

Table B.5: Real communication times using query Q_3 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	505.61	529.86	628.12	633.13	630.26	647.15	635.61	604.12	555.52	571.93	550.36	618.30	644.49	634.30	637.54	599.72	552.72
220	268.42	283.73	307.61	347.15	370.22	376.43	370.80	356.71	319.04	339.52	310.20	368.53	377.63	371.57	375.11	350.42	322.14
330	177.22	193.99	210.83	226.54	252.72	272.03	267.49	257.32	233.04	246.70	226.75	258.56	266.58	261.33	267.70	246.42	240.75
440	131.43	139.20	160.31	170.43	182.91	205.63	211.54	208.85	187.12	198.08	185.52	208.03	213.61	211.58	224.35	198.06	197.21
550	110.79	116.15	130.58	139.60	147.25	166.46	173.53	175.39	163.37	174.49	166.85	182.98	188.63	184.92	185.63	173.05	173.52
660	95.50	97.75	112.70	117.04	125.39	140.53	145.56	148.54	139.27	150.21	152.88	166.58	171.77	167.61	169.26	160.04	155.89
770	85.39	86.07	94.11	103.36	110.14	121.20	124.97	128.85	122.33	135.60	135.12	155.12	156.72	156.89	157.69	147.19	142.56
880	78.90	78.91	84.94	91.20	97.71	107.48	111.58	112.56	110.02	117.65	120.42	138.74	143.51	146.50	157.27	137.58	134.62
990	71.36	70.68	76.85	82.65	86.81	96.05	98.80	100.67	95.37	106.56	106.69	120.80	133.58	132.53	137.90	126.28	127.13
1100	65.45	64.99	68.60	74.92	78.87	87.26	89.58	91.59	86.30	96.32	96.65	110.18	117.64	122.57	127.11	118.94	119.25
2200	39.40	38.78	36.89	37.28	39.94	45.12	46.34	46.56	43.90	48.86	49.37	53.68	57.21	58.42	62.02	59.61	59.23
3300	30.34	29.74	26.34	25.85	27.02	29.95	31.00	31.36	29.44	32.00	33.19	35.47	38.77	39.65	40.64	39.29	41.81
4400	27.49	27.09	20.80	20.34	21.07	23.00	23.98	23.58	22.49	25.78	25.49	27.07	28.52	27.83	30.19	29.69	31.55
5500	24.52	24.51	17.70	17.56	17.33	19.21	19.75	19.50	18.66	19.65	20.66	21.64	23.18	23.07	24.42	24.53	25.32
6600	26.59	22.55	15.33	14.92	14.83	16.00	16.61	16.19	15.70	16.86	17.52	17.99	19.23	19.82	20.29	20.22	21.39
7700	22.72	21.85	13.90	13.48	13.08	14.47	14.47	13.97	13.50	15.03	15.56	15.48	15.95	16.79	18.45	17.01	18.63
8800	23.75	21.42	12.73	12.17	12.03	12.70	13.08	12.55	11.93	12.34	13.87	13.33	13.84	14.74	14.88	15.47	16.09
9900	23.09	20.70	11.34	11.35	10.91	11.88	11.85	11.42	11.25	11.96	12.98	11.91	12.35	13.01	13.53	13.67	14.57
$11x10^3$	22.28	19.44	10.79	10.53	10.25	10.83	10.55	10.81	10.01	10.43	11.37	11.45	10.99	11.29	12.06	12.15	13.49
$22x10^3$	20.03	17.01	7.90	7.30	6.88	7.33	6.96	6.69	6.51	7.13	7.55	6.62	7.02	6.92	7.09	7.01	8.23
$33x10^3$	20.04	17.01	6.58	6.46	5.95	6.37	5.84	5.96	5.44	5.55	5.80	5.08	5.20	5.34	5.64	5.78	7.06
$44x10^3$	18.83	17.42	6.15	5.97	5.34	5.53	5.45	5.05	4.56	4.97	4.64	4.58	4.32	4.60	4.95	4.88	5.44
$55x10^3$	18.93	9.80	5.84	5.67	4.98	5.19	4.71	4.52	4.22	4.86	4.68	3.96	4.09	4.30	4.44	4.30	7.87
$66x10^3$	18.56	15.79	5.49	5.43	4.76	5.02	4.64	4.08	3.87	4.21	4.01	3.63	4.05	3.70	3.85	3.85	4.23
$77x10^3$	18.43	9.49	5.41	5.50	4.63	4.67	4.23	3.85	3.52	3.63	3.76	3.39	3.63	3.27	3.51	3.49	3.94
$88x10^3$	18.05	13.74	5.18	5.16	4.47	4.53	3.86	3.82	3.49	4.08	3.33	3.29	3.20	3.44	3.14	3.53	3.63
$99x10^3$	18.56	13.80	5.01	4.88	4.26	4.82	4.04	3.78	3.46	3.62	4.01	3.14	3.04	2.92	3.05	3.44	3.48
$11x10^4$	18.31	13.71	5.09	4.95	4.23	4.27	4.18	3.72	3.55	3.45	3.99	2.99	2.84	3.03	2.87	3.21	3.03
2210^4	12.08	10.34	4.77	4.38	3.76	4.02	3.47	3.10	2.71	2.70	2.71	2.53	2.60	2.27	2.29	2.69	2.98
3310^4	11.99	9.57	4.22	4.31	3.58	3.86	3.15	2.90	2.54	2.55	2.85	2.27	2.36	2.30	2.11	2.59	2.42
4410^4	11.77	8.63	4.41	4.02	3.58	3.64	3.67	2.94	2.79	2.54	3.49	2.19	2.05	2.05	2.24	2.01	2.25
5510^4	11.49	8.42	4.20	4.15	3.69	3.67	3.22	3.01	2.71	2.56	3.10	2.16	2.14	2.03	2.04	2.33	2.06
6610^4	11.37	7.98	5.02	3.94	3.50	3.63	3.13	3.05	2.67	2.60	5.00	2.44	2.21	2.25	2.13	2.35	2.35
7710^4	10.81	7.46	3.95	3.85	3.39	3.71	3.02	2.90	2.67	2.94	2.50	2.16	2.24	2.22	2.18	2.56	2.31
8810^4	10.40	7.36	4.38	3.76	3.38	3.62	3.27	2.86	2.86	2.56	3.12	2.18	2.06	2.32	2.09	2.00	1.94
9910^4	10.24	7.25	4.12	3.95	3.40	3.74	3.04	2.98	2.70	2.57	3.06	2.56	2.18	2.10	2.01	2.40	2.04
1110^5	9.46	7.21	4.21	3.82	3.36	3.59	3.46	2.89	2.71	2.53	2.55	2.27	2.19	2.06	2.03	2.34	2.08

Table B.6: Estimated communication times using query Q_3 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	714.52	718.51	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32	727.32
220	364.42	362.98	377.70	389.78	389.78	389.78	389.78	389.78	389.78	389.78	389.78	389.78	389.78	389.78	389.78	389.78	389.78
330	247.72	246.93	254.57	264.09	276.36	277.26	277.26	277.26	277.26	277.26	277.26	277.26	277.26	277.26	277.26	277.26	277.26
440	189.37	187.07	190.95	198.09	207.29	216.49	221.01	221.01	221.01	221.01	221.01	221.01	221.01	221.01	221.01	221.01	221.01
550	154.36	152.62	154.42	160.30	165.85	173.21	180.57	187.93	187.25	187.25	187.25	187.25	187.25	187.25	187.25	187.25	187.25
660	131.02	128.43	130.07	133.60	139.87	144.36	150.49	156.62	162.76	164.75	164.75	164.75	164.75	164.75	164.75	164.75	164.75
770	114.35	112.20	112.67	115.82	119.91	125.28	129.01	134.26	139.52	144.78	150.03	148.68	148.68	148.68	148.68	148.68	148.68
880	101.85	99.11	98.60	101.36	104.93	109.63	112.89	117.49	122.09	126.69	131.29	135.89	136.62	136.62	136.62	136.62	136.62
990	92.13	89.74	88.57	91.11	94.38	97.46	101.64	104.45	108.53	112.62	116.71	120.80	124.89	128.98	127.24	127.24	127.24
1100	84.35	81.51	80.54	82.01	84.95	87.72	91.49	95.25	97.69	101.37	105.05	108.73	112.41	116.09	119.77	119.74	119.74
2200	49.69	46.33	43.19	43.31	44.50	45.52	46.95	48.91	50.22	52.14	53.32	55.20	57.08	58.96	60.85	61.77	63.61
3300	38.14	34.36	30.74	30.41	30.69	31.45	32.49	33.47	34.39	35.26	36.57	37.36	38.64	39.92	41.20	41.85	43.10
4400	32.37	28.56	24.72	23.97	24.03	24.41	24.97	25.74	26.48	27.17	27.82	28.83	29.42	30.40	31.38	31.89	32.85
5500	28.76	24.93	20.95	20.10	20.03	20.19	20.69	21.11	21.73	22.32	22.88	23.40	24.22	24.69	25.49	25.91	26.70
6600	26.48	22.63	18.43	17.52	17.20	17.38	17.64	18.02	18.35	18.85	19.33	19.78	20.47	20.88	21.56	21.93	22.60
7700	24.84	20.89	16.63	15.55	15.32	15.37	15.46	15.81	16.12	16.57	17.01	17.41	17.80	18.16	18.76	19.08	19.67
8800	23.62	19.67	15.38	14.18	13.91	13.86	13.98	14.16	14.45	14.69	15.08	15.45	15.79	16.12	16.65	16.95	17.47
9900	22.59	18.65	14.32	13.12	12.71	12.69	12.69	12.87	13.15	13.38	13.75	13.92	14.23	14.73	15.02	15.29	15.76
$11x10^3$	21.84	17.90	13.48	12.27	11.84	11.75	11.78	11.84	12.11	12.33	12.53	12.85	13.15	13.44	13.71	13.96	14.39
$22x10^3$	18.38	14.35	9.79	8.36	7.75	7.48	7.33	7.27	7.29	7.34	7.43	7.50	7.62	7.73	7.81	7.98	8.14
$33x10^3$	17.24	13.17	8.53	7.08	6.39	6.06	5.84	5.75	5.69	5.68	5.68	5.71	5.78	5.82	5.91	5.99	6.06
$44x10^3$	16.66	12.59	7.93	6.43	5.73	5.35	5.10	4.96	4.89	4.85	4.85	4.82	4.85	4.87	4.92	4.95	5.02
$55x10^3$	16.32	12.24	7.55	6.03	5.32	4.92	4.66	4.51	4.41	4.35	4.32	4.29	4.30	4.30	4.32	4.36	4.39
$66x10^3$	16.11	12.01	7.31	5.79	5.05	4.64	4.38	4.21	4.09	4.02	3.99	3.96	3.93	3.92	3.95	3.97	3.98
$77x10^3$	15.95	11.85	7.13	5.60	4.85	4.44	4.17	3.99	3.86	3.78	3.73	3.70	3.67	3.67	3.66	3.66	3.68
$88x10^3$	15.84	11.73	7.01	5.46	4.72	4.29	4.01	3.82	3.70	3.60	3.56	3.51	3.47	3.47	3.44	3.46	3.46
$99x10^3$	15.76	11.64	6.91	5.36	4.60	4.17	3.88	3.70	3.57	3.48	3.41	3.36	3.32	3.31	3.30	3.30	3.28
$11x10^4$	15.68	11.57	6.83	5.27	4.51	4.07	3.78	3.60	3.46	3.37	3.30	3.24	3.20	3.18	3.16	3.15	3.14
$22x10^4$	15.45	11.29	6.50	4.91	4.13	3.66	3.36	3.15	2.99	2.88	2.79	2.72	2.66	2.62	2.59	2.56	2.53
$33x10^4$	15.45	11.26	6.41	4.81	4.01	3.54	3.23	3.00	2.84	2.72	2.63	2.55	2.49	2.43	2.39	2.36	2.33
$44x10^4$	15.51	11.28	6.39	4.77	3.96	3.48	3.16	2.94	2.77	2.65	2.55	2.47	2.40	2.35	2.30	2.26	2.23
$55x10^4$	15.59	11.32	6.39	4.76	3.94	3.45	3.13	2.90	2.74	2.60	2.50	2.42	2.35	2.30	2.25	2.21	2.18
$66x10^4$	15.68	11.38	6.41	4.76	3.93	3.44	3.12	2.89	2.71	2.58	2.48	2.39	2.32	2.26	2.22	2.18	2.14
$77x10^4$	15.83	11.48	6.45	4.78	3.95	3.45	3.12	2.88	2.71	2.57	2.47	2.38	2.31	2.25	2.20	2.16	2.12
$88x10^4$	15.92	11.54	6.48	4.79	3.95	3.45	3.11	2.88	2.70	2.57	2.46	2.37	2.30	2.24	2.18	2.14	2.10
$99x10^4$	16.02	11.60	6.50	4.80	3.96	3.45	3.11	2.87	2.70	2.56	2.45	2.36	2.29	2.22	2.17	2.13	2.09
$11x10^5$	16.09	11.65	6.52	4.81	3.96	3.45	3.11	2.87	2.69	2.55	2.44	2.35	2.28	2.21	2.16	2.11	2.08

Table B.7: Real communication times using query Q_4 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	20.15	20.72	24.48	27.61	30.12	32.35	33.78	35.09	37.15	39.71	40.86	44.78	46.13	45.93	51.93	52.60	53.15
220	11.22	11.30	12.26	13.71	14.97	15.95	16.70	16.99	17.66	19.36	19.19	22.28	22.58	22.52	25.61	25.27	25.97
330	8.74	8.57	8.81	9.65	10.29	11.10	11.19	11.25	11.65	12.79	13.23	14.80	15.20	15.37	16.73	17.31	17.29
440	7.23	6.81	6.91	7.53	7.91	8.58	8.85	8.68	8.89	9.67	10.35	11.18	11.57	11.65	12.74	12.91	13.35
550	6.56	5.80	5.89	6.28	6.84	7.30	7.54	7.26	7.30	8.25	8.58	9.15	9.58	9.76	10.37	10.56	10.85
660	5.86	5.32	5.46	5.77	6.18	6.55	6.75	6.54	6.57	7.16	7.40	8.01	8.15	8.43	9.06	9.26	9.51
770	5.25	4.82	4.95	5.43	5.58	6.03	6.08	6.03	6.15	6.61	6.93	7.39	7.54	7.88	8.34	8.60	8.79
880	4.96	4.57	4.47	4.90	5.01	5.36	5.49	5.30	5.41	5.82	6.15	6.44	6.64	6.97	7.28	7.54	7.68
990	4.62	4.24	4.11	4.50	4.61	4.93	4.97	4.89	4.90	5.25	5.54	5.73	5.98	6.19	6.53	6.69	6.91
1 100	4.37	4.04	3.85	4.21	4.26	4.52	4.63	4.46	4.51	4.87	5.10	5.33	5.58	5.66	5.94	6.12	6.41
2 200	3.47	2.84	2.52	2.71	2.69	2.74	2.76	2.63	2.62	2.77	2.85	2.94	3.05	3.13	3.26	3.35	3.44
3 300	3.38	2.47	1.95	2.17	2.16	2.19	2.15	1.99	1.98	2.07	2.13	2.15	2.24	2.30	2.37	2.43	2.50
4 400	2.95	2.24	1.72	1.95	1.88	1.89	1.82	1.69	1.67	1.71	1.75	1.76	1.84	1.87	1.93	1.96	1.99
5 500	3.22	2.13	1.56	1.80	1.71	1.70	1.63	1.51	1.47	1.51	1.54	1.55	1.58	1.60	1.67	1.68	1.73
6 600	3.14	2.04	1.48	1.68	1.59	1.59	1.51	1.37	1.35	1.38	1.39	1.37	1.42	1.44	1.49	1.50	1.59
7 700	2.75	2.12	1.46	1.62	1.50	1.52	1.41	1.28	1.25	1.27	1.28	1.26	1.32	1.33	1.36	1.37	1.39
8 800	2.71	2.17	1.41	1.58	1.46	1.47	1.35	1.22	1.18	1.19	1.19	1.19	1.22	1.22	1.27	1.26	1.27
9 900	2.56	2.17	1.37	1.49	1.39	1.38	1.28	1.16	1.12	1.13	1.12	1.13	1.15	1.16	1.20	1.17	1.19
$11x10^3$	2.52	2.09	1.39	1.46	1.35	1.34	1.26	1.12	1.07	1.09	1.07	1.06	1.08	1.09	1.14	1.13	1.15
$22x10^3$	2.54	2.01	1.43	1.34	1.26	1.26	1.14	1.01	0.92	0.94	0.93	0.90	0.88	0.88	0.93	0.87	0.90
$33x10^3$	2.73	2.17	1.45	1.31	1.28	1.29	1.19	1.05	0.96	0.94	0.92	0.90	0.89	0.83	0.89	0.83	0.86
$44x10^3$	2.59	2.15	1.43	1.34	1.29	1.30	1.20	1.08	0.99	0.95	0.95	0.87	0.85	0.83	0.87	0.81	0.82
$55x10^3$	2.53	2.08	1.49	1.31	1.27	1.28	1.21	1.08	1.00	0.95	0.92	0.88	0.85	0.82	0.85	0.80	0.82
$66x10^3$	2.49	2.06	1.45	1.32	1.29	1.29	1.23	1.05	1.02	0.97	0.93	0.87	0.86	0.83	0.85	0.79	0.81
$77x10^3$	2.50	2.03	1.44	1.33	1.29	1.31	1.23	1.08	1.01	1.00	0.94	0.87	0.85	0.82	0.84	0.80	0.82
$88x10^3$	2.46	2.00	1.46	1.32	1.32	1.30	1.25	1.08	1.01	0.97	0.95	0.87	0.83	0.81	0.82	0.79	0.81
$99x10^3$	2.44	2.02	1.46	1.30	1.32	1.28	1.21	1.08	1.02	0.97	0.95	0.87	0.84	0.81	0.82	0.79	0.79
$11x10^4$	2.40	1.98	1.47	1.33	1.33	1.30	1.24	1.07	1.02	0.98	0.95	0.87	0.84	0.80	0.82	0.79	0.78
$22x10^4$	2.37	1.96	1.47	1.30	1.31	1.28	1.24	1.08	1.03	0.98	0.92	0.86	0.82	0.79	0.81	0.78	0.76

Table B.8: Estimated communication times using query Q_4 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	24.76	24.48	25.61	27.48	29.52	31.64	33.79	35.97	38.16	40.36	42.56	44.77	46.98	49.19	51.41	53.63	55.84
220	15.45	14.84	14.86	15.61	16.54	17.55	18.59	19.65	20.72	21.81	22.90	23.99	25.09	26.19	27.29	28.39	29.50
330	12.35	11.63	11.28	11.66	12.22	12.85	13.52	14.21	14.91	15.62	16.34	17.07	17.79	18.52	19.25	19.98	20.72
440	10.80	10.03	9.49	9.68	10.05	10.50	10.98	11.49	12.01	12.53	13.06	13.60	14.14	14.69	15.23	15.78	16.32
550	9.87	9.06	8.41	8.49	8.75	9.09	9.46	9.86	10.26	10.68	11.10	11.53	11.95	12.39	12.82	13.25	13.69
660	9.25	8.42	7.69	7.70	7.89	8.15	8.45	8.77	9.10	9.44	9.79	10.14	10.49	10.85	11.21	11.57	11.93
770	8.81	7.96	7.18	7.14	7.27	7.48	7.72	7.99	8.27	8.56	8.85	9.15	9.45	9.76	10.06	10.37	10.68
880	8.48	7.62	6.80	6.71	6.81	6.98	7.18	7.41	7.65	7.90	8.15	8.41	8.67	8.93	9.20	9.47	9.74
990	8.22	7.35	6.50	6.38	6.45	6.58	6.76	6.95	7.16	7.38	7.60	7.83	8.06	8.30	8.53	8.77	9.01
1 100	8.01	7.13	6.26	6.12	6.16	6.27	6.42	6.59	6.78	6.97	7.17	7.37	7.58	7.78	8.00	8.21	8.42
2 200	7.08	6.17	5.19	4.93	4.86	4.86	4.90	4.96	5.03	5.11	5.20	5.29	5.39	5.48	5.58	5.68	5.79
3 300	6.77	5.85	4.83	4.54	4.43	4.39	4.39	4.42	4.45	4.49	4.54	4.60	4.66	4.72	4.78	4.84	4.91
4 400	6.62	5.69	4.65	4.34	4.21	4.16	4.14	4.14	4.16	4.19	4.22	4.25	4.29	4.33	4.38	4.42	4.47
5 500	6.53	5.59	4.54	4.22	4.08	4.02	3.99	3.98	3.99	4.00	4.02	4.05	4.07	4.10	4.14	4.17	4.21
6 600	6.46	5.53	4.47	4.14	3.99	3.92	3.89	3.87	3.87	3.88	3.89	3.91	3.93	3.95	3.97	4.00	4.03
7 700	6.42	5.48	4.42	4.08	3.93	3.86	3.81	3.79	3.79	3.79	3.80	3.81	3.82	3.84	3.86	3.88	3.91
8 800	6.39	5.45	4.38	4.04	3.89	3.80	3.76	3.74	3.72	3.72	3.73	3.73	3.74	3.76	3.77	3.79	3.81
9 900	6.36	5.42	4.35	4.01	3.85	3.77	3.72	3.69	3.68	3.67	3.67	3.68	3.68	3.69	3.71	3.72	3.74
$11x10^3$	6.34	5.40	4.33	3.98	3.82	3.73	3.68	3.65	3.64	3.63	3.63	3.63	3.63	3.64	3.65	3.67	3.68
$22x10^3$	6.25	5.30	4.22	3.86	3.69	3.59	3.53	3.49	3.46	3.44	3.43	3.42	3.42	3.41	3.41	3.41	3.42
$33x10^3$	6.22	5.27	4.18	3.82	3.65	3.55	3.48	3.44	3.40	3.38	3.36	3.35	3.34	3.34	3.33	3.33	3.33
$44x10^3$	6.20	5.26	4.16	3.80	3.63	3.52	3.46	3.41	3.38	3.35	3.33	3.32	3.31	3.30	3.29	3.29	3.28
$55x10^3$	6.19	5.24	4.15	3.79	3.61	3.51	3.44	3.39	3.36	3.33	3.31	3.30	3.28	3.27	3.27	3.26	3.26
$66x10^3$	6.18	5.24	4.15	3.78	3.61	3.50	3.43	3.38	3.35	3.32	3.30	3.28	3.27	3.26	3.25	3.24	3.24
$77x10^3$	6.18	5.23	4.14	3.78	3.60	3.49	3.42	3.37	3.34	3.31	3.29	3.27	3.26	3.25	3.24	3.23	3.23
$88x10^3$	6.18	5.23	4.14	3.77	3.59	3.49	3.42	3.37	3.33	3.30	3.28	3.27	3.25	3.24	3.23	3.22	3.22
$99x10^3$	6.17	5.23	4.13	3.77	3.59	3.48	3.41	3.36	3.33	3.30	3.28	3.26	3.25	3.23	3.22	3.22	3.21
$11x10^4$	6.17	5.23	4.13	3.77	3.59	3.48	3.41	3.36	3.32	3.30	3.27	3.26	3.24	3.23	3.22	3.21	3.20
$22x10^4$	6.16	5.22	4.12	3.76	3.58	3.47	3.40	3.34	3.31	3.28	3.25	3.24	3.22	3.21	3.20	3.19	3.18

Table B.9: Real communication times using query Q_5 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	9.587	9.054	11.016	12.818	12.352	12.763	13.296	12.889	12.201	10.701	10.376	11.129	11.025	10.846	10.516	10.455	11.529
220	5.281	5.189	5.844	7.031	7.361	7.646	7.968	7.678	7.565	6.562	6.442	7.048	6.782	6.771	6.237	6.466	6.728
330	3.226	3.773	4.348	4.635	4.835	5.479	5.764	5.548	5.880	5.087	4.921	5.310	4.977	5.218	4.880	4.583	4.908
440	2.639	2.964	3.390	3.720	3.749	4.062	4.510	4.772	4.773	4.416	4.145	4.505	4.234	4.528	4.115	4.043	4.334
550	2.280	2.463	2.967	3.135	3.071	3.379	3.699	3.803	4.234	3.845	3.796	3.991	3.795	4.014	3.735	3.712	3.741
660	1.977	2.149	2.669	2.630	2.641	2.781	3.166	3.261	3.527	3.318	3.418	3.655	3.511	3.672	3.470	3.368	3.409
770	1.814	1.916	2.161	2.293	2.245	2.479	2.685	2.917	3.124	2.943	2.969	3.458	3.277	3.375	3.202	3.139	3.186
880	1.702	1.816	1.953	2.091	2.045	2.257	2.441	2.459	2.856	2.603	2.668	2.895	2.871	3.146	2.937	2.911	2.974
990	1.549	1.684	1.777	1.878	1.891	1.960	2.108	2.233	2.568	2.410	2.453	2.646	2.657	2.795	2.745	2.812	2.848
1100	1.437	1.558	1.749	1.670	1.713	1.804	1.983	2.442	2.172	2.219	2.212	2.382	2.420	2.516	2.504	2.480	2.697
2200	0.906	0.896	0.989	0.931	0.887	0.945	1.061	1.058	1.181	1.147	1.171	1.269	1.258	1.310	1.308	1.306	1.372
3300	0.703	0.701	0.749	0.672	0.625	0.667	0.742	0.729	0.828	0.807	0.814	0.881	0.865	0.901	0.900	0.899	0.921
4400	0.642	0.555	0.533	0.504	0.503	0.538	0.585	0.597	0.640	0.620	0.637	0.676	0.663	0.692	0.692	0.681	0.700
5500	0.600	0.514	0.480	0.465	0.421	0.456	0.487	0.497	0.524	0.518	0.515	0.561	0.552	0.572	0.576	0.567	0.584
6600	0.543	0.484	0.423	0.434	0.370	0.391	0.420	0.407	0.444	0.446	0.451	0.482	0.477	0.495	0.493	0.477	0.504
7700	0.524	0.454	0.379	0.369	0.327	0.343	0.379	0.363	0.402	0.408	0.385	0.428	0.423	0.435	0.424	0.413	0.420
8800	0.487	0.400	0.347	0.348	0.305	0.300	0.336	0.338	0.360	0.351	0.351	0.382	0.372	0.381	0.380	0.368	0.378
9900	0.492	0.402	0.368	0.313	0.285	0.298	0.312	0.314	0.325	0.319	0.317	0.346	0.331	0.352	0.346	0.338	0.342
$11x10^3$	0.456	0.391	0.330	0.297	0.265	0.285	0.286	0.280	0.296	0.300	0.298	0.321	0.309	0.322	0.323	0.309	0.318
$22x10^3$	0.388	0.337	0.235	0.260	0.187	0.195	0.193	0.187	0.194	0.189	0.188	0.195	0.188	0.208	0.186	0.183	0.174
$33x10^3$	0.388	0.294	0.203	0.177	0.160	0.160	0.162	0.168	0.164	0.151	0.146	0.152	0.144	0.147	0.147	0.147	0.139
$44x10^3$	0.348	0.300	0.185	0.170	0.143	0.147	0.147	0.137	0.142	0.130	0.126	0.128	0.125	0.128	0.121	0.119	0.117
$55x10^3$	0.360	0.303	0.196	0.157	0.140	0.139	0.138	0.126	0.128	0.117	0.115	0.119	0.113	0.113	0.110	0.111	0.102
$66x10^3$	0.351	0.283	0.164	0.139	0.128	0.129	0.131	0.126	0.118	0.110	0.107	0.108	0.105	0.102	0.097	0.091	0.094
$77x10^3$	0.361	0.259	0.173	0.149	0.122	0.133	0.135	0.117	0.108	0.104	0.101	0.102	0.095	0.099	0.091	0.090	0.088
$88x10^3$	0.348	0.263	0.174	0.152	0.121	0.116	0.113	0.108	0.105	0.096	0.097	0.096	0.095	0.093	0.086	0.083	0.081
$99x10^3$	0.338	0.252	0.163	0.139	0.121	0.121	0.120	0.103	0.102	0.092	0.089	0.090	0.087	0.090	0.086	0.080	0.085
$11x10^4$	0.314	0.270	0.150	0.135	0.113	0.116	0.114	0.104	0.102	0.094	0.090	0.092	0.082	0.087	0.078	0.072	0.070
$22x10^4$	0.239	0.198	0.146	0.117	0.106	0.100	0.102	0.091	0.090	0.076	0.074	0.077	0.072	0.074	0.069	0.065	0.061
$33x10^4$	0.247	0.207	0.140	0.126	0.101	0.099	0.103	0.089	0.084	0.074	0.071	0.072	0.067	0.066	0.059	0.055	0.050
$44x10^4$	0.253	0.207	0.152	0.133	0.108	0.106	0.105	0.089	0.086	0.078	0.074	0.076	0.070	0.072	0.062	0.059	0.051
$55x10^4$	0.240	0.200	0.144	0.120	0.111	0.106	0.101	0.096	0.091	0.084	0.080	0.079	0.073	0.073	0.067	0.060	0.051
$66x10^4$	0.219	0.182	0.147	0.144	0.107	0.110	0.109	0.098	0.092	0.082	0.079	0.080	0.072	0.073	0.065	0.065	0.053
$77x10^4$	0.218	0.191	0.139	0.119	0.103	0.107	0.107	0.107	0.093	0.078	0.080	0.078	0.073	0.076	0.069	0.065	0.052
$88x10^4$	0.209	0.181	0.141	0.114	0.105	0.104	0.099	0.090	0.092	0.081	0.079	0.076	0.074	0.072	0.062	0.065	0.052
$99x10^4$	0.212	0.181	0.144	0.114	0.107	0.102	0.102	0.093	0.093	0.083	0.080	0.076	0.071	0.077	0.063	0.061	0.052
$11x10^5$	0.218	0.173	0.140	0.111	0.104	0.101	0.103	0.095	0.087	0.080	0.075	0.078	0.073	0.069	0.060	0.061	0.052

Table B.10: Estimated communication times using query Q_5 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	17.376	17.473	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782	17.782
220	8.862	8.917	9.185	9.633	9.573	9.573	9.573	9.573	9.573	9.573	9.573	9.573	9.573	9.573	9.573	9.573	9.573
330	6.025	6.005	6.191	6.423	6.721	6.837	6.837	6.837	6.837	6.837	6.837	6.837	6.837	6.837	6.837	6.837	6.837
440	4.606	4.594	4.694	4.873	5.041	5.265	5.489	5.469	5.469	5.469	5.469	5.469	5.469	5.469	5.469	5.469	5.469
550	3.789	3.712	3.756	3.899	4.034	4.213	4.392	4.571	4.648	4.648	4.648	4.648	4.648	4.648	4.648	4.648	4.648
660	3.216	3.154	3.164	3.286	3.402	3.511	3.660	3.809	3.959	4.108	4.101	4.101	4.101	4.101	4.101	4.101	4.101
770	2.806	2.729	2.740	2.817	2.916	3.047	3.137	3.265	3.393	3.521	3.649	3.710	3.710	3.710	3.710	3.710	3.710
880	2.499	2.433	2.423	2.493	2.582	2.667	2.781	2.858	2.970	3.081	3.193	3.305	3.417	3.417	3.417	3.417	3.417
990	2.279	2.203	2.176	2.216	2.295	2.370	2.472	2.574	2.640	2.739	2.839	2.938	3.037	3.137	3.236	3.189	3.189
1100	2.086	2.001	1.959	2.017	2.066	2.160	2.225	2.317	2.376	2.466	2.555	2.645	2.734	2.824	2.913	3.003	3.006
2200	1.235	1.145	1.061	1.065	1.082	1.120	1.156	1.190	1.221	1.268	1.315	1.343	1.388	1.434	1.480	1.526	1.547
3300	0.951	0.854	0.761	0.747	0.755	0.774	0.790	0.814	0.837	0.869	0.890	0.921	0.940	0.971	1.002	1.033	1.048
4400	0.813	0.713	0.611	0.589	0.591	0.600	0.614	0.626	0.644	0.661	0.686	0.701	0.726	0.740	0.763	0.787	0.799
5500	0.728	0.625	0.518	0.494	0.492	0.497	0.504	0.520	0.529	0.543	0.557	0.577	0.589	0.601	0.620	0.640	0.650
6600	0.670	0.569	0.458	0.434	0.427	0.427	0.434	0.443	0.452	0.464	0.476	0.488	0.498	0.515	0.525	0.541	0.550
7700	0.633	0.530	0.417	0.388	0.380	0.378	0.385	0.390	0.397	0.409	0.414	0.424	0.440	0.449	0.457	0.472	0.479
8800	0.601	0.497	0.384	0.353	0.345	0.341	0.344	0.348	0.356	0.362	0.371	0.381	0.389	0.397	0.405	0.418	0.425
9900	0.577	0.474	0.360	0.327	0.318	0.315	0.316	0.320	0.324	0.330	0.339	0.343	0.351	0.359	0.371	0.378	0.384
$11x10^3$	0.559	0.454	0.338	0.305	0.293	0.291	0.290	0.294	0.298	0.304	0.309	0.317	0.320	0.327	0.338	0.344	0.350
$22x10^3$	0.476	0.369	0.249	0.212	0.195	0.188	0.183	0.182	0.181	0.183	0.185	0.187	0.188	0.191	0.195	0.197	0.201
$33x10^3$	0.448	0.341	0.219	0.180	0.162	0.153	0.147	0.145	0.143	0.142	0.142	0.143	0.144	0.145	0.147	0.149	0.151
$44x10^3$	0.436	0.329	0.205	0.166	0.147	0.137	0.130	0.127	0.124	0.123	0.123	0.122	0.122	0.123	0.124	0.125	0.126
$55x10^3$	0.424	0.317	0.195	0.155	0.136	0.125	0.119	0.115	0.112	0.110	0.109	0.108	0.108	0.108	0.109	0.109	0.110
$66x10^3$	0.420	0.313	0.189	0.149	0.130	0.119	0.112	0.107	0.104	0.102	0.101	0.100	0.100	0.099	0.100	0.100	0.100
$77x10^3$	0.422	0.313	0.188	0.147	0.127	0.116	0.108	0.103	0.100	0.098	0.097	0.096	0.094	0.094	0.094	0.094	0.095
$88x10^3$	0.416	0.308	0.183	0.142	0.122	0.111	0.103	0.099	0.095	0.093	0.091	0.090	0.089	0.088	0.089	0.088	0.088
$99x10^3$	0.415	0.306	0.181	0.140	0.120	0.108	0.101	0.096	0.092	0.090	0.088	0.087	0.086	0.085	0.085	0.084	0.084
$11x10^4$	0.414	0.305	0.180	0.138	0.118	0.106	0.099	0.094	0.090	0.087	0.085	0.084	0.083	0.082	0.082	0.081	0.081
$22x10^4$	0.420	0.307	0.176	0.133	0.112	0.099	0.091	0.085	0.081	0.077	0.075	0.073	0.071	0.070	0.069	0.068	0.068
$33x10^4$	0.432	0.314	0.179	0.134	0.112	0.098	0.089	0.083	0.079	0.075	0.073	0.070	0.069	0.067	0.066	0.065	0.064
$44x10^4$	0.445	0.323	0.183	0.136	0.113	0.099	0.090	0.084	0.079	0.075	0.073	0.070	0.068	0.067	0.065	0.064	0.063
$55x10^4$	0.447	0.325	0.183	0.136	0.113	0.099	0.090	0.083	0.078	0.074	0.071	0.069	0.067	0.065	0.064	0.063	0.062
$66x10^4$	0.473	0.343	0.193	0.143	0.118	0.103	0.094	0.087	0.081	0.077	0.074	0.072	0.070	0.068	0.066	0.065	0.064
$77x10^4$	0.476	0.345	0.194	0.143	0.118	0.103	0.093	0.086	0.081	0.077	0.074	0.071	0.069	0.067	0.066	0.064	0.063
$88x10^4$	0.457	0.331	0.185	0.137	0.113	0.099	0.089	0.082	0.077	0.073	0.070	0.067	0.065	0.064	0.062	0.061	0.060
$99x10^4$	0.517	0.375	0.210	0.155	0.127	0.111	0.100	0.092	0.087	0.082	0.079	0.076	0.073	0.071	0.070	0.068	0.067
$11x10^5$	0.463	0.335	0.187	0.138	0.114	0.099	0.089	0.082	0.077	0.073	0.070	0.067	0.065	0.063	0.062	0.060	0.059

Table B.11: Real communication times using query Q_6 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	0.35	0.33	0.36	0.36	0.36	0.33	0.34	0.38	0.39	0.38	0.35	0.33	0.34	0.32	0.37	0.35	0.34
220	0.34	0.35	0.33	0.35	0.34	0.36	0.35	0.35	0.32	0.34	0.33	0.32	0.32	0.32	0.34	0.34	0.30
330	0.34	0.34	0.32	0.32	0.33	0.32	0.36	0.35	0.32	0.36	0.33	0.32	0.35	0.36	0.33	0.32	0.34
440	0.35	0.34	0.33	0.37	0.35	0.33	0.33	0.36	0.34	0.40	0.35	0.32	0.32	0.33	0.36	0.32	0.32
550	0.37	0.32	0.33	0.35	0.34	0.34	0.33	0.33	0.37	0.35	0.36	0.30	0.34	0.32	0.33	0.32	0.33
660	0.36	0.32	0.37	0.37	0.34	0.34	0.33	0.34	0.35	0.35	0.35	0.32	0.34	0.32	0.33	0.33	0.30
770	0.34	0.34	0.34	0.34	0.34	0.34	0.33	0.33	0.36	0.36	0.35	0.33	0.32	0.35	0.32	0.32	0.33
880	0.37	0.33	0.34	0.34	0.35	0.35	0.38	0.36	0.33	0.38	0.33	0.31	0.34	0.32	0.32	0.32	0.32
990	0.36	0.35	0.41	0.35	0.36	0.32	0.36	0.38	0.34	0.35	0.35	0.33	0.33	0.32	0.34	0.34	0.31
1 100	0.35	0.35	0.35	0.35	0.33	0.34	0.33	0.35	0.35	0.37	0.33	0.32	0.32	0.32	0.33	0.32	0.32
2 200	0.33	0.34	0.34	0.36	0.34	0.34	0.35	0.35	0.33	0.34	0.33	0.31	0.32	0.32	0.33	0.32	0.35
3 300	0.37	0.34	0.35	0.36	0.34	0.32	0.34	0.33	0.33	0.36	0.34	0.34	0.35	0.34	0.32	0.32	0.33
4 400	0.34	0.37	0.36	0.37	0.35	0.32	0.33	0.36	0.36	0.36	0.33	0.32	0.32	0.32	0.33	0.33	0.32
5 500	0.38	0.34	0.34	0.33	0.34	0.37	0.34	0.37	0.34	0.36	0.36	0.33	0.32	0.33	0.33	0.32	0.32
6 600	0.35	0.33	0.36	0.33	0.34	0.33	0.37	0.36	0.35	0.37	0.33	0.32	0.32	0.31	0.34	0.35	0.32
7 700	0.39	0.35	0.33	0.36	0.34	0.32	0.35	0.34	0.33	0.35	0.34	0.32	0.32	0.33	0.35	0.32	0.33
8 800	0.37	0.36	0.33	0.33	0.35	0.32	0.35	0.36	0.34	0.36	0.34	0.33	0.33	0.31	0.34	0.33	0.33
9 900	0.37	0.36	0.36	0.36	0.33	0.33	0.35	0.37	0.34	0.37	0.34	0.31	0.33	0.32	0.35	0.31	0.35
$11x10^3$	0.35	0.35	0.35	0.39	0.34	0.33	0.35	0.35	0.33	0.37	0.33	0.32	0.30	0.31	0.33	0.36	0.32
$22x10^3$	0.36	0.35	0.34	0.33	0.38	0.35	0.37	0.34	0.36	0.36	0.32	0.32	0.32	0.32	0.34	0.31	0.32
$33x10^3$	0.36	0.34	0.34	0.33	0.35	0.32	0.34	0.34	0.38	0.32	0.33	0.33	0.33	0.31	0.33	0.33	0.34
$44x10^3$	0.36	0.34	0.33	0.32	0.35	0.34	0.34	0.35	0.33	0.36	0.34	0.33	0.32	0.32	0.32	0.32	0.34
$55x10^3$	0.38	0.36	0.33	0.35	0.34	0.35	0.34	0.34	0.33	0.36	0.34	0.34	0.32	0.33	0.32	0.34	0.32
$66x10^3$	0.35	0.36	0.35	0.34	0.34	0.33	0.34	0.35	0.32	0.35	0.33	0.31	0.33	0.31	0.33	0.32	0.32
$77x10^3$	0.37	0.35	0.33	0.36	0.33	0.32	0.37	0.34	0.38	0.37	0.33	0.31	0.33	0.32	0.35	0.33	0.32
$88x10^3$	0.34	0.35	0.34	0.35	0.34	0.34	0.37	0.34	0.37	0.37	0.33	0.31	0.33	0.33	0.33	0.34	0.32
$99x10^3$	0.40	0.33	0.34	0.34	0.36	0.36	0.35	0.35	0.33	0.33	0.34	0.31	0.33	0.33	0.34	0.32	0.33
$11x10^4$	0.36	0.37	0.34	0.35	0.35	0.33	0.34	0.33	0.33	0.36	0.33	0.33	0.32	0.34	0.33	0.32	0.32
$22x10^4$	0.36	0.35	0.33	0.33	0.32	0.36	0.34	0.33	0.36	0.34	0.36	0.32	0.31	0.32	0.33	0.32	0.34
$33x10^4$	0.35	0.40	0.34	0.34	0.34	0.35	0.35	0.33	0.34	0.35	0.34	0.33	0.33	0.33	0.32	0.33	0.33
$44x10^4$	0.39	0.36	0.35	0.34	0.36	0.33	0.35	0.33	0.33	0.35	0.33	0.34	0.32	0.33	0.32	0.34	0.33
$55x10^4$	0.35	0.35	0.34	0.35	0.37	0.36	0.36	0.36	0.34	0.38	0.34	0.31	0.32	0.33	0.32	0.33	0.33
$66x10^4$	0.34	0.36	0.33	0.33	0.38	0.35	0.34	0.36	0.36	0.34	0.35	0.33	0.33	0.32	0.34	0.35	0.32
$77x10^4$	0.36	0.34	0.34	0.36	0.33	0.33	0.38	0.36	0.35	0.35	0.36	0.33	0.33	0.33	0.33	0.31	0.35
$88x10^4$	0.34	0.35	0.38	0.33	0.38	0.34	0.36	0.33	0.35	0.36	0.36	0.32	0.33	0.33	0.34	0.33	0.34
$99x10^4$	0.35	0.35	0.36	0.35	0.35	0.34	0.36	0.37	0.36	0.34	0.36	0.32	0.32	0.32	0.34	0.32	0.31
$11x10^5$	0.38	0.35	0.38	0.34	0.35	0.34	0.34	0.35	0.34	0.35	0.34	0.32	0.32	0.34	0.34	0.34	0.33

Table B.12: Estimated communication times using query Q_6 in high-bandwidth network

F \ M	1.5Kb	2Kb	4Kb	6Kb	8Kb	10Kb	12Kb	14Kb	16Kb	18Kb	20Kb	22Kb	24Kb	26Kb	28Kb	30Kb	32Kb
110	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
220	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
330	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
440	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
550	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
660	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
770	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
880	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
990	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
1 100	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
2 200	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
3 300	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
4 400	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
5 500	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
6 600	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
7 700	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
8 800	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
9 900	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$11x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$22x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$33x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$44x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$55x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$66x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$77x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$88x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$99x10^3$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$11x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$22x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$33x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$44x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$55x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$66x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$77x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$88x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$99x10^4$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$
$11x10^5$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$	$5.2e^{-4}$

Appendix C

Iterations of MIND algorithm on different queries

To find the best configuration F and M that minimize the communication time for a given q query. The MIND optimization algorithm (Section 5.2) applied for each query gives the following iterations. In these iterations, we report for each iteration: the F, M, communication cost for current iteration (Com. time K), communication cost fro next iteration (Com. time K+1) and the difference between the communication times in current and next iterations ((K) - (K+1)). The input that should be introduced MIND optimization algorithm:

- The estimated size of the query result (V) and tuple size (T) are indicated in Table 3.3,
- weights α and β are calibrated on Q_3 (Section 4.3).

The iterations of MIND optimization algorithm in high bandwidth network ($10Gbps$) on Q_1 are presented in Table C.1.

The iterations of MIND optimization algorithm in low bandwidth network ($50Mbps$) on Q_1 are presented in Table C.1.

The iterations of MIND optimization algorithm in high bandwidth network ($10Gbps$) on Q_2 are presented in Table C.3.

The iterations of MIND optimization algorithm in low bandwidth network ($50Mbps$) on Q_2 are presented in Table C.4.

The iterations of MIND optimization algorithm in high bandwidth network ($10Gbps$) on Q_3 are presented in Table C.5.

The iterations of MIND optimization algorithm in low bandwidth network ($50Mbps$) on Q_3 are presented in Table C.6.

The iterations of MIND optimization algorithm in high bandwidth network ($10Gbps$) on Q_4 are presented in Table C.7.

The iterations of MIND optimization algorithm in low bandwidth network ($50Mbps$) on Q_4 are presented in Table C.8.

The iterations of MIND optimization algorithm in high bandwidth network ($10Gbps$) on Q_5 are presented in Table C.9.

The iterations of MIND optimization algorithm in low bandwidth network ($50Mbps$) on Q_5 are presented in Table C.9.

The iterations of MIND optimization algorithm in high bandwidth network ($10Gbps$) on Q_6 are presented in Table C.11.

Table C.1: Iterations of MIND optimization algorithm using Q_1 in high bandwidth network ($10Gbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	5	512	16 146.000		
1	7	372	16 146.000	10 838.000	5 307.900
2	11	458	10 838.000	7 302.000	3 535.700
3	17	563	7 302.000	4 934.200	2 367.800
4	25	691	4 934.200	3 346.900	1 587.300
5	38	848	3 346.900	2 281.600	1 065.300
6	58	1 042	2 281.600	1 565.600	716.050
7	87	1 279	1 565.600	1 083.400	482.130
8	131	1 572	1 083.400	758.110	325.310
9	198	1 933	758.110	538.060	220.050
10	299	2 379	538.060	388.770	149.290
11	453	2 930	388.770	287.120	101.640
12	688	3 613	287.120	217.630	69.488
13	1 047	4 461	217.630	169.900	47.734
14	1 598	5 517	169.900	136.930	32.969
15	2 447	6 834	136.930	114.020	22.914
16	3 761	8 483	114.020	97.981	16.037
17	5 802	10 554	97.981	86.672	11.309
18	8 992	13 162	86.672	78.631	8.041
19	14 001	16 458	78.631	72.864	5.767
20	21 912	20 635	72.864	68.691	4.173
21	34 471	25 946	68.691	65.645	3.046
22	54 509	32 712	65.645	63.401	2.244
23	86 631	32 767	63.401	61.781	1.620
24	110 000	32 767	61.781	60.725	1.056

The iterations of MIND optimization algorithm in low bandwidth network ($50Mbps$) on Q_6 are presented in Table C.11.

Table C.2: Iterations of MIND optimization algorithm using Q_1 in low bandwidth network ($50Mbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	5	512		22 576.7	
1	8	766	22 576.7	15 364	7 212.7
2	11	1 143	15 363.9	10 555	4 808.9
3	17	1 702	10 555.1	7 348.5	3 206.6
4	26	2 527	7 348.5	5 209.8	2 138.7
5	39	3 734	5 209.8	3 782.7	1 427.1
6	59	5 481	3 782.7	2 829.4	953.25
7	90	7 978	2 829.4	2 191.5	637.91
8	137	11 484	2 191.5	1 763.2	428.29
9	212	16 312	1 763.2	1 474.1	289.1
10	328	22 825	1 474.1	1 277.4	196.68
11	512	31 450	1 277.4	1 142.2	135.17
12	801	32 767	1 142.2	1 059.7	82.502
13	1 259	32 767	1 059.7	1 010.3	49.444
14	1 966	32 767	1 010.3	983.27	27.028
15	2 993	32 767	983.27	972.56	10.715
16	4 269	32 767	972.56	973.14	-0.588
17	5 262	32 767	973.14	977.58	-4.437
18	5 258	32 767	977.58	977.56	0.021
19	5 260	32 767	977.56	977.57	-0.011
20	5 259	32 767	977.57	977.57	0.005
21	5 260	32 767	977.57	977.57	-0.003
22	5 259	32 767	977.57	977.57	0.001
23	5 260	32 767	977.57	977.57	-0.001
24	5 259	32 767	977.57	977.57	0.000

Table C.3: Iterations of MIND optimization algorithm using Q_2 in high bandwidth network ($10Gbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	20	512			
1	30	393	4 039.400	2 712.400	1 327.000
2	45	477	2 712.400	1 828.200	884.190
3	68	583	1 828.200	1 235.900	592.240
4	102	713	1 235.900	838.850	397.090
5	153	873	838.850	572.280	266.570
6	230	1 071	572.280	393.070	179.210
7	347	1 315	393.070	272.360	120.700
8	524	1 615	272.360	190.900	81.467
9	792	1 986	190.900	135.770	55.127
10	1 198	2 444	135.770	98.354	37.416
11	1 815	3 010	98.354	72.867	25.487
12	2 757	3 712	72.867	55.432	17.434
13	4 198	4 585	55.432	43.448	11.984
14	6 409	5 671	43.448	35.164	8.284
15	9 817	7 027	35.164	29.402	5.762
16	15 093	8 724	29.402	25.366	4.036
17	23 300	10 857	25.366	22.516	2.849
18	36 130	13 545	22.516	20.488	2.029
19	56 296	16 942	20.488	19.031	1.457
20	88 162	21 250	19.031	17.977	1.055
21	138 780	26 728	17.977	17.205	0.771
22	219 600	32 767	17.205	16.638	0.568
23	349 240	32 767	16.638	16.215	0.423
24	550 000	32 767	16.215	15.898	0.317

Table C.4: Iterations MIND optimization algorithm using Q_2 in low bandwidth network ($50Mbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	20	512	5 885.9		
1	30	766	5 885.9	4 009.5	1876.4
2	45	1 144	4 009.5	2 758.5	1251
3	68	1 705	2 758.5	1 924.3	834.16
4	103	2 534	1 924.3	1 368	556.33
5	155	3 749	1 368	996.79	371.2
6	236	5 516	996.79	748.9	247.89
7	359	8 052	748.9	583.07	165.83
8	550	11 636	583.07	471.8	111.27
9	848	16 608	471.8	396.76	75.037
10	1 315	23 376	396.76	345.78	50.980
11	2 052	32 425	345.78	310.81	34.975
12	3 218	32 767	310.81	290.6	20.207
13	5 055	32 767	290.6	278.57	12.025
14	7 863	32 767	278.57	272.30	6.272
15	11 840	32 767	272.3	270.23	2.074
16	16 446	32 767	270.23	270.89	-0.665
17	19 255	32 767	270.89	271.96	-1.065
18	18 724	32 767	271.96	271.73	0.225
19	19 006	32 767	271.73	271.85	-0.118
20	18 869	32 767	271.85	271.79	0.058
21	18 939	32 767	271.79	271.82	-0.030
22	18 904	32 767	271.82	271.81	0.015
23	18 922	32 767	271.81	271.81	-0.008
24	18 913	32 767	271.81	271.81	0.004

Table C.5: Iterations of MIND optimization algorithm using Q_3 in high bandwidth network ($10Gbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	40	512	2 003.700		
1	57	682	2 003.700	1 353.000	650.740
2	86	889	1 353.000	905.340	447.660
3	129	1 135	905.340	612.200	293.140
4	193	1 428	612.200	410.300	201.890
5	291	1 779	410.300	278.010	132.290
6	438	2 203	278.010	188.640	89.370
7	660	2 719	188.640	128.230	60.413
8	995	3 349	128.230	88.232	39.998
9	1 501	4 122	88.232	60.280	27.952
10	2 269	5 074	60.280	42.158	18.122
11	3 433	6 249	42.158	29.318	12.840
12	5 205	7 702	29.318	20.905	8.413
13	7 907	9 502	20.905	14.997	5.908
14	12 043	11 740	14.997	10.941	4.057
15	18 394	14 528	10.941	8.196	2.745
16	28 187	18 012	8.196	6.245	1.951
17	43 355	22382	6.245	4.878	1.367
18	66 958	27 883	4.878	3.899	0.979
19	103 870	32 767	3.899	3.219	0.680
20	162 420	32 767	3.219	2.745	0.474
21	256 930	32 767	2.745	2.446	0.299
22	408 630	32 767	2.446	2.254	0.191
23	649 460	32 767	2.254	2.151	0.104
24	1 021 600	32 767	2.151	2.088	0.063

Table C.6: Iterations of MIND optimization algorithm using Q_3 in low bandwidth network ($50Mbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	40	512	3 068.2		
1	60	766	2 929	1 993.05	935.95
2	90	1 143	1 993.13	1 369.1	624.03
3	136	1 704	1 369.1	952.96	416.1
4	206	2 532	952.96	675.43	277.53
5	311	3 745	675.43	490.24	185.19
6	473	5 507	490.24	366.54	123.7
7	721	8 034	366.54	283.75	82.784
8	1 107	11 600	283.75	228.17	55.586
9	1 710	16 542	228.17	190.64	37.528
10	2 659	23 261	190.64	165.1	25.539
11	4 159	32 234	165.1	147.54	17.563
12	6 539	32 767	147.54	137.32	10.221
13	10 310	32 767	137.32	131.22	6.098
14	16 124	32 767	131.22	127.98	3.24
15	24 487	32 767	127.98	126.82	1.16
16	34 505	32 767	126.82	127.05	-0.235
17	41 324	32 767	127.05	127.61	-0.555
18	40 523	32 767	127.61	127.53	0.075
19	40 940	32 767	127.53	127.57	-0.039
20	40 735	32 767	127.57	127.55	0.019
21	40 839	32 767	127.55	127.56	-0.01
22	40 787	32 767	127.56	127.56	0.005
23	40 813	32 767	127.56	127.56	-0.002
24	40 800	32 767	127.56	127.56	0.001

Table C.7: Iterations of MIND optimization algorithm using Q_4 in high bandwidth network ($10Gbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	2	512	979.97		
1	3	315	979.97	657.46	322.51
2	4	404	657.46	442.51	214.95
3	7	508	442.51	298.67	143.85
4	10	634	298.67	202.3	96.369
5	15	785	202.3	137.66	64.641
6	23	969	137.66	94.235	43.422
7	35	1 193	94.235	65.018	29.217
8	52	1 469	65.018	45.319	19.698
9	79	1 807	45.319	32.006	13.313
10	119	2 224	32.006	22.984	9.023
11	181	2 739	22.984	16.848	6.136
12	274	3 377	16.848	12.659	4.188
13	417	4 168	12.659	9.787	2.873
14	636	5 152	9.787	7.806	1.981
15	972	6 379	7.806	6.432	1.373
16	1 492	7 913	6.432	5.473	0.959
17	2 299	9 837	5.474	4.799	0.675
18	3 558	12 259	4.799	4.320	0.479
19	5 531	15 315	4.320	3.978	0.342
20	8 642	19 186	3.978	3.732	0.247
21	13 572	24 101	3.732	3.552	0.180
22	21 426	30 360	3.552	3.420	0.131
23	33 998	32 767	3.420	3.323	0.097
24	54 879	32 767	3.323	3.258	0.065

Table C.8: Iterations of MIND optimization algorithm using Q_4 in low bandwidth network ($50Mbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	2	512	1 308.7		
1	3	765	1 308.7	889.56	419.17
2	5	1 141	889.56	610.08	279.48
3	7	1 698	610.08	423.72	186.36
4	10	2 518	423.72	299.42	124.3
5	16	3 712	299.42	216.46	82.958
6	24	5 434	216.46	161.03	55.425
7	36	7 879	161.03	123.93	37.107
8	55	11 283	123.93	98.993	24.933
9	85	15 925	98.993	82.143	16.850
10	131	22 118	82.143	70.661	11.482
11	204	30 224	70.661	62.755	7.907
12	319	32 767	62.755	57.672	5.083
13	500	32 767	57.672	54.584	3.087
14	784	32 767	54.584	52.821	1.763
15	1 207	32 767	52.821	52.014	0.807
16	1 766	32 767	52.014	51.895	0.119
17	2 300	32 767	51.895	52.126	-0.231
18	2 433	32 767	52.126	52.209	-0.083
19	2 376	32 767	52.170	52.192	-0.023
20	2 390	32 767	52.209	52.172	0.037
21	2 406	32 767	52.172	52.191	-0.019
22	2 391	32 767	52.191	52.182	0.009
23	2 399	32 767	52.182	52.187	-0.005
24	2 395	32 767	52.187	52.184	0.002

Table C.9: Iterations of MIND optimization algorithm using Q_5 in high bandwidth network ($10Gbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	40	512	52.338		
1	53	682	52.338	35.341	16.997
2	80	889	35.341	23.648	11.693
3	120	1 135	23.648	15.991	7.657
4	180	1 428	15.991	10.717	5.273
5	271	1 779	10.717	7.262	3.456
6	408	2 203	7.262	4.928	2.334
7	614	2 719	4.928	3.350	1.578
8	926	3 349	3.350	2.305	1.044
9	1 398	4 122	2.305	1.575	0.730
10	2 112	5 074	1.575	1.101	0.473
11	3 196	6 249	1.101	0.766	0.335
12	4 846	7 702	0.766	0.547	0.220
13	7 362	9 502	0.547	0.392	0.154
14	11 212	11 740	0.392	0.286	0.107
15	17 125	14 528	0.286	0.215	0.071
16	26 243	18 012	0.215	0.163	0.052
17	40 365	22 382	0.163	0.128	0.035
18	62 340	27 883	0.128	0.103	0.025
19	96 706	32 767	0.103	0.085	0.018
20	151 220	32 767	0.085	0.073	0.012
21	239 210	32 767	0.073	0.065	0.008
22	380 450	32 767	0.065	0.061	0.003
23	604 670	32 767	0.061	0.059	0.002
24	951 130	32 767	0.059	0.064	0.005

Table C.10: Iterations of MIND optimization algorithm using Q_5 in low bandwidth network ($50Mbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	40	512	72.079		
1	60	766	72.079	49.109	24.525
2	90	1 144	49.109	33.795	15.315
3	136	1 705	33.795	23.583	10.212
4	205	2 535	23.583	16.773	6.8104
5	311	3 752	16.773	12.229	4.544
6	471	5 522	12.229	9.194	3.0345
7	718	8 064	9.194	7.1642	2.0298
8	1 100	11 659	7.1642	5.8023	1.3619
9	1 696	16 655	5.8023	4.884	0.91826
10	2 631	23 464	4.884	4.2603	0.62371
11	4 106	32 582	4.2603	3.8325	0.42777
12	6 440	44 632	3.8325	3.5879	0.24465
13	10 115	46 857	3.5879	3.4424	0.14548
14	15 722	48 219	3.4424	3.3672	0.07521
15	23 626	49 127	3.3672	3.3433	0.02394
16	32 660	49 497	3.3433	3.3523	-0.00907
17	37 901	48 974	3.3523	3.365	-0.01265
18	36 768	48 108	3.365	3.362	0.00303
19	37 372	48 339	3.362	3.3636	-0.0016
20	37 078	48 219	3.3636	3.3628	0.00078
21	37 229	48 278	3.3628	3.3632	-0.0004
22	37 154	48 248	3.3632	3.363	0.0002
23	37 192	48 263	3.363	3.3631	-0.0001
24	37 172	48 256	3.3631	3.363	0.00005

Table C.11: Iterations of MIND optimization algorithm using Q_6 in high bandwidth network ($10Gbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	40	512	0.00097		
1	59.852	392.61	0.00097	0.00049	0.00048

Table C.12: Iterations of MIND optimization algorithm using Q_6 in low bandwidth network ($50Mbps$).

It.	F	M	$C(\mathsf{F}_k, \mathsf{M}_k)$	$C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$	$C(\mathsf{F}_k, \mathsf{M}_k) - C(\mathsf{F}_{k+1}, \mathsf{M}_{k+1})$
0	40	512	0.00097		
1	60.118	765.713	0.00097	0.00066	0.00031

Appendix D

Network overhead

Pictures provided in this appendix are produced from trace files generated by TCPDUMP [1] and analysed by wireshark [2]. These pictures are captured in client node (TCP layer).

[1] http://www.tcpdump.org/
[2] https://www.wireshark.org/

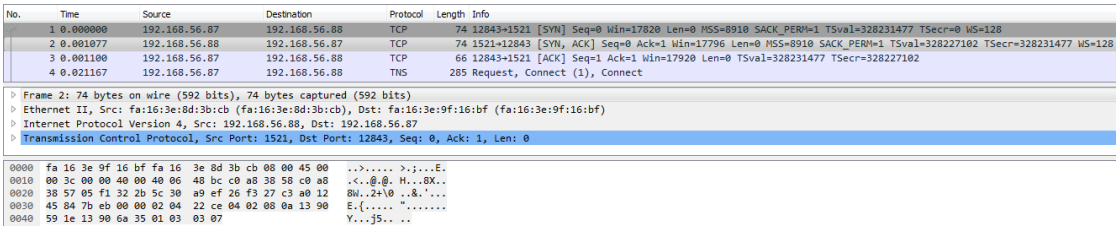
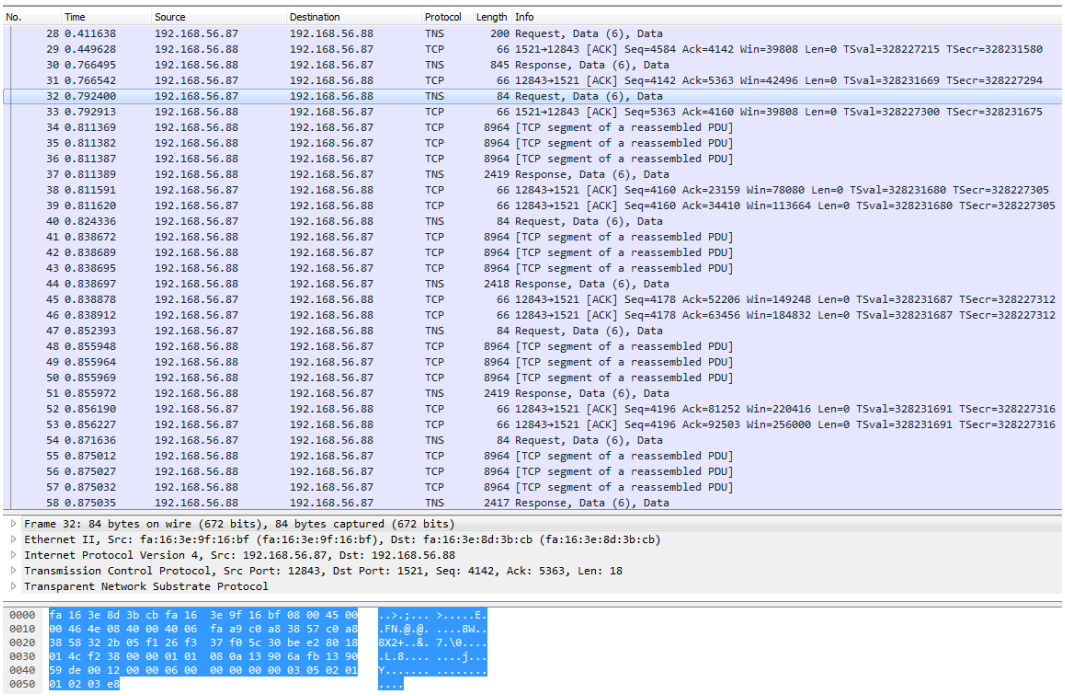
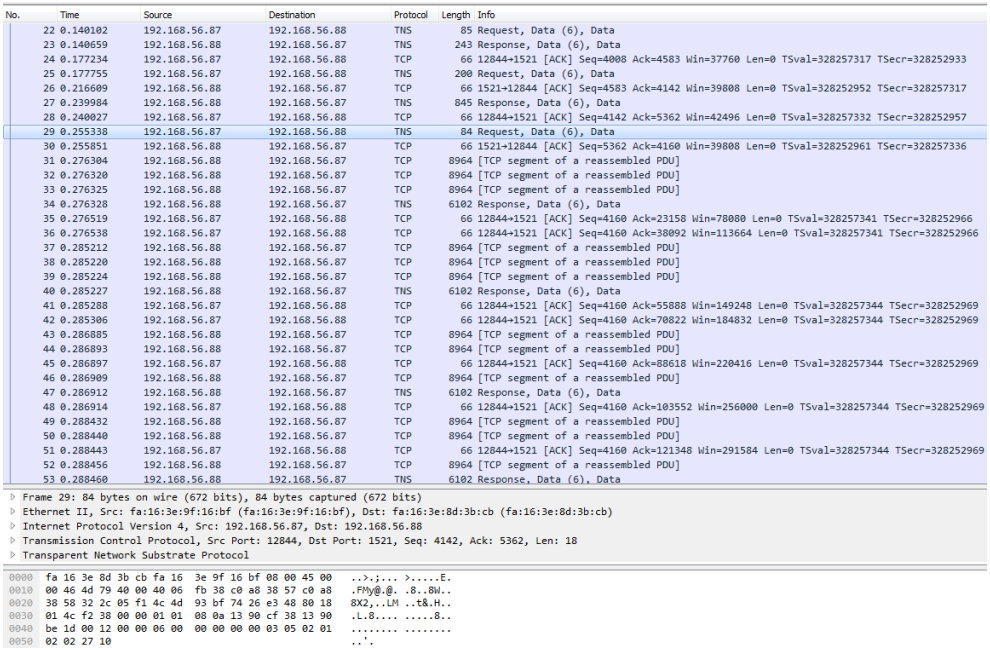


Figure D.1: Connexion of client node to DBMS node in TCP/IP layer.

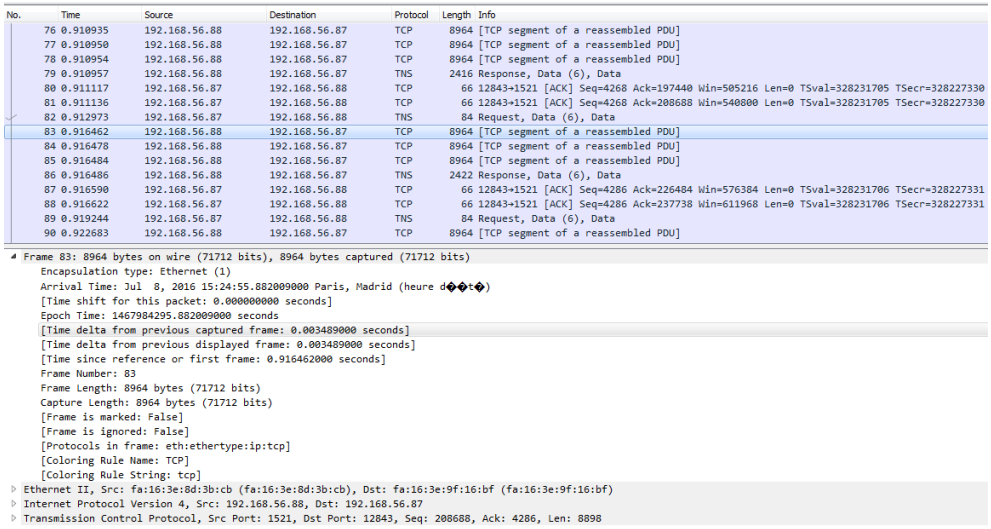


(a) Requested batch of F= 1.1K tuples (e.g., lines 32, 40, 47 and 54) is sent in one message of M=32KB (e.g., lines 30, 37, 44, 51 and 58) and each M is split into four network packets (e.g., lines 34, 35, 36 and 37 constitute one M). Each DBMS response of batch (e.g., line 30 in Figure) is succeeded by another request for batch of F= 1.1K tuples (e.g., line 32 in Figure).

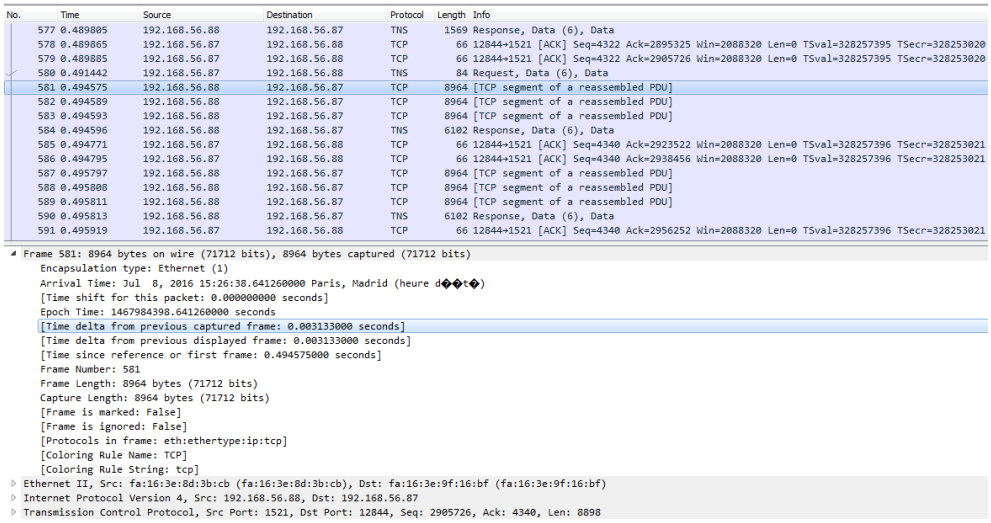


(b) Requested batch of F= 110K tuples (e.g., lines 29) is sent in several messages of M=32KB communicated in pipeline (e.g., lines 34, 40, 47, 53, ...) and each M is split into four network packets (e.g., lines 31, 32, 33 and 34 constitute one M).

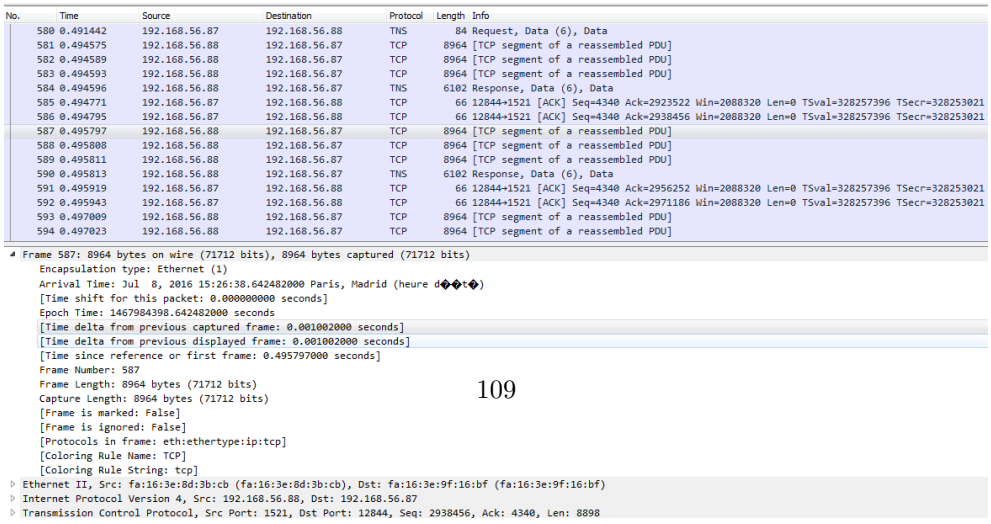
Figure D.2: Zoom on requested batches in TCP network layer.



(a) Communication time of first message in a batch of F= 1.1K tuples and M=32KB for query Q_3 in high-bandwidth network. Recall that a batch is communicated in one data message, which presents an expensive communication time.



(b) Communication time of first message in a batch of F= 110K tuples and M=32KB for query Q_3 in high-bandwidth network. The first message presents an expensive communication time.



(c) Cost of next messages in a batch of F= 110K tuples and M=32KB for query Q_3 in high-bandwidth network. The time for communicating not first messages in its batch is more cheaper than the first message in its batch.

Figure D.3: Cost of first and next messages in batches in TCP network layer.

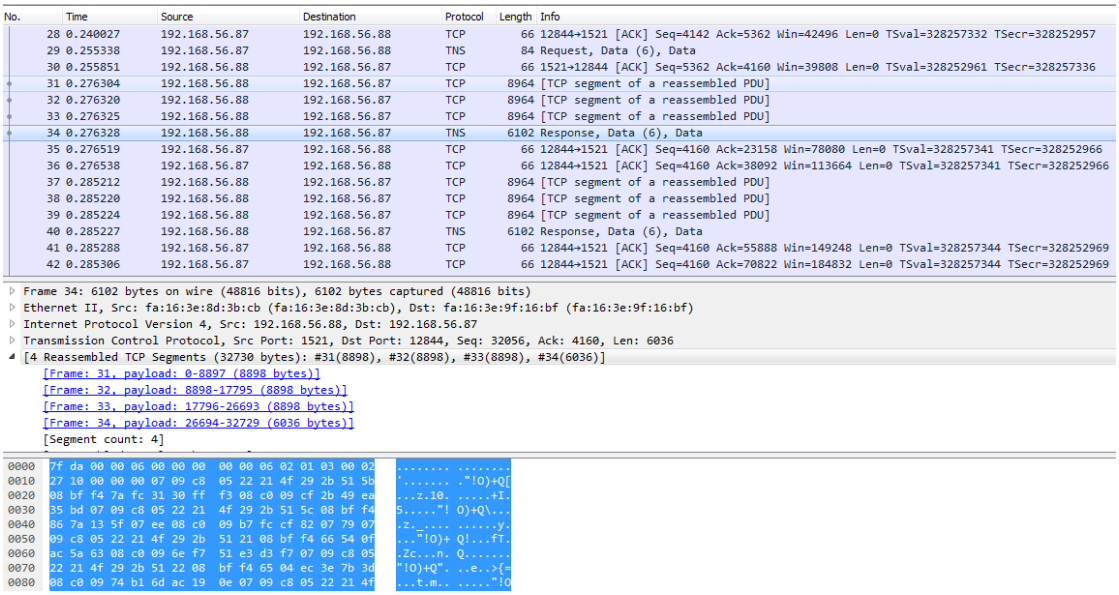


Figure D.4: Zoom on message of M=32KB fragmented in four network packets according to MTU (which is setted to 8.95KB).

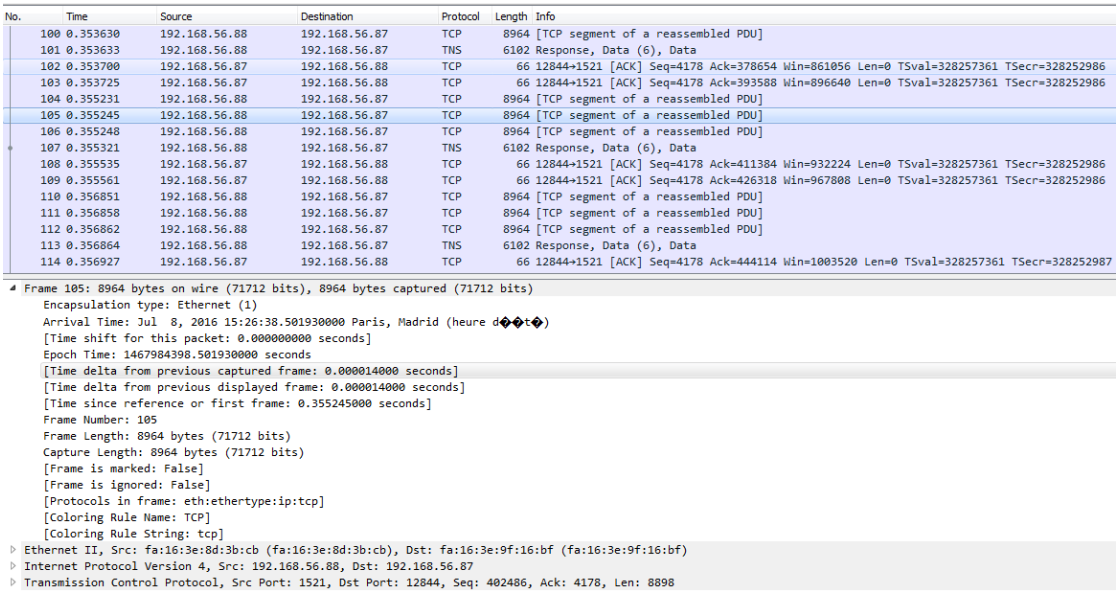


Figure D.5: Overhead time of fragmentations of message of M=32KB into network packets according to MTU (which is setted to 8.95KB).