A Framework for Ranking Data Sources and Query Processing Sites in Database Middleware Systems

By

Eliana Valenzuela Andrade

A thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
In
COMPUTING AND INFORMATION SCIENCES AND ENGINEERING

University of Puerto Rico
Mayagüez Campus
2009

Approved by:

Jaime Ramírez Vick, Ph.D. Member, Graduate Committee	Date
Pedro I. Rivera Vega, Ph.D. Member, Graduate Committee	Date
Domingo Rodríguez Rodríguez, Ph.D. Member, Graduate Committee	Date
Manuel Rodríguez Martínez, Ph.D. President, Graduate Committee	Date
Hector J. Carlo, Ph.D. Graduate School Representative	Date
Nestor Rodríguez Rivera, Ph.D. Chairperson of the Program	Date

ABSTRACT

A Framework for Ranking Data Sources and Query Processing Sites in Database Middleware Systems

By

Eliana Valenzuela Andrade

This dissertation presents a novel approach to the problem of finding the characteristics of the data sources and query processing sites in a distributed database system. We model the network as a graph with nodes representing data sources and query processing sites, some of which might be replicated.

We introduce a heuristic technique inspired in Ant Colony Optimization Theory to dynamically discover, assess, and catalog each data source or query-processing site. Our goal is to find and update possible paths to access the computational resources or data provided by the highest quality sites.

We define this concept of quality in terms of performance and freshness. We define the possible mathematical models for each one of these measures. We study different techniques to launch the ants from each node to explore the system, based on the idea of rounds. We discuss the development of the "Lazy Ants" approach to send ants to explore the system, which reduce the number of ants in the system but keeps a high quality of the metadata.

We discuss our system prototype developed using CSIM and also present performance and freshness studies designed to analyze the quality of paths found by the

Ant Colony based approach. These experiments show that our algorithm can quickly discover high quality sites from which data or query processing capabilities can be consumed.

Finally, we present a summary of results, contributions, and the future work for this research topic.

RESUMEN

Un Esquema Para Caracterizar Fuentes de Datos y Sitios de Procesamiento de "Queries" en Sistemas de Bases de Datos de Tipo "Middleware"

Por

Eliana Valenzuela Andrade

Esta disertación presenta un novedoso enfoque para caracterizar las Fuentes de Datos y Sitios de procesamiento de "queries" en un sistema de base de datos distribuídos, algunos de los cuales pueden estar replicados. Durante el proyecto modelamos la red como un grafo donde los nodos representan los sitios de procesamiento de "queries" y las fuentes de datos.

Para lograr el propósito de este proyecto se utiliza una técnica heurística inspirada en la Teoría de Optimización de Colonias de Hormigas ("Ant Colony Optimization Theory" en inglés) para dinámicamente descubrir, evaluar y catalogar cada fuente de datos ó sitio de procesamiento de "queries". El objetivo principal es hallar y actualizar a través del tiempo los posibles caminos de mejor calidad para acceder a los recursos computacionales ó de datos. También se define el concepto de calidad en términos de desempeño y frescura. Adicionalmente se muestran los modelos matemáticos usados para estimadar estas métricas. Adicionalmente estudiamos distintas técnicas para enviar las hormigas desde cada nodo para explorar el sistema, basado en la idea de rondas. Discutimos el acercamiento de las "Hormigas Vagas" como técnica novedosa para explorar el sistema y que permite reducir el número de hormigas enviadas pero mantiene la calidad de los metadatos.

Explicamos nuestro prototipo desarrollado usando CSIM y también presentamos estudios experimentales sobre desempeo y frescura, para analizar la calidad de los resultados obtenidos. Estos experimentos muestran que nuestro algoritmo puede descubrir rápidamente rutas de alta calidad donde data o servicios de "querie" se pueden consumir.

Finalmente presentamos una resumen de los resultados, las contribuciones de esta disertación y el trabajo futuro para este tópico de investigación.

To Alonso, my great husband, Maria Alejandra and Daniel Felipe, my amazing kids, Jaime, my wonderful brother, and Jaime and Yineth my lovely parents.

ACKNOWLEDGMENTS

This thesis is the result of years of work during which I have been accompanied and supported by many people. It is now my great pleasure to take this opportunity to thank them.

First, I would to thanks to my advisor, Professor Manuel Rodríguez Martínez, for his guidance and support. He has been an excellent advisor who can bring the best out from his students, an outstanding researcher, and a wonderful human being who is honest, brilliant and helpful to others. Was a pleasure work form him for the last two years and he is the better example as researcher and professor that I have had. I would also like to thank my Ph.D. committee members, Dr. Jaime Ramírez-Vick, Dr. Domingo Rodríguez-Rodríguez and Dr. Pedro I. Rivera-Vega for their support and interest in this thesis.

Also I would like to thanks to Professor Nestor Rodríguez and all the staff of the CISE program for their constant support. Thanks to all my professors during the past years for their guidance. Thanks to Professor Mercedes Ferrer for her support and guidance during the critical moments, ten years ago and last year. Thanks to my ADMG partners, Fernando, Harold and Osvaldo, for their help and knowledge. Thanks to Juddy, Oliver, John, Lolita, Juan Pablo and Marie for their help and partnership during the bad and good moments over the past five years. Thanks to my all my friends for their support during this time.

Thanks to Dr. Naduthur, Nora, Dr. Chaparro, Dr. Ramírez and Tamar, Professor Rojas, Dr. Vera, Evelyn and Priscilla for your support and because all of you trust in me.

My husband, you were extremely supportive in my work, so thank you very much. Maria Alejandra and Daniel Felipe, you are the biggest motivation to finish my dissertation, so thanks for your unconditional love which helped me during the hardest moments. Thanks to my mother and father for their support and prayers to finish

this thesis. Thanks to Carmen Alers and all her family, because they were and will be our family here in Puerto Rico forever. And last, but not least, I would like to acknowledge my brother for following up my work and always be there for me.

Finally, and most importantly, thank GOD for giving me wisdom and guidance throughout every moment of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The Internet has become a valuable tool for scientists and engineers to publish data collections produced as part of their research projects. Nowadays, we can find terabytes of data from Biological experiments, Geospatial instruments, atmospheric observations and so on. These data collections are often replicated to increase their availability and reliability. Likewise, there are many software tools and computing environments necessary to process and analyze these data in order to generate new knowledge. Many of these computing resources are replicated as well, although this replication might be more in terms of functionality rather than exact computing hardware. For example, two sites might provide a service to analyze satellite images and extract information about surface radiation, but one site might be a Red Hat Linux cluster with 128 nodes, whereas the other one is an IBM AIX cluster with 256 nodes.

The users of these data products are interested in more that simply downloading the data and running specific algorithms on these. They also want the data to be join with data from other sites, and even produce new data as a result of this integration. This requirement has encouraged numerous research activities in data integration, including the development of database middleware systems to integrate and access these data collections (see Figure 1.1). Depending on the data model of the data sources, different approaches have been used to achieve this integration: a)

Navigational integration, b) Warehouse Integration, and c) Mediator Based Integration (see [17], [35], [50], [61]). In all of these approaches, one or more integration servers (IS) broker access to the data sources, while a set of wrappers (W) control access to the data.

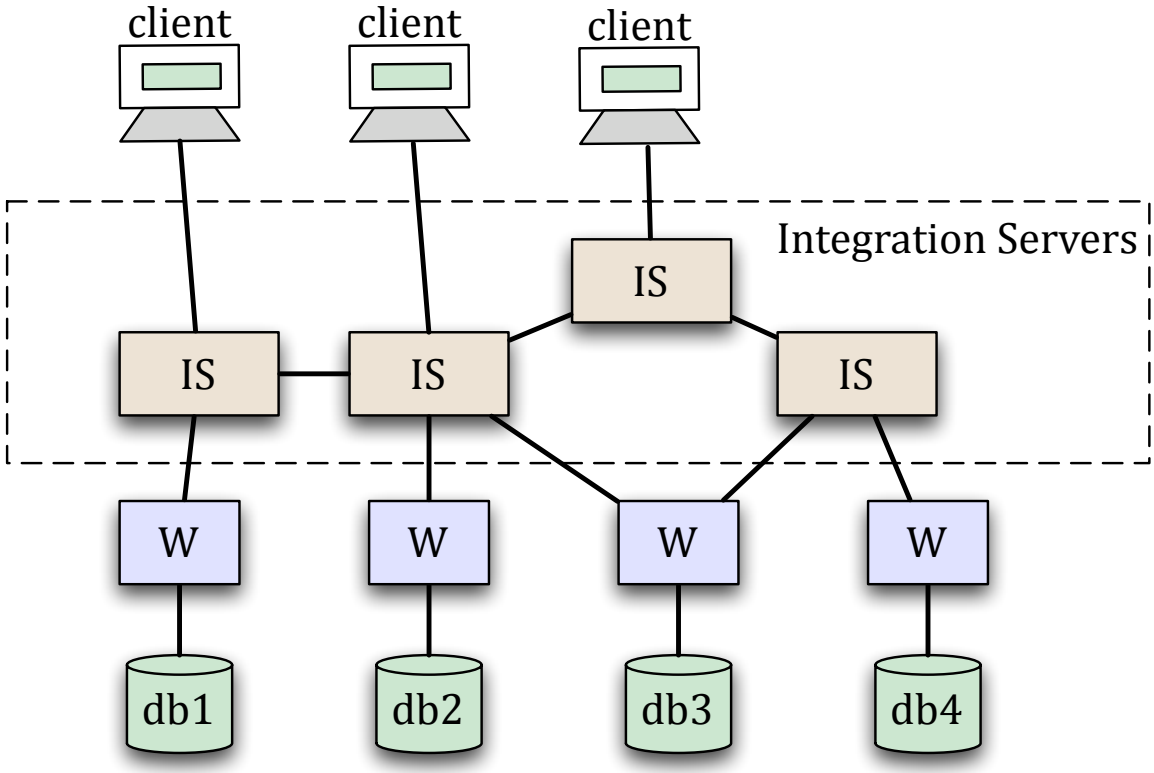


Figure 1.1. Database Middleware Architecture.

However, this data integration cannot be effective unless the integration system has accurate information about the contents of the data sources and the performance characteristics of the sites capable of processing the data. Thus, given a query Q it is necessary to determine which data sources can provide the data to satisfy the query and which sites can provide the computing power to generate the query results. Typically, query optimizers for middleware systems have relied on a catalog system that has such information. However, little attention has been paid to the fact that such information might change as the data sources are updated or query processing

sites become loaded with requests. The distributed nature of the problem makes it unfeasible to have a solution that relies on system administrators to periodically update and publish system metadata through the system. Likewise, trying to discover the information at query time can only slow down query execution and reduce system throughput.

The proposed solution addresses this problem by proving a methodology to periodically a) discover the characteristics of the data sources and query processing sites, and b) establish a new rank of these sites based on their quality at the moment. Here, quality is a domain and implementation specific qualification. It might be defined in terms of the raw computing powers, or based on the freshness of the data or some other characteristic. For example, a simulation application might need to access the sites with the most processing power. In such a case, quality refers to raw performance power of available CPU, memory, disk and network. But a stock market analysis application might require the latest data, and here quality refers to sites updated with the latest financial information.

Since a distributed system can be modeled as a graph, we can interpret our problem as a type of "shortest path" search as we can see in Figure 1.2. That is, given a site q_1 that receives a query Q we need to find the shortest paths to data sources $s_1, ..., s_n$ and processing sites $p_1, p_2, ..., p_k$ necessary to answer the query. Here a path represents the interconnection to components participating in solving the query. The cost of the path might represent response time, resource source usage, or last update time for the data. These paths can then be fed to a query optimizer as candidate sub-plans to be examined in search of the global query plan.

1.1 Motivation

Consider a global database schema DB_1 with information about protein sequences and other protein-related information. The relations in this database hold information

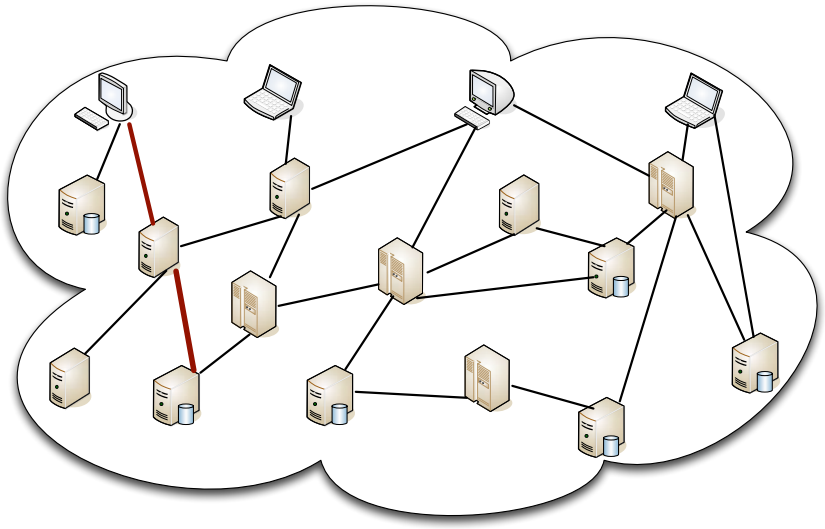


Figure 1.2. Network as a Graph

about sequences, structures and conserved domains. Let us also assume that two research labs Lab_1 and Lab_2, have local databases that follow the DB_1 schema and are replicas of each other (with small variations due to out-of date data). A third research lab Lab_3 has a second schema DB_2 with relations storing information about the taxonomy for the species from which a protein sequence has already been derived. Users of these data need to integrate the two schemas to integrate the protein data to the species data.

Suppose that a middleware system is used to achieve the data integration. Consider the following query that is posed by a user at research lab Lab_3: *Get the sequence of ten proteins in homo sapiens that are related to diabetes.* A client application C sends a query to a server on Lab_3 which gets the data for the species from the tables of DB_2 at Lab_3 . This server must join these data with the corresponding data from tables of DB_1. The question is: which site shall be used: Lab_1 or Lab_2? Let us denote the table with species information as R and that with protein information as S. Then we need to compute the expression $R \bowtie S$. Figure 1.3 depicts three possible scenarios that we might use to resolve the query. We can propose other scenarios and include all the strategies to select the better way to do the join, but this is irrelevant

for the purpose of this example.

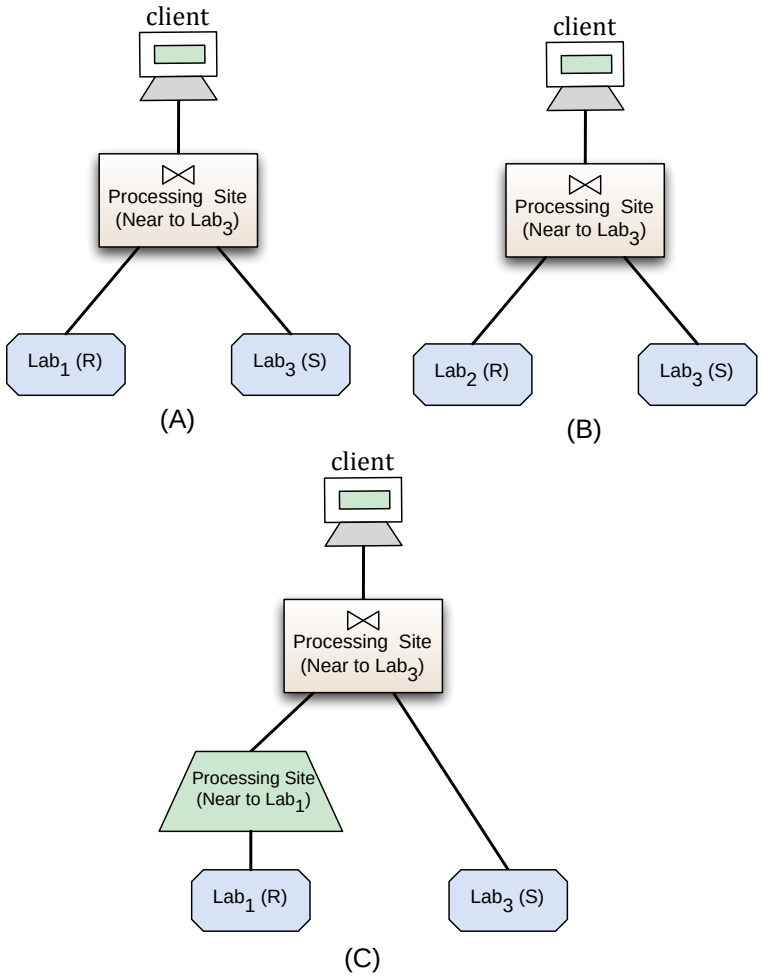


Figure 1.3. Possible Scenarios to Resolve the Query

The query optimizer must be able to choose from these scenarios, and generate a final plan. The goodness of this plan will depend on how accurate is the information regarding the performance or data freshness at sites Lab_1 and Lab_2, according with needs, desires, and expectations of the client. Moreover, if a fourth site Lab_4 is deployed with data corresponding to DB_1, it is important to discover and characterize such site. Clearly, an autonomous and de-centralized mechanism must be available to collect, organize, rank and publish site information.

1.2 Problem Statement

We can formalize this discussion as follows. Let $G(V, E)$ be a directed graph where V is a collection of sites and E is a collection of edges that represent connectivity between sites $v_u, v_v \in V$. Graph G is directed since connectivity between sites might be asymmetric (e.g., DSL links). Nodes in V can be classified as data providers (i.e., wrappers) or as query processing providers (i.e., integration server). When a query Q is received at a node $v_u \in V$, we need to find the shortest paths to a collection of nodes $U \subseteq V$ that can be used to solve the query Q. Each path is of the form $(v_u, v_w, ..., v_v)$, such that v_u is a *query processing provider* whom receives a client query and v_v is a *service that can access the data* with one or more relations of interest to the *query processing provider* v_u. Each of these shortest paths leads to high quality data sources and query processing sites. These paths can then be fed to a query optimizer as candidate sub-plans to help in finding the actual plan to solve the query. Given a path P in G, the cost $C(P)$ of this path is defined by a domain-specific cost metric m, which can be response time, resource usage or last update time, among others. This metric characterizes the quality of the solution query plan to answer the query.

1.3 Contributions

The main contributions of this dissertation can be summarized as follows:

- Development of a de-centralized approach to dynamically characterize data sources and query processing sites in a distributed database system. Evidence is presented about its potential benefits in supporting the query optimization process in distributed and replicated systems that do data integration via middleware technology.
- Definition of a quality metric for data sources and query processing sites. This

quality metric can be defined in terms of performance or data freshness. The goal of this quality metric is to establish a rank for the sites and system from the perspective of a particular site. Using this rank, the candidate sites for data extraction and query processing can be chosen and fed to a query optimizer to generate a query plan. To the best of our knowledge no other middleware system performs such assessment of data sources.

- The development of heuristic technique based on Ant Colony Theory [22] to implement the site characterization process. These technique has been shown to provide good solutions to problems in other areas in Computer Science and Networking, and we expect to capitalize on this experience in our research project.
- The study of different techniques to launch the ants from each node to explore the system, based on the idea of rounds.
- The development of the Lazy Ants approach to send ants to explore the system, which reduce the number of ants in the system while keeping a high quality metadata.
- The implementation of a system prototype using Java and CSIM. This implementation shows that our prototype version of the Ant Colony Optimization (ACO) algorithm is able to find good paths between the nodes, in a set up where the cost between nodes changes over time. This implementation experience will be very useful in our future integration with the NetTraveler [68] Prototype.
- A publication of initial results of AntFinder in the 2009 Asia Modeling Symposium [67].

1.4 Dissertation Structure

This Chapter has addressed the introduction of this dissertation; the rest of the document is organized as follows. We first develop the necessary background theory in Chapter 2 about Distributed Databases and Middleware Systems and data replication. Chapter 3 presents an overview on the Middleware architecture used: The NetTraveler System. Chapter 4 presents an Ant Colony Framework. Then, Chapters 5 and 6 present the framework about the metrics with the experimentation results. Then, Chapter 7 presents our approach to explore the artificial ants will be launched on the network and the experimentation results. After that, Chapter 8 considers Ethical issues and finally Chapter 9 presents the conclusion and future work based on this dissertation.

CHAPTER 2

Literature Review

2.1 Overview

This Chapter presents relevant work in the areas that form the basis of this dissertation, which include: Distributed Database Systems, Database Middleware Systems, and Data Replication. An extensive amount of work has been carried out in these fields; hence this Chapter discusses only the aspects that are more relevant to this document.

2.2 Distributed Databases

Distributed Database Systems (DDBS) is composed by a collection of Database Management Systems (DBMS) that are physically apart and connected via a computer network. Theses DBMS have agreed to form a federation of sites that share their collections of data and query processing capabilities.

Several DDBSs have been prototyped in the last decades; among the most interesting we have R* [70] by IBM, and Mariposa [63] developed by the University of California at Berkley. Figure 2.1 shows the typical architectural organization of a DDBS. In general, the following assumptions are made by the majority of the Distributed Database Systems [5, 63, 19, 34, 41]:

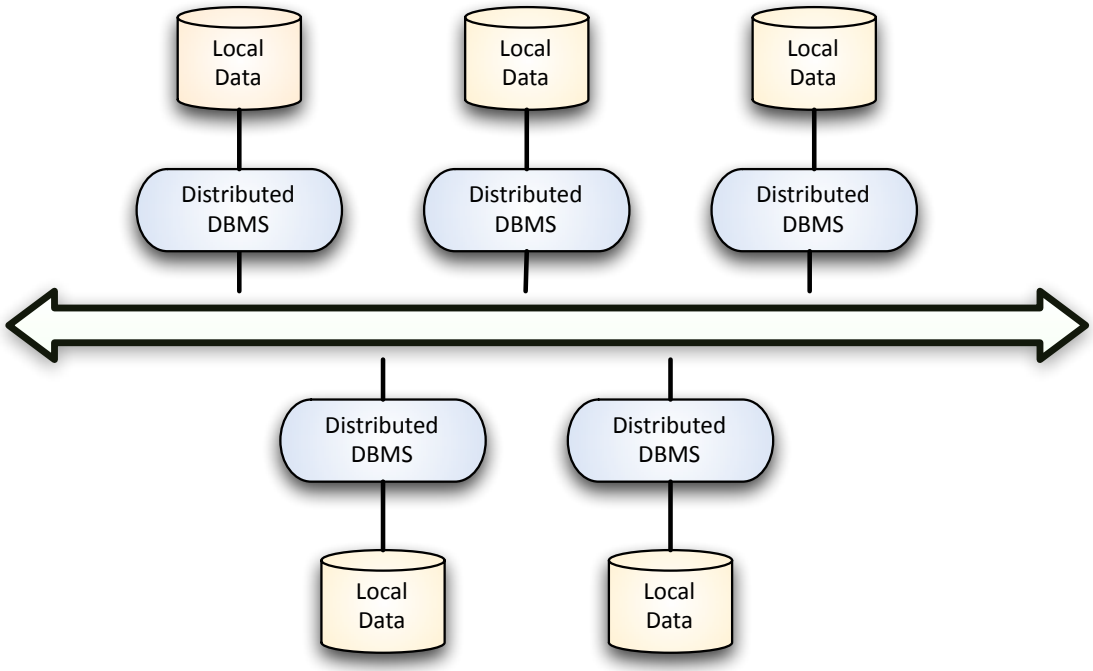


Figure 2.1. Typical Distributed Database Architecture

- All sites are independent from one another and autonomous. No central authority dictates what data should be stored at each site or how it should be accessed.
- All sites follow the same data model, typically the relational model.
- All sites run the same DBMS software or a DBMS for which there is a common communication protocol.
- Users should be able to make queries without knowing or specifying where the relations are located.
- Users should be able to perform transactions that affect data at several sites just as they would execute transactions over a local Database Management System.
- The effects of a transaction across sites should be atomics; all the changes persist if the transaction commits and none persist if it aborts.

Early DDBS, like R*, assumed that all sites were full-fledged database servers. Latter, an approach in which the DBMS was divided into a client-server architecture

was introduced [19]. In this architecture, the server, or back-end component, is responsible for managing the data and executing transactions. Meanwhile, the client, or front-end component, has the role of interacting with users, requesting data from one or more servers. Servers are run in machines with very fast disks, multiple processors and lots of memory, while clients are run on a vast array of machines, ranging from workstations to handheld devices. This client-server DBMS architecture became the most studied in the past two decades because of its simplicity of implementation due to the clear separation of functionality between client and server. The bulk of the work done has been related to server-side replication and client-side caching techniques ([19], [64]). The idea behind these techniques is to minimize communication costs by reducing network access.

Query optimization techniques in DDBSs have resembled that of traditional DBMSs, pioneered by the System R* prototype [5] developed at IBM. This optimization strategy, described in [30], is based on dynamic programming and follows a cost-based model to calculate total resource consumption for a query. Each operator in the plan is given a cost and the overall cheapest plan, calculated by adding the cost of each operator, is always selected. In a typical, single-site DBMS the cost estimate is dominated by disk access time. In a distributed environment, however, other factors such as communication costs and differences in local computational costs must also be taken into consideration. The R* prototype system was one of the very first to introduce these additional factors into the cost model. The "classic" cost-model has been proven useful in optimizing the overall throughput of a system. However, this type of optimizer will not always find the plan with the lowest response time for a query in cases where the machines are lightly loaded and the communication channel is fast, since it cannot take into consideration intraquery parallelism [41]. Intraquery parallelism occurs when a query plan has several operators that can be evaluated in parallel because they can be evaluated at different sites. Another model that does

take into consideration intraquery parallelism is the response-time model [41].

2.3 Database Middleware Systems

Database middleware systems have been used as a solution to integrate heterogeneous data from multiple sources. Database middleware arise as an alternative to Distributed Database Engines such as R* [70], and Mariposa [63], which required existing data sources to be purged, and their data re-ingested into a DDBMS common to all sites. Existing database middleware solutions have an architecture based on a central integration server to provide client applications with a uniform view of the data, and a single-point of access to the federated sites. The integration server relies on the capabilities of translators to extract the data from the sources, some of them replicated, and perform schema mapping operations to convert data from local schemas into a global schema specified by the client to the integration server. Once the data items have been translated, they are sent back to the integration server for further processing. Most of the query processing occurs at the integration server site and the data sources often act as mere Input/Output (I/O) nodes. A catalog associated with the integration server provides the metadata necessary to guide the process to find data sources, schema mapping rules, and query processing strategies. This topic is discussed in Section 2.4.1.

Two approaches dominate the spectrum of possible database middleware implementation schemes. The first approach is to use a relational DBMS such as Oracle or IBM DB2 as the integration server, and use database gateways [13, 14, 15, 19] as the translators that allow the integration DBMS to access distributed data. This approach has been supported mostly by the commercial sector. In the second approach, a Mediator System [4, 20, 27, 55, 57] specifically customized for distributed processing is employed. This second solution features an integration server called the mediator, and a group of wrappers acting as the data translators. This latter approach has

been supported mostly by research groups from both industry and academia.

2.4 Database Middleware Architecture

It is, we assume that the database middleware system is based on an architecture on which one or more integration servers (IS) connect to various wrappers (W) that take care of extracting the data from the sources, as shown in Figure 2.2. The integration server layer imposes a global schema on the heterogeneous data sources, and all queries posed by the user are expressed in terms on this global schema. We assume an *unstructured system*, where there is no central coordination site. The nodes form an overlay network for the purpose of exchanging tuples related to a given query. Each integration server contains a local catalog with metadata representing its own global schema, data source sites, users permissions, and so on. Without loss of generality, and to simplify our presentation, we assume that this schema follows the relational model and that all queries are expresses in SQL.

A client application sends its queries to one of the integration servers, and this server connects to other integration servers and wrappers to get the query solved. We assume that the integration servers have capabilities to either negotiate access to a query processing infrastructure or provide it altogether by means of a query execution engine. The integration servers rely on the wrappers to: a) extract the data from the sources, b) map the data from the local schema into the global schema, and c) execute some of the query operators necessary to generate the results. The wrappers deliver their results to the integration server(s). Finally, the integration server originally contacted by the client, takes care of collecting all results and delivering them to the client.

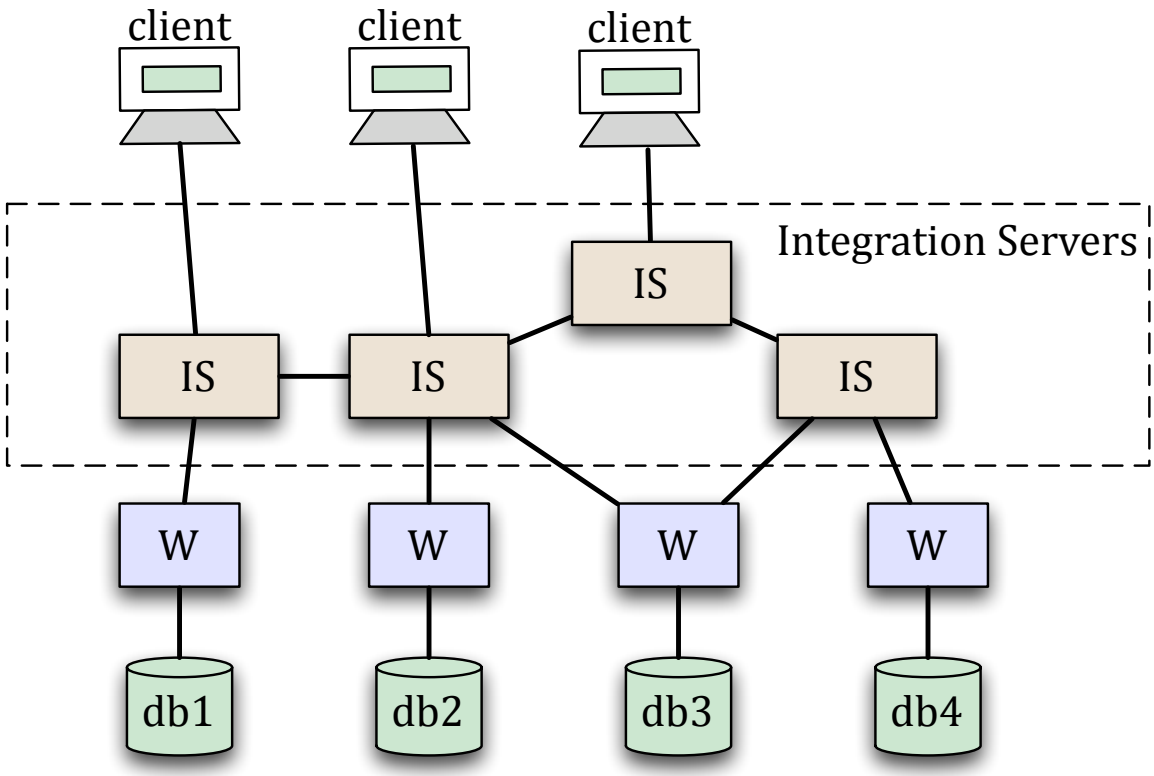


Figure 2.2. Typical Database Middleware Architecture.

2.4.1 Middleware Systems Catalog

System R* [46] and Distributed Ingres [62] relied on the existence of a catalog with metadata about the characteristics of the system components, without providing techniques for loading the information into this catalog in an autonomous fashion. The catalog system in Ingres [62] was located in a central site. In contrast, System R* used a distributed catalog mechanism, in which, each participating site publishes metadata about the data it is hosting. The approach that we introduce in this dissertation provides a mechanism by which the system can discover new sites, rank them in terms of performance, or some other criteria, and then load these data into the catalog in an autonomous fashion. Whether the middleware uses a centralized or distributed catalog, our framework can be used for discovery and ranking of the sites in the system.

The Mariposa [63] system uses an economic paradigm for query processing. In

this scheme, a bid is placed among participating sites in the middleware system to assign query processing operators. Alternatively, a purchase order is placed for a specific site to run one or more query operators. Mariposa also assumes the existence of a catalog to find candidate sites for the bid and relies on an advertisement system in which servers announce their willingness to run queries. The characteristics of sites involved in either a bid, or purchase order, are evaluated as part of the offer they provide, and a decision for query operator assignment is reached based on the best deal found in the process. The authors in [63] indicate that the bidding process done at query time results in a somewhat slow process and the authors advocate for the purchase order concept, in which a site in need for query processing simply submits a work order to a site for which a pre-order agreement for query processing has been established. Our ant-based approach differs from this scheme, because the agents are constantly exploring the system to discover new sites and to update the status information for already discovered sites. At query time, the catalog has the information with the most likely sites that can handle the query in an efficient manner. Thus, there is no need to go into an expensive biding process to identify sites to run the query. Moreover, our system can reduce the possibility of falling into the trap of submitting a purchase order to a site that cannot honor its agreement due to a work overload ("over booking"), because the ants are constantly evaluating the characteristics of the nodes in the distributed system. Certainly, combining our framework with the economic paradigm proposed in Mariposa can be an interesting solution because it opens the possibility for validating "vendors" (i.e., query processing sites) against their actual behavior, just like vendors are evaluated in online stores such as eBay. This opens up the possibility for having purchase orders that reflect a more factual information about current system dynamics.

The Garlic system from IBM [33, 57] explored the issues of data integration and query optimization across diverse data sources. In Garlic there is a central cat-

alog with information about remote data sources, and the mediator system explores the capabilities of those remotes sites at run time. In contrast, our framework uses the artificial ants to constantly look for sites and assess the capabilities of new or existing sites so that the query optimization can be started quickly and with accurate information. Our framework can be coupled with Garlic to enable a more comprehensive discovery of the capabilities of remote data sources in a highly heterogenous environment.

The MOCHA [54] middleware explored the issues of automatic code deployment at run time, in a effort to load the code needed for query processing at strategic locations that could result in a reduction of data transfer and increased response time. Like most of the previous systems mentioned here, MOCHA assumed the existence of a catalog with metadata about schemas, data sources, and code implementing query operators. Our framework can be used to extend MOCHA, enabling it to discover a wider range of code repositories. This can result in a wider selection of functions and query operators to efficiently run the queries submitted to the user. Recent work in the *AReNA* [74] is closely related to this aspect of our work. The authors in this system investigate mechanisms to obtain latency information from data source and query processing sites in the system. This information is then aggregated to obtain a performance profile of the sites, which is stored in the system catalog. The catalog system itself is distributed throughout the system. Our approach differs in the use of ACO algorithms to discover and rank the sites. In addition, other performance characteristics such as power consumption (critical in mobile environments) could also be added into the system.

2.5 Data Replication

Data replication is a technique used to copy frequently accessed data into several remote server to increase availability of the data by having multiple redundant copies,

as shown in Figure 2.3. Data replication has been studied extensively in the research literature [29, 71, 39], and in most cases the focus has been on algorithms to update replicated data collections as efficiently as possible. The work in [24] explored a mechanism to locate replicated objects in a distributed database by means of a distributed index but data freshness was not a consideration. Recent work in replication middleware [10, 52] has focused on middleware solutions that implement replication schemes atop commercial database engines. Again, the focus is on how to keep replicas as fresh as possible with respect to the master. Our work differs from (but complements) these efforts since the problem we are considering is how to pick the replicated data collection(s) that satisfy the freshness constrains imposed by the user on a query.

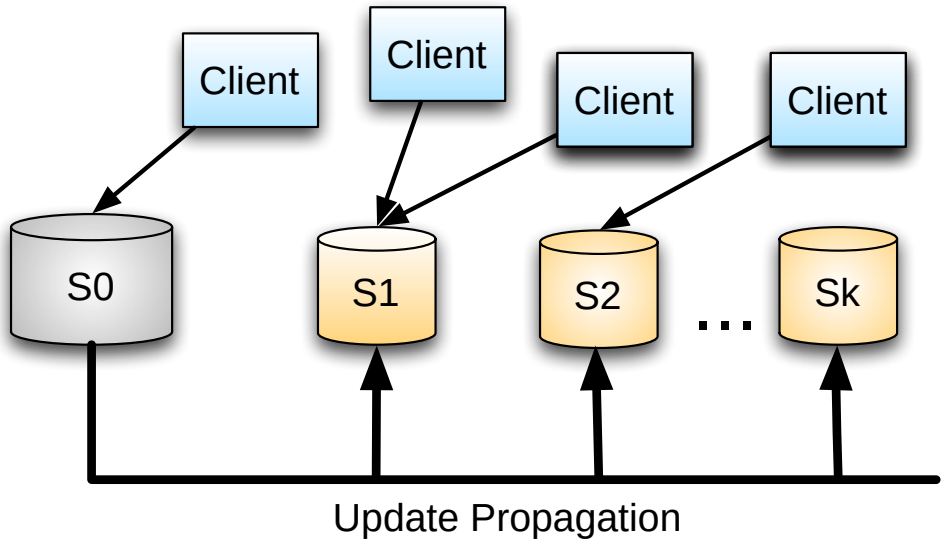


Figure 2.3. Access to replicated data collections.

Data freshness has been studied before in the context of materialized views and web views. Data freshness based on the currency of data was studied in [52] as a mechanism to keep materialized views up-to-date. The authors in [42, 43] use data freshness to guide the update process of materialized web views inside a web server. The major goal is to use the freshness metric as part of an algorithm to choose

which web views to materialize and keep fresh. Meanwhile, the authors in [11] use data freshness to develop strategies to frequently update a local copy of a remote database. However, these approaches do not address the issue of extracting the data from a replica that is fresh enough to provide adequate answers to a query.

Various definitions of data freshness are analyzed in depth by the authors of [7], from which we incorporated and adapted several of these freshness metrics. Their dissertation presents a taxonomy of data freshness metrics, and analyzes their applicability for managing replicated or cached data in data warehouses, database middleware systems, and data caching systems. AntFinder leverages on these metrics, but also addresses the problem of routing queries to the sources that have data with the necessary freshness to satisfy the user's request.

The work in [56] most closely related to our own. This study explored an approach to route OLAP queries within a database cluster to the database node that has data fresh enough to satisfy the user request, using an algorithm called Freshness Aware Scheduling (FAS). The desired level of freshness is specified as a parameter of the query request, but is not integrated into the SQL string used to specify the query. Their algorithm also handles routing of updates to database nodes, while keeping data consistency. However, their approach is more difficult to scale to a wide-area network since it uses a centralized scheduler to perform all query and update routing. AntFinder provides a decentralized alternative for database middleware systems that connect sources over a wide-area network.

CHAPTER 3

NetTraveler System

3.1 Chapter Overview

In this Chapter we present an overview of NetTraveler [68], a database middleware system that is been developed by the Advanced Data Management Group (ADM) at University of Puerto Rico, Mayagüez, under the supervision of Dr. Manuel Rodríguez-Martínez. NetTraveler is the model database middleware system on which our AntFinder system operates.

3.2 Architecture Overview

NetTraveler [68] is a middleware system designed for Wide Area Networks (WANs), which are modeled in NetTraveler as a collection of applications $H = \{h_1, h_2, \ldots, h_n\}$, each having a specific role in helping a client to solve a given query. The collection of applications H is running on host computers spread over a group of LANs that compose the entire WAN environment, as shown in Figure 3.1. These LANs consist either on wired or wireless technologies, such as Ethernet, DSL, IEEE 802.11b, and 3G networks.

From the set H we have a subset of applications $C = \{c_1, c_2, \ldots, c_i\}, i < n$, which have client capabilities to submit queries. These capabilities come from running

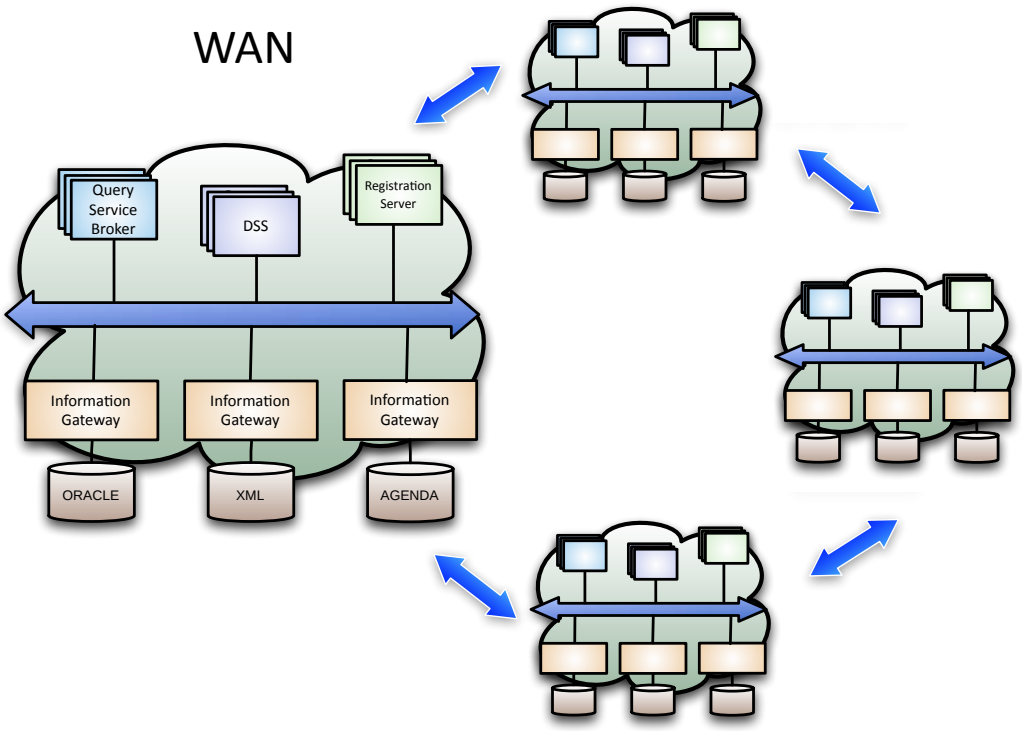


Figure 3.1. NetTraveler Architecture

a given interface (most likely a GUI) that the end-user can use to pose queries to the system. The data to answer those queries comes from another subset $S \subset H$ known as the data sources $S = \{s_1, s_2, \ldots, s_j\}, j < n$.

Each data source $s \in S$ is an application such as a DBMS, Web Server, XML-based data server, or some other customized server application. When a client $c \in C$ needs to pose a query to sources in S, it needs to contact a server application known as the *Query Service Broker (QSB)*.

A collection of *QSBs* $B \subset H, B = \{b_1, b_2, \ldots, b_k\}, k < n$, take on the responsibility of finding the computational resources (data, disk access, CPU time, network time, etc.) required to extract data from the target data sources in S to answer the queries posed by the clients in C. *QSBs* perform query related tasks such as query parsing, query optimization and query execution. Also, *QSBs* exhibit Peer-to-Peer (P2P) behavior since a broker might contact other brokers in B to assist solving a given query. This is done to prevent a centralized operational model, in which a cen-

tral broker needs to know all data sources, and becomes a focal point through which all queries must pass. This would make the system unreliable and inefficient as the central broker site becomes a single-point of failure and a performance bottleneck.

The *QSBs* in B can access the data sources in S by means of a server application known as the *Information Gateway*. A collection of *Information Gateways (IGs)*,$G \subset H, G = \{g_1, g_2, \ldots, b_m\}$, $m < n$ have the role of providing access to the brokers to the wealth of information contained in the data sources in S. It is at this level, of the *IGs*, that data extraction occurs. *IGs* can currently extract data from relational databases such as PostgreSQL and MySQL or another information sources, such as records store on different files. In addition, *IGs* can execute query operators, particularly those that can filter out unwanted results, such as predicates. Clearly, metadata is needed for the brokers to be able to find the required data sources and their associated *IGs*. These metadata must be spread throughout the system to advertise the availability of resources. The responsibility for this metadata dissemination is given to a type of server known as the *Registration Server (RS)*.

The collection of *Registration Servers (RSs)*, $R \subset H, R = \{r_1, r_2, \ldots, r_p\}, p < n$, deals with the problem of advertising metadata, encoded in XML, describing resources such as: query operators, local tables, global tables, data types, CPU cycles, data sources, network bandwidth, disk space, and so on. Two or more *RSs* work as peers to exchange these metadata, just as network routers advertise routes to each other to enable future packet forwarding decisions.

The last two elements in NetTraveler are known as the *Data Synchronization Server* and the *Data Processing Server*. A collection of *Data Synchronization Servers (DSSs)*, $D \subset H$, $D = \{d_1, d_2, \ldots, d_t\}, t < n$, groups server applications that help clients in caching query results, obtaining extra disk space, and keeping synchronized copies of data natively stored by a client which also happens to behave as a data source from time to time. More importantly, a *DSS* can become a **proxy** for a client

$c \in C$, gathering the results intended for client c if the client goes offline or experiences some type of failure.

Finally, a collection of *Data Processing Severs (DPSs)*, $P \subset H$, $P = \{p_1, p_2, \ldots, d_v\}$, $p < n$, contains the server applications that provide a commodity service for computational tasks during query processing. These tasks include query execution, sorting, or any other type of specific computational operation required.

The elements in NetTraveler are logically organized into groups of cooperative applications known as *ad-hoc federations.* Federations are ad-hoc because they can be formed or dissolved over time, based on the decisions taken by its members. A federation can spawn more than one LAN, and a LAN can have elements that belong to more than one federation. The simplest federation is one made out of one local group, which consists of one *QSB*, one or more data sources and their associated *IGs*, one or more clients, one*RS* one *DSS*, and one *DPS*. In some instances, having a *Data Processing Server* might be optional, particularly in cases where the applications only require simple queries to the data sources. *DPSs* will be most likely used in environments that require complex processing capabilities, or which have many low-powered devices.

Two local groups L_1 and L_2 can be combined to form a cluster by making the data broker from L_1, B_{L_1}, become a peer of the broker of L_2, B_{L_2}. In our framework, Peer relationship is bidirectional, hence B_{L_1} becomes a peer for B_{L_2}. As a consequence of these events, the *RS* in each local group becomes the peer of its counterpart in the other local group, and they begin exchanging metadata about the resources available in each local group. A cluster of three local groups can be made by adding a third local group L_3 and making its broker,B_{L_3}, become the peer of either B_{L_1}, B_{L_2}, or both. The same happens with the *RSs* in each one of these groups. Larger and more complex clusters can be constructed in this fashion and, as you can see, clusters represent complex federations with multiple brokers cooperating to share access to the

data. Likewise, the *RSs* in each of the groups exchange the metadata that enable the brokers to find the resources needed to solve the queries and keep track the location of client and servers as well.

CHAPTER 4

Ant Colony Framework

4.1 Overview

Our key idea is to map the problem of finding the parameters of the servers in the middleware system into a problem of finding shortest paths in a graph. The problem of finding shortest path in a dynamic network is a combinatorial optimization problem. If the system is relatively small, we can easily apply tools such as dynamic programming to find a solution. But when the system is dynamic and large, this type of methodology does not scale well, and some type of heuristic search must be employed. In fact, we can employ stochastic optimization heuristic algorithms such as *randomized search, tabu search, simulated annealing* or some other new methodologies such as swarm optimization. In this work, we used the swarm optimization technique known as **Ant Colony Optimization (ACO)**. This technique fits well our needs since it is a distributed and de-centralized approach that has been effective in solving data communication problems [21, 38, 44, 58, 59, 60]. In this Chapter, we discussed relevant aspects on Ant Colony Theory, which include: Basic Social Networks Concepts, the mapping between real and artificial ants in our middleware framework, the original and the adapted algorithm. We also review an extensive amount of work that has been carried out prior to this dissertation as well a work

done as part of it.

4.2 Behavior of Real Ants

ACO algorithms are inspired by the observation of real ant colonies. Ants are social insects, which means that the behavior of every ant is based on the survival of the colony and not on its own survival. One of the most important behavioral features of ant colonies is their capacity to find shortest paths between food sources and their nest. This behavior is achieved by an indirect method of communication between ants based on *pheromone trails*. Ants leave their nest to find sources of food, and when they do find food they return to the nest to alert the others. When an ant moves from the nest and discovers a food source, it returns to the nest leaving a pheromone trail along along the way. Other nearby ants become attracted to this trail and walk along the path leaving more pheromones, which in turn makes the path more attractive to other ants. There might be many paths from the nest to the food source being explored by different ants, but the shortest path is eventually discovered. The reason for this is that the shortest path is the one with the strongest and quickest to fill pheromone trail, as shown in the upper part of Figure 4.1. This communication mechanism was named *stigmergy* by French biologist Pierre-Paul Grassé in 1959 [28]. Notice, however, that a shortest path might become inefficient or unavailable (e.g., blocked by an obstacle or by a human spraying insecticide). Nature solves this problem by letting pheromone evaporate and by allowing ants to randomly chose to visit other paths. Thus, a path would remain good only if the ants continue to use and strengthen its pheromone level. If the ants abandon the path, the pheromone starts to evaporate and less ants use it, until it no longer works as a solution. New alternative paths are found because some ants wander into them and if they do find food, they begin the pheromone strengthening process again to alert the others. In *stigmergy* the communication takes place when individual parts of the system modify their local

environment laying down pheromones.

There are several aspects that need to be understood about this social behavior. First, the intelligence that emerges from the social network of ants has a clear goal: find the shortest path to the food. Second, such social behavior does find optimal or near optimal solutions to the problem [22]. Third, the process is completely decentralized since each ant is exploring its surrounding at its own pace and following its own individual path. Fourth, the behavior of the ants is random but biased towards movement on trails with strong pheromone. New paths can be found because some ants venture into exploring other alternative paths with less pheromone. Hence, ACO is a stochastic evolutionary process and the solution gets adapted, improved or changed over time as system dynamics change. Thus, adaptive behavior is built-in into the process.

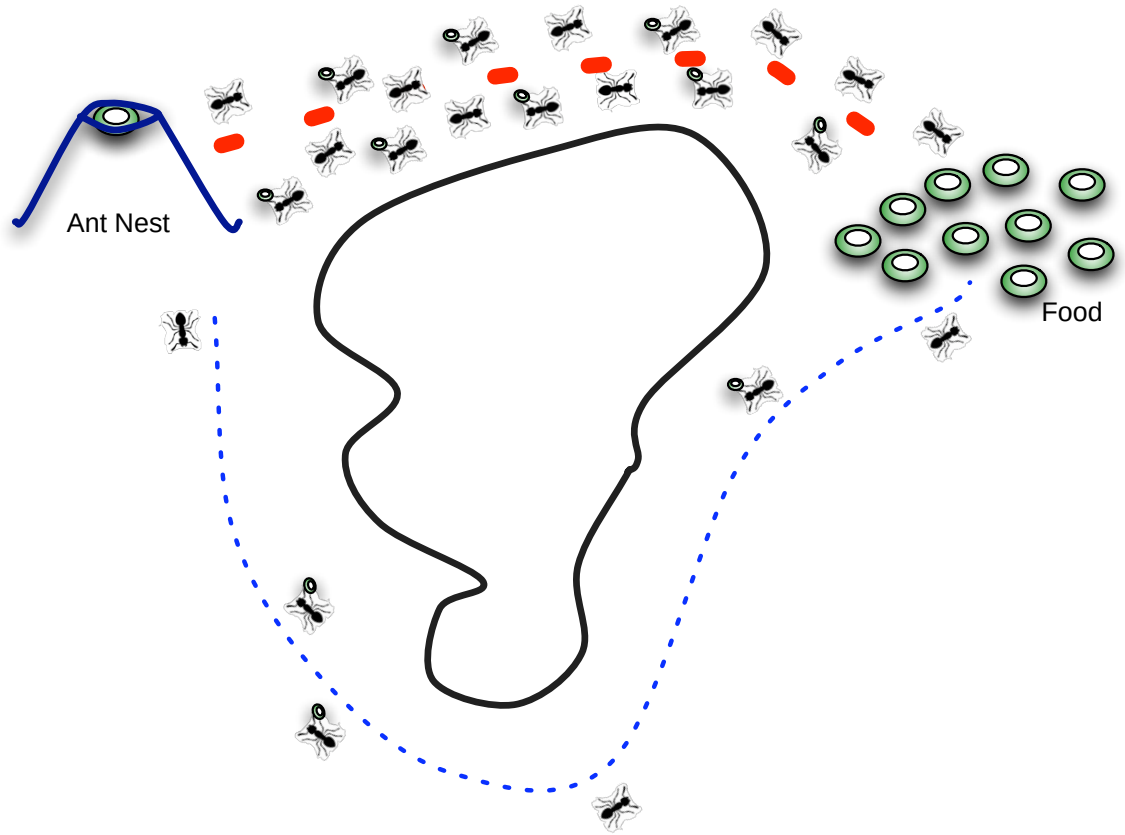


Figure 4.1. Visualization of Ant Behavior

4.3 Ant Colony Optimization Metaheuristic

According to Glover [26] a metaheuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. In the case of Ant Colony, these metaheuristic algorithms model the real ants' behavior and also add some other aspects to improve the performance and obtain optimal solutions to more complex problems.

Like a real ant, an artificial ant can construct a solution, starting from an initial state selected according to the problem scope. During this process the artificial ant can collect information about the problem characteristics and about its own performance, and uses this information to modify the problem status. A possible optimal solution is found using the incremental constructive approach. In the same form real and artificial ants may work concurrently or/and independently, showing a cooperative behavior. Although, i artificial ant it is necessary to model the stigmergy, meaning that the artificial ants do not communicate directly between them.

In each problem tackled with ACO it is necessary to define the notion of neighborhood, and how the artificial ants build a solution by moving through a (finite) sequence of states in this neighborhood. Moves are selected by applying a stochastic local search policy directed by (i) ant's private information (the ant internal state, or memory) and by (ii) publicly available pheromone trail and a priori problem-specific local information [22].

In the artificial ants, we can exploit its memory space to track information about the viability of the solutions. This memory can store information about the history of explored paths and can avoid taking unfeasible states. Therefore, artificial ants (as real ants) must build feasible solutions using only knowledge about the local state and about the effects of actions that can be performed in this local state. The local information includes both; problem specific heuristic information and knowledge

coded in pheromone trails, accumulated by all the ants from the beginning of the search effort. The decision about when the ants should release pheromone on the artificial "environment" and how much pheromone should be deposited depends on the characteristics of the problem. Ants can release pheromone while building the solution (on-line step by step), or after a solution has been built, moving back to all visited states (on-line delayed), or both. After an artificial ant achieves its goals of building a solution and depositing pheromone, the ant dies, which means that it disappears from the system.

In general the quantity of pheromone left by an ant depends of the goodness of the solution.This goodness must be evaluated based on some metric that characterizes the path being explored. Some metrics include expected resources usage along the path, the response time to process the data along the path, or the freshness of the data along the path. Once the ant gathers enough information to compute this metric, then it is ready to determine the amount of pheromone to leave at the site. Notice that it is also necessary to implement the pheromone evaporation mechanisms, which are run concurrently to the pheromone accumulation process. The pheromone for the artificial ants is represented by ant-decision tables (stochastic tables), which and are used by the ants' decision policy to direct their search toward the better zones. We shall see more on this in Section 4.5.

The basic ACO metaheuristic has two important components: a) the generation and activation of artificial ants for pheromone accumulation and b) the pheromone evaporation mechanism. Some additional components in artificial ant colony are the called daemon action which use global information and can modify "offline" the pheromone trial based in a particular behavior of some ants or in specific situations during the execution of the search process. In Figure 4.2 we can see a high level description of Ant Colony Metaheuristic in pseudo-code, that includes the generation and the activity of artificial ants, the pheromone evaporation and the daemon

activities as well. For each problem the algorithm must be customized to reflect the constraints and expectations of the particular problem to solve.

4.4 Mapping Real and Artificial Ants

Applying Ant Colony Theory to computational problems involves the definition of the ants, food sources and the nest. In our database middleware framework, the food are the data sources. The nest is the site from which the queries are originated. The path between the nest and the food contains query processing sites that can be used to process the data. Powerful sites can be seen as members of good paths, whereas slow sites can be seen as leading through bad paths. In our framework, the ants are small autonomous programs that visit each node in the network and inspect its characteristics. As they visit each site, they leave a bit of pheromone at the site. The amount of pheromone left is directly proportional to the quality of the site. This pheromone level is incremented as new ants visit the site and find it to be good, thus leaving even more pheromone. If the site is not good, ants leave less pheromone. Eventually, the shortest (best) path to reach a set of data sources from a given site s is found. However, care must be taken to avoid *stagnation* (premature convergence) by focusing on local minima. In real ant colonies, Nature solves this problem by letting pheromone trails evaporate over time and allowing some ants to follow new paths. If a path remains a good one, ants will follow it and keep the pheromone alive. If the path becomes bad, the ants will abandon it and the pheromone will evaporate. Thus, a good path will remain so, only if ants continue to use it. This behavior must also be incorporated in solutions based on Ant Colony Theory.

```
ACO Meta Heuristic()
1  while (termination-criterion-not-satisfied)
2        do schedule activities();

schedule activities();
4  ants generation and activity();
5  pheromone evaporation();
6  daemon actions();                ▷ optional

ants generation and activity();
8  while (available-resources);
9        do schedule the creation of a new ant();
10          new active ant();

new active ant()
12 initialize ant();
13 M ← update ant memory();
14 while (current-state ≠ target-state)
15       do A ← read local ant-routing table();
16          P ← compute transition probabilities(A, M, problem − constrains);
17          move to next state(next − state);
18          if (Online step-by-step pheromone update)
19            then
20                 deposit pheromone on the visited arc();
21                 update ant-routing table();
22          M ← update internal state();
23 if (online delayed pheromone update)
24    then
25          evaluate solution();
26          deposit pheromone on all visited arcs();
27          update ant-routing table();
28 die();
```

Figure 4.2. The Ant Colony Optimization Metaheuristic [22]

4.5 AntFinder: Ant Colony Algorithm for Data Source and Processing site Discovery

4.5.1 Middleware Representation

Next, we present conceptualization of our framework, following the mapping exposed in 4.4. In our approach, the artificial ants will walk on a graph $G = (V, E)$ as a representation of our middleware system. An artificial ant uses the artificial pheromone trails, represented by a pheromone matrix $\mathcal{T}_u$ associated with each node v_u hosting the services in the system, that can be visualized, as shown in Figure 4.3. This matrix represents possible paths to move from the node v_u to a data source or a query processing site v_v. Thus, $\mathcal{T}_u$ is an $n \times m$ matrix with rows representing neighboring nodes to v_u, whereas the columns represent possible destinations. We shall refer to v_v as the destination node. Given a node v_u, each *pheromone element* (w, v) of matrix $\mathcal{T}_u$ is denoted as τ_{uwv} and represents the learned desirability for an ant on node v_u and with destination v_v to move to service (node) v_w (e.g., use v_w). The pheromone element has three indices since the complete problem consists of the solution of many minimum cost paths: $n(n-1)/2$. Therefore, an ant on a service v_u can have any of the remaining $n - 1$ services as destination. Each column of the pheromone matrix complies with the hyper-cube framework proposed by Blum, et al. [6], to automatically rescale its value to always lie in the interval $[0, 1]$, and complying with:

$$\sum_{j \in \mathcal{N}_i} \tau_{uwv} = 1,\ d \in [1, n]\ and\ \forall i, \tag{4.1}$$

where $\mathcal{N}_i$ is the set of neighbors of node v_u.

Notice that the pheromones represent probabilities to chose the next node to visit. Hence, the sum of these probabilities must be equal to 1. Thus, the algorithm cannot simply update pheromone values, but must make sure that it normalizes the

values to always add up to 1. This implies that the the update of a pheromone value will result in the update of other related pheromone values. It might be the case that by increasing the pheromone for a path, we must decrease the pheromone in another path.

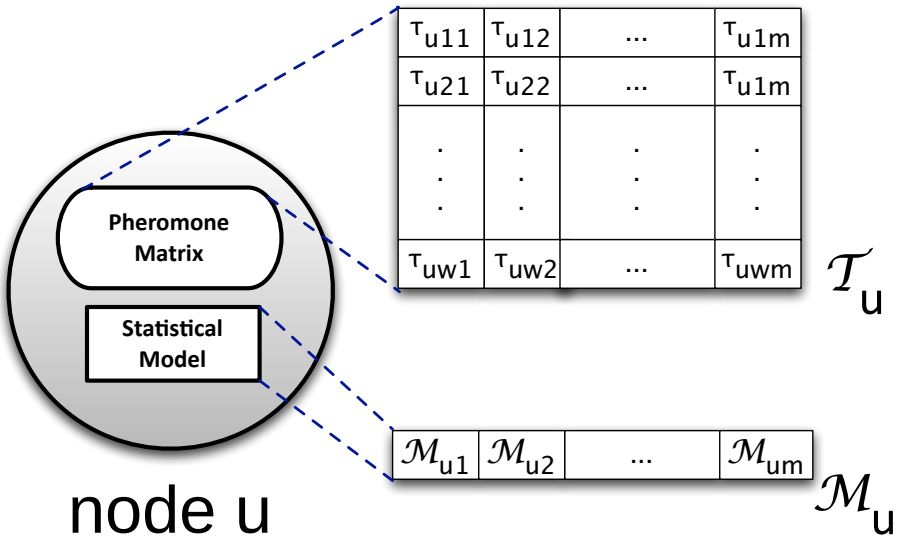


Figure 4.3. Pherome Matrix and Statistical Model at node u.

In addition to pheromone information, every service v_u maintains a parametric statistical model $\mathcal{M}_u$ of the cost of moving information (based on a performance or freshness metric) around the network as seen by node v_u. This model $\mathcal{M}_u$ is updated by the backward traveling ants. It is used to evaluate the paths produced by the artificial ants, and shall be available for the system to used it during the query optimization process. The model $\mathcal{M}_u\,(\mu_{uv},\ \sigma^2_{uv},\ \mathcal{B}_{uv},\ \mathcal{F}_{uv})$ is an array with v elements, one for each known destination node v_v from node v_u. Each element in this array is a tuple with five structures of the form $(\mu_{uv},\ \sigma^2_{uv},\ \mathcal{B}_{uv},\ \mathcal{F}_{uv})$. This model is adaptive and each element in the array is recalculated for every ant that visits the node. First and second elements are the path sample mean cost μ_{uv} and the variance σ^2_{uv} computed over the trips made by the artificial ants as we will explain later, in Section 5.3.2.

The third component is a list that store the better observed paths $\mathcal{B}_{uv}$ from v_u to v_v, ordered. The last component $\mathcal{F}_{id}$ used to store the information associated with the *freshness* of this path between node u and node v.

In summary, variables $\mathcal{T}_u$ and $\mathcal{M}_u$, could be visualized as system-wide state memory stored in nodes capturing different aspects of the network dynamics. The model $\mathcal{M}_u$ maintains metrics of all known service nodes, while the *pheromone matrix* $\mathcal{T}_u$ gives relative goodness metrics for each source-destination pair under the current connectivity in the system.

4.5.2 Ants Walk on Database Middleware System

The ACO algorithm that we have developed as part of this work, which is based on the original AntNet algorithm [23]. It has two sets of artificial ants. The first type of ants travel in a forward direction saving information about the metrics and status found at each node in an effort to discover a good path. The second set of artificial ants travel backwards, placing an updated pheromone trail and storing path-related statistics at each node, thus updating the variables $\mathcal{T}_i$ and $\mathcal{M}_i$ at each node v_i. Both sets of ants have the same basic structure but different purpose. A high-level description of the Ant Algorithm is presented in Figure 4.4 . The algorithm has two main phases, the *Solution Construction Phase* and the *Structures Update Phase*. We will explain these phases in more detail in the next paragraphs.

As we can see from the Figure 4.4, at each source node v_i in the system, artificial ants are sent to explore paths to the appropriate destination nodes v_d. Initially, system wide information is loaded in the execution environment of each node v_i (see Figure 4.4, Line 8). This information includes loading the list of target destinations and the initial value of the pheromone matrix. This system-wide information is represented by variable $\mathcal{S}$ in the algorithm. Next, each node v_u enters into a loop sending ants at regular intervals to explore different paths to reach a node v_v from itself (i.e.

node v_u). This is shown in lines 9-16 in the Figure 4.4. We shall now focus on the solution construction phase.

4.5.3 Solution Construction Phase

In this phase, we expect to generate and send ants at regular intervals Δt from each node v_u towards each destination node v_v (see lines 10-13 from Figure 4.4). The purpose of these ants is to discover a feasible low cost path based on the established metrics and to explore the current quality of such path. The idea is to find the status of the data sources, and discover possible sites in which the data can be processed along the way. Forward ants follow the same path as regular query processing requests, and they can suffer the same problems that a regular query might suffer. When an ant is located at a node v_u, it must decide what is the next node v_w to visit in its journey towards node v_v, where is the next step to the ant in the algorithm (see line 22 from Figure 4.4).

This selection is random but the neighbors with very strong pheromone trails have higher probability of being chosen since these are more likely to lead to the best possibility. The pheromone value that is inspected to make this decision represents a probability for choosing w as the next step in the ant's journey.

During their forward travel, the ants build a memory of their paths and the quality of the nodes in that path. This information is kept in a in-memory stack $Stack_{u \to v}(w)$, so it can be recovered for use during the *update phase*.

The steps that are taken during the *Solution Construction Phase* can be summarized as follows:

1. At each node v_u, each forward ant traveling to node v_v, picks the next node v_w to move to, choosing among the neighbors it has not visited before (see line 22 from algorithm). The neighbor $v_w \in \mathcal{N}_i$ is chosen with probability $P_{uwv} = \tau_{uwv}$
2. If a cycle is found, it implies that the forward ant returned to an already visited

```
AntNet(t, t_end)
 1  ▷ t current time
 2  ▷ t_sim simulation end time
 3  ▷ u source node
 4  ▷ v destination node
 5  ▷ w current node
 6  for each(S ∈ System) ▷ In parallel
 7        do
 8           S ← InitServices();
 9           while (t ≤ t_sim)
10                 do
11                    ▷ In parallel
12                    if (time to lauch ant)
13                      then
14                            u ← Select Source();
15                            v ← Select Destination();
16                            Launch Forward Ant(u, v);

Launch Forward Ant(u, v);
18  for each(ActiveForwardAnt(u, w, v))
19        do
20           while (w ≠ v)
21                 do
22                    next ← Select Next(w, v);
23                    move(w, next);
24                    memory ← Memorize(next, cost);
25                    w ← next;
26           Launch Backward Ant(v, u, memory);

Launch Backward Ant(v, u, memory);
28  for each(ActiveBackwardAnt(u, w, v))
29        do
30           while (w ≠ u)
31                 do
32                    next ← POP Memory();
33                    move(w, next);
34                    w ← next;
35                    Update Pheromone Matrix(w, u, v);
36                    Update StatisticalModel(w, u, v);
```

Figure 4.4. Ant Colony algorithm for Data Source and processing Site Discovery

node, and the cycle is removed from memory. Some other rules for cycle removal can be implemented, for example, if the length of the cycle is greater than the complete path already visited, then the ant dies immediately. This is done to prevent this ant from adding information that is inconsistent due to the fact that it is trapped in a cycle.

3. When the destination node v_v is reached, the forward ant generates a backward ant, and transfers to the new backward ant the information about the source node v_u and all its in-memory data structures. This is shown in line 26 of the algorithm. After this step, the forward ant dies (i.e., disappears from the system).

4. The backward ants follows the same path as its matching forward ant, but in opposite direction. Also, they are not queued like normal queries and forward ants, but rather have separate queues with higher priorities, since they are now updating the nodes with path information previously collected by the forward ant.

4.5.4 Structures Update Phase

The backward ant follows the same path as its matching forward ant, but in the opposite direction. A backward ant updates the two major data structures in every visited node v_w: the site statistics $\mathcal{M}_w$ and the pheromone trail $\mathcal{T}_w$. When the cost value obtained by the forward ant for the subpath between node v_u and node v_w (where $v_w \in$ set of nodes visited by the forward ant from v_u) is statistically "good" then it is used to update the corresponding $\mathcal{M}$ and $\mathcal{T}$ of all nodes in such path. In contrast, when the cost of any subpath is not statistically "good" (larger than a specific maximum value δ), those values are not used. For example, a node might be congested for a short period of time due to some unusual situation, while the ant happens to visit the node at that moment. Obviously, a bad performance (or

freshness) value will result from the observation, but we do not wish to penalize the node because of this rare event. Notice, however if the path continues to be bad, then by not updating the path, the alternative paths may become stronger, since the pheromone trail in the bad path will "evaporate".

Figure 4.5 shows an example in which the forward ant with destination node v_3 moves along the path $v_1 \rightarrow v_2 \rightarrow v_3$ and arrives at node v_3. Then, it generates a backward ant from node v_3 to v_1, following the path $v_3 \rightarrow v_2 \rightarrow v_1$, as we can see from figure 4.5. At each node v_w, $w = 2, 1$, the backward ant uses the stack $S_{1\rightarrow 3}(v_w)$ to update the values $\mathcal{M}$ and $\mathcal{T}$. Specifically, at node v_2, the backward ant updates the statistics $\mathcal{M}_2\,(\mu_{23},\ \sigma^2_{23},\ \mathcal{B}_{23},\ \ \mathcal{F}_{23})$ and the pheromone trail $\mathcal{T}_2$ directly at position τ_{233} and indirectly all other τ_{2j3}, where $j \in \mathcal{N}_2$ and j can connect to node v_3. Notice that in the first case, the algorithms just updates the tuple $(\mu_{23},\ \sigma^2_{23},\ \mathcal{B}_{23},\ \mathcal{F}_{23})$ of statical model $\mathcal{M}_2$. In the second case, the entry τ_{233} of $\mathcal{T}_2$ is update directly, and all other τ_{2j3} are adjusted because of the normalization necessary to make the sum of all τ_{2j3} equal to 1. The same explanation applies for the case node v_1, in which the update is done to:

- $\mathcal{M}_1\,(\mu_{13},\ \sigma^2_{13},\ \mathcal{B}_{13},\ \mathcal{F}_{13})$ and the pheromone trail $\mathcal{T}_1$ directly at position τ_{123} and indirectly all other τ_{1j3}, where $j \in \mathcal{N}_1$, and j can connect to node v_3.
- $\mathcal{M}_1\,(\mu_{12},\ \sigma^2_{12},\ \mathcal{B}_{12},\ \mathcal{F}_{12})$ and the pheromone trail $\mathcal{T}_1$ directly at position τ_{122} and indirectly all other τ_{1j2}, where $j \in \mathcal{N}_1$, and j can connect to node v_2.

Next we explain the way the statistical mode $\mathcal{M}$ and the pheromone matrix $\mathcal{T}$ are updated for all nodes along the path (v_u, v_v) traversed by a particular ant. Let v_w be a node in this path, such that v_w is either an intermediate node or the destination node v_v. For each node v_w in the path (v_u, v_v), we consider every sub-path (v_w, v_v), and we update the statistical model $\mathcal{M}_w$ for every node v_w. To accomplish this, the estimated mean μ_{wv} and variance σ^2_{wv} are updated using the metodology that we will explain later, according to equations 5.9 and 5.10, as well as the best value observed

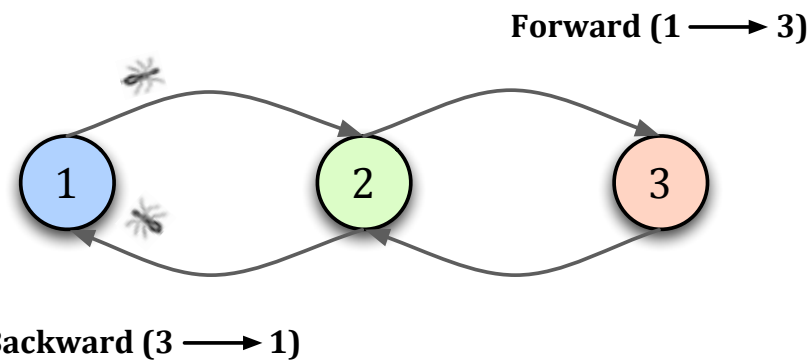


Figure 4.5. AntNet: Data Structure Update, adapted from Dorigo

$\mathcal{B}_{wv}$, with cost, and the freshness parameter $\mathcal{F}_{wv'}$. The statistical model $\mathcal{M}_w$ has to be able to capture the variability of the system and play a critical role in the updating process for the pheromone matrix $\mathcal{T}_w$, as explained next.

The pheromone matrix $\mathcal{T}_w$ is updated be incrementing the pheromone τ_{wfv}, where f is the next node chosen to be visited in the journey to reach v_v. This action will indirectly decrement the pheromone $\tau_{wkv},\ k \in \mathcal{N}_w, k \neq f$, because of the normalization required for the pheromone values to achieve a sum of 1. Let $o_{u \to v}$ be the cost that the forward ant found for the path (v_u, v_v). This cost can represent performance, freshness or some other measure of quality. This cost $o_{w \to v}$ represents the only available explicit feedback signal to score the path (v_w, v_v). Also note that the cost $o_{w \to v}$ cannot be associated with an exact error measured, given that the "optimal" trip time is unknown, because it depends on the whole network load status. For example, if the system is overloaded, all the costs $o_{w \to v}$ found in the trips will score poorly compared to the values observed when the system is under a low usage situation. Thus, paths that were previously bad now become good and vice-versa. This is the basis of our adaptive approach, because the algorithm is able to detect the current system behavior and adapt the statistics accordingly. Notice, also that under a given load condition and a client specifications, a path P_1 with lower cost than a path P_2 will be scored higher.

The cost $o_{w \to v'}$ is used to compute a pheromone reinforcement signal r, that

modifies the current pheromone value. The reinforcement r is a function of $o_{w \to v'}$ and $\mathcal{M}_w$, and its values always lie in the interval $[0, 1]$. An r value close to 1 indicates that the path is very good with respect to other previously found paths and it gets reinforced by a good amount (up to 1). In contrast, a value close to 0 indicates that the path is not very good and the reinforcement of the path is very low, thus attempting to make it less attractive. We discuss the exact definition of r in the next Section.

The value r is used by the backward ant moving from node v_f to node v_w to increase the pheromone values $\tau_{wfv'}$. The pheromone $\tau_{wfv'}$ is increased by amount r as follows [23]:

$$\tau_{wfv'} \leftarrow \tau_{wfv'} + r \cdot (1 - \tau_{wfv'}) \tag{4.2}$$

Notice that, given the same value r, small pheromones values $\tau_{wfv'}$ are increased proportionally more than large pheromones values, favoring in this way a quick exploration of the newly discovered paths that are good. Pheromones $\tau_{wkv'}$ for destination v' of the other neighbor nodes $k, k \in \mathcal{N}_w, k \neq f$, evaporate implicitly by normalization as we show in equation 4.3. That is, their values are reduced so that the sum of pheromones on services existing from node v_w will remain 1.

$$\tau_{wkv'} \leftarrow \tau_{wkv'} - r \cdot \tau_{jwkv'} \tag{4.3}$$

It is important to remark that every discovered path $(v_w, v_{v'})$ increases its selection probability. Additionally, the use of the *differential path effect* (ants over shortest trip reinforces the system more frequently) permits that good paths receive either high reinforcements, independent of their frequency, or low and frequent reinforcements. In fact, for any load condition, a path receives one or more high reinforcements only if it is much better than previously explored paths.

In the next example, we illustrate the pheromone update and implicit evaporation process. Consider a middleware configuration with four nodes v_1, v_2 , v_3 and v_4, as shown in Figure 4.6. Suppose that one ant was sent from node v_4 to v_1 and it chose the path (v_1, v_4) and, then when the backward ant returned to v_1, after calculations using $\mathcal{M}_1$, obtained that $r = 0.35$, and using this value, we update pheromone trail was updated for neighbor v_1, and at the same path another neighbor suffer the evaporation process, in this case v_2 (using Equations 4.2 and 4.3), as we show in Table 4.1.

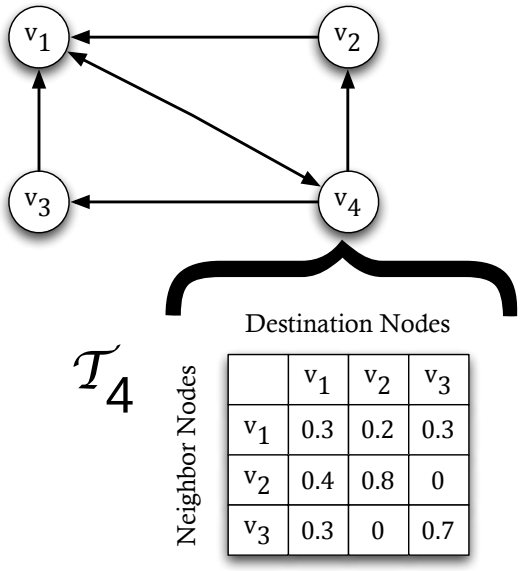


Figure 4.6. Example of the Pheromone Trail Update

4.5.5 Calculation of reinforcement value r

The calculation of the value r is very important in the system. It must be assigned after considering three main aspects:

- The aath should receive an increment in their selection probability proportional to their goodness.
- The goodness is a relative measure, which depends on the load conditions than

Table 4.1. Calculations for Pheromone Update and Implicit Evaporation Process

	Pheromone Values for Destination v_1	
Neighbor	τ_{before}	τ_{after}
v_1	0.3	$0.3 + 0.35 * (1 - 0.3)$ 0.545
v_2	0.4	$0.4 - (0.35 * 0.4)$ 0.260
v_3	0.3	$0.3 - (0.35 * 0.3)$ 0.195

can be estimated using the statistical models $\mathcal{M}$.

- It is important not to follow all the traffic fluctuations.

That means that is important to have a good trade-off between stability and adaptivity. Several methods can be used to assign the r value, trying to consider the above three requirements:

- *r as a constant.* The simplest way is to set r as constant. Independent of the result of the trip, the all discovered path are rewarded in the same way. In this simple but meaningful case the core of the algorithm is based on the "real" ants behavior to discover shortest paths via *stigmergic* communication mediated by pheromone trials. In other words, here we use the *differential path effect*. The obvious problem with this approach lies in the fact that, although ants following longer paths arrive much later than those following shorter paths, they nevertheless have the same effect on the pheromone matrices as the ants that found the shorter path.

- *r as a discrete function.* In this case, we set different values for the reinforcement value r according the goodness of the path based on a confidence interval $[I_{inf}, I_{sup}]$ for $\mu_{wv'}$ and $\sigma_{wv'}$ that we store in $\mathcal{M}_w$. Following the Dorigo's approach [23], we use the Chebyshev inequality that allows the definition of a

confidence interval for a random variable following any distribution [51]. I_{inf} is set to $\mathcal{B}_{wv'}$ and we calculate I_{sup} as follows:

$$I_{sup} = \hat{\mu}_{wv'} + \left(\frac{1}{\sqrt{(1-\nu)}} \left(\frac{\hat{\sigma}_{wv'}}{\sqrt{n}} \right) \right) \tag{4.4}$$

where,

ν is a selected confidence level and

n is the size of the sample that we consider for the confidence interval

Usually, for specific probability densities, the Chebyshev bound is not very tight, here its use is justified by the fact that only a raw estimate of the confidence interval is required and that in this way there is no need to make any assumptions on the distribution of any μ.

The reinforcement value can be set using the values that we present in equation 4.5. It is important to mention that this reinforcement values can be modified according with the experience and the simulation results.

$$r = \begin{cases} 0.75,\ If\ o_{w \to v'} < I_{inf} \\ 0.10,\ If\ I_{inf} \leq o_{w \to v'} \leq \mu_{wv'} \\ 0.25,\ If\ \mu_{wv'} < o_{w \to v'} \leq I_{sup} \\ 0,\ Otherwise \end{cases} \tag{4.5}$$

Also notice that when the value of $o_{w \to v'}$ is greater than I_{sup}, we set the value to zero to reinforcement r and in this form, we discard this value for the updating process of the pheromone matrix.

- *Dorigo approach.* In this case we can use a more elaborate approach proposed by Dorigo [23] using the same interval confidence definition that we presented before. The following functional form gave good results in his simulation work:

$$r = c_1 \left(\frac{\mathcal{B}_{wv'}}{o_{w \to v'}} \right) + c_2 \left(\frac{I_{sup} - I_{inf}}{(I_{sup} - I_{inf}) + (o_{w \to v'} - I_{inf})} \right) \tag{4.6}$$

where

- c_1 and c_2 are parameters which weigh the importance of each term
- The first term represent simply evaluates the ratio between the best trip cost observed store in the system
- The second term evaluate how far the value $o_{w \to v'}$ is from I_{inf} in relation with the extension of the confidence interval, that is, considering the stability in the latest trip times. Note that the denominator of this term could go to zero, when $o_{w \to v'} = I_{inf} = I_{sup}$. In this case the whole term is set to zero.

During the execution of this project we explored all options and we determined that we obtain the better results using Dorigo approach.

CHAPTER 5

Autonomic Ranking of Data Sources and Query Processing Sites using Ant Colony Theory

5.1 Overview

This Chapter presents the framework used by AntFinder for characterizing sites based on performance. We explain why this is an important issue in the framework. Additionally we present the metrics and explain how it works over the structures. Finally, we discuss experiments carried out on an implementation of AntFinder in Java, that was deployed on a simulation built with CSIM for Java.

5.2 Introduction

Wide-area networks pose many challenges to developers of database middleware solutions for data integration. Very often, data sources have stale or incomplete data, limited query processing capabilities or restricted access privileges. Likewise, query processing nodes might become slow because they get overloaded with query requests. Thus, the cost of processing a given query Q is often hard to predict. If we add

into this mix a few mobile sites that serve data or process queries, then this unpredictability increases since these nodes might change network connectivity, go offline, or decrease their performance to save battery life. In this scenario, it becomes very difficult to establish a reliable method to estimate the cost of running a query in the system. A great deal of effort has been devoted to the problem of optimizing queries in local and distributed databases [46, 41, 33]. Most of these approaches assume that a query optimizer will explore a search space of query plans until it finds one with minimal cost, defined as either response time, resource usage, power costs, or network cost, among many others.

This optimization process, however, cannot be effective unless the middleware system has accurate information about the performance characteristics of the data sources and query processing sites. Thus, given a query Q, it is necessary to determine which sites can provide the data to satisfy the query and which sites can provide the computing power to generate the query results. The query optimizer for the middleware system can then generate an efficient query plan to answer the query. Typically, query optimizers for middleware systems have relied on a catalog system that has such information. This is best illustrated in Figure 5.1. As we can see, a query submitted by a client is first parsed and the catalog is used to validate it. Then, rewrite rules are applied to simplify and generate a better representation of the query. Next, the query optimizer can apply its search techniques and optimization rules to explore the space of candidate plans and find an optimal plan. The catalog is used to obtain information about the data, performance characteristics and query execution capabilities of candidates execution sites in the system. This makes the catalog a key ingredient for the completion of these three initial phases in the processing of a query. Once an optimal plan has been found, an executable version of the plan is generated and sent to the target remote query execution sites. These sites generate the query results and send them to the client application.

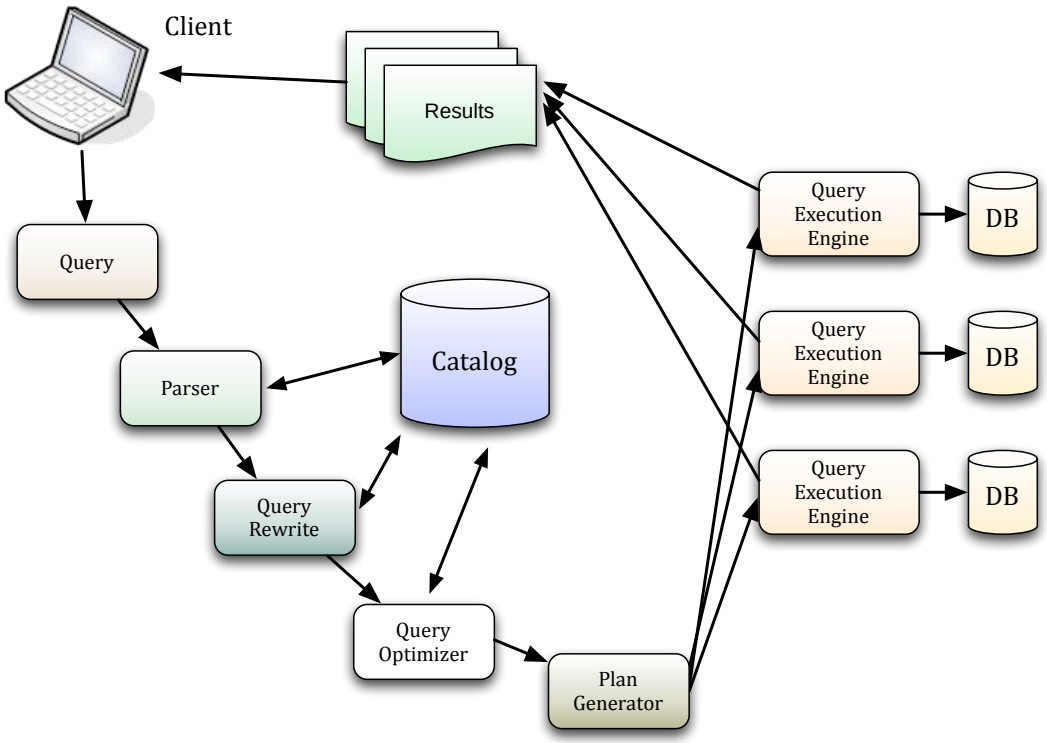


Figure 5.1. Query Processing Cycle

Clearly, the job done by the query optimizer will be only as good as the metadata it uses to guide the optimization process. Yet, most systems assume that a system administrator will manage the catalog and provide adequate metadata to populate it. Relatively little attention has been paid to the fact that the metadata in the catalog is quite dynamic, changing as new mobile sites enter (or leave) the system, existing data sources are updated, network connectivity changes, and query processing sites become loaded with requests. The ad hoc and distributed nature of the problem makes it unfeasible to have a solution that relies on system administrators to periodically update and publish system metadata throughout the system. Likewise, trying to discover the information at query time can only slow down query execution and reduce system throughput.

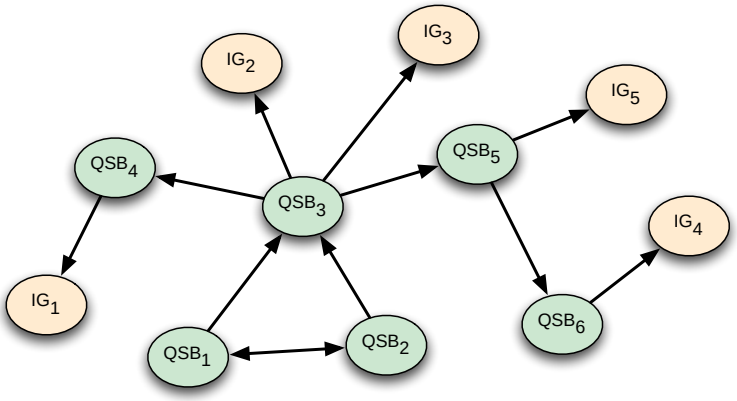
In this Chaper we propose an autonomic framework for *continuously discovering and ranking the sites in a database middleware system for mobile, wide-area*

environments. Our framework, called AntFinder, can help keep the catalog updated, and feed accurate information to a query optimizer to make better decisions for query operator ordering and placement. Our proposed solution addresses this problem by proving a methodology to a) discover the performance characteristics of the data sources and query processing sites, and b) rank these sites based on their *quality.* In AntFinder, each site in the system participates in a social network to continuously share system metadata, discover new sites, and rank the capabilities of known sites. This social network is built by employing Ant Colony Theory, and each site exposes its metadata to be inspected by autonomous software robots called artificial ants [23]. These ants use the information that is found to establish a ranking of the sites, giving a notion of the quality of a site u as seen by other site v. The goal of our method is to give the optimizer accurate information about a series of paths leading from a site v to one or more data sources. These paths include query sites that can be used to process de data. The optimizer can use these paths as interesting candidate query sub-plans to be considered for building the final plan.

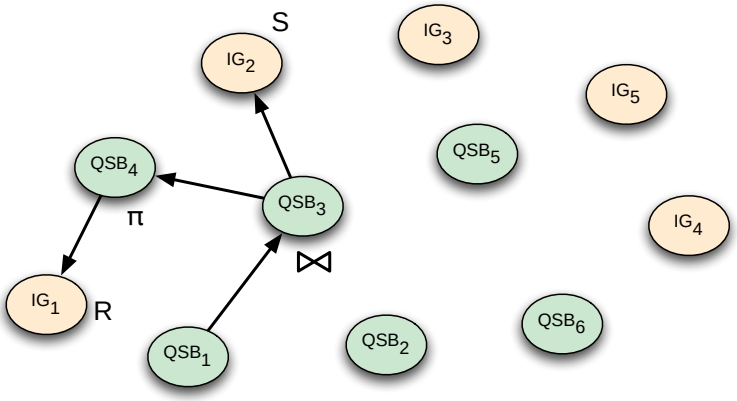
5.3 Motivation

Since there are many possible sites to supply data or computing power in our Middleware System, the optimizer would need to explore several alternatives and chose the one with lower costs. If we model the system as a graph G, each alternative plan is a tree T which happens to be a subgraph of G, $T \subseteq G$. In our model, G is a connected, directed graph $G = (V, E)$, where V is a collection of services (every $v_u \in V$ is a service that can be a QSB, or an IG) and E is the set of available communications links between pairs of services. For every edge $(v_u, v_v) \in E$, we have a weight $w(u, v)$ specifying the "cost" to connect v_u and v_v. In this Chapter we shall use (u, v) and (v_u, v_v) interchangeably to help keep the equations readable. The cost $w(u, v)$ represents the effort incurred or benefit received when results are first produced in v_v,

then sent from site v_v to site v_u and finally consumed at site v_u. Figure 5.2a shows a group of $QSBs$ and IGs interconnected under this scheme. Figure 5.2b shows the imposition of a query execution tree on a set of $QSBs$ and IGs. The plan represents a possible solution for the query $\sigma(\pi(R) \bowtie S)$, where R and S are two relations available in the system. The root of the tree is located on node QSB_1, while the leaf nodes are IG_1 and IG_2. The join operation is run at node QSB_3 and the projection operation is run at node QSB_4. We shall now turn our attention to the method by which we shall estimate the cost of each $w(u, v)$ in G.



(a) System Representation



(b) Execution Tree

Figure 5.2. Graph Representation.

5.3.1 Performance Definition

The performance metric is used to charactize the quality of a given site based on the raw computational capabilities available to process operators in the query plan. This concept is also called **Quality of Service**, QoS, because it attempts to provide a notion on how fast or efficient can the system be when executing query operators in a query plan. QoS can be defined in terms of resource usage as a way to indicate how efficient a plan is in minimizing the amount of resources (e.g., disk, memory, network bandwidth, battery) that must be spent solving a query. Alternatively, QoS can be defined in terms of response time, which indicates the total time elapsed to produce the query results. In this case, the shorter the response time the better the plan is.

We can consider different approaches and parameters to estimate QoS for a given plan, broadly classifying in terms of local and network resources :

- **Local resources**: For a query operator this is the time that we expect to spent to complete the processing of tuples using the resources on a given host. We can consider:
  - Execution time: CPU processing time.
  - Input/Output time (disk time or tape time): Time to retrieve the necessary information from the disk to resolve a query.
- **Network resources**: For a query operator, this is the time that we expect to spent fetching tuples from the producer site. Different parameters can be use:
  - Network traffic
  - Bandwidth
  - Latency

In the optimization process for a query, sometimes local resources can be omitted if the network costs far exceed local costs. However, recent advances in CPU and network technologies make it necessary to consider both elements in the systems.

5.3.2 Forecast Model for Performance

We now present several alternative models to estimate the performance metric of the system. Since we are measuring this performance in terms of time, either response time or resource usage time, we assume this metric to be a continuous variable. In the simplest model, we can use a fixed value as the performance estimator cost $\hat{p}$ for every edge (u, v) present in our system. We can use a value based in experience about the system, as we see in the following equation.

$$\hat{p}(u,v)_t = v_{u,v} \tag{5.1}$$

where $v_{u,v}$ is the performance constant cost between services u and v, valid for time period t. In this case, we use a static model, and then the changes in the system do not be considerate.

If we need to have a better estimator of the performance of the system, including the dynamic changes in the topology, we could include past information so we can infer future behavior with more accuracy. If we just want to use a single previous observation, we may estimate the performance using the following structure: The first time that the cost of a service needs to be determined we use a constant value $v_{u,v}$ as the estimator. This constant value is obtained from either past experience or some calibration of the system. After that, we use as estimator of the service cost the last value measured for an operation in this node. This scheme is summarized in the following equation:

$$\hat{p}(u,v)_t = \begin{cases} v_{u,v}, & t = 0 \\ p(u,v)_{t-1}, & t > 0 \end{cases} \tag{5.2}$$

where $p(u,v)_{t-1}$ is the previously observed value.

We can include far more information in the calculation of our estimators.

In fact, we can use some short-range time series forecast models such as *Moving Average*, *Weighted Moving Average* or *Exponential Smoothing*, as we shall explain next. Clearly, our assumption here is that our performance metric estimator can be modeled using short-range time series. However, if our performance evaluation show that these inference methods are inaccurate, we can try to implement more complex models that include cycles, trends and seasonality of our metric, using time series theory.

5.3.2.1 Moving Averages

In this model, we define the forecast cost for the next period of time as the arithmetic average of a specific past range of measured values. For example, if we choose as our past range the three previous periods (or measures over three past observations), we can mathematically express the model as follows:

$$\hat{p}(u,v)_t = \begin{cases} v_{u,v}, & t \leq 3 \\ \frac{\sum_{k=(t-3)}^{(t-1)} p(u,v)_k}{3}, & t > 3 \end{cases} \tag{5.3}$$

As we can see from the previous equation, the estimator is computed as a constant value $v_{u,v}$ for the first three requests. The reason for this is that there are not enough measured values to compute an average. After that, the estimator is computed based on the previous three measured values for the given performance metric. In general, if wish to use this method to estimate a performance metric based on n previous measures, the equation becomes:

$$\hat{p}(u,v)_t = \begin{cases} v_{u,v}, & t \leq n \\ \frac{\sum_{k=(t-n)}^{(t-1)} p(u,v)_k}{n}, & t > n \end{cases} \tag{5.4}$$

5.3.2.2 Weighted Moving Average

This model is similar to the moving average model described before, but instead of using an arithmetic average of the past n performance parameters, it uses a *weighted* average of the past n performance parameters. Typically, more weight is placed on the most recent time periods. For example, if we choose as our past range the three previous periods (or measures over three past observations), we can mathematically express the model as follows:

$$\hat{p}(u,v)_t = \begin{cases} v_{u,v}, & t \leq 3 \\ \alpha_1 p(u,v)_{t-1} + \alpha_2 p(u,v)_{t-2} + \alpha_3 p(u,v)_{t-3}, & t > 3 \end{cases} \tag{5.5}$$

Here α_t is the weigth for the period t that we choose. Also notice that the weight of $\alpha 1 > \alpha 2$ and $\alpha 2 > \alpha 3$. In general, if we wish to use this method with n previous observations, we have:

$$\hat{p}(u,v)_t = \begin{cases} v_{u,v}, & t \leq n \\ \alpha_1 p(u,v)_{t-1} + \alpha_2 p(u,v)_{t-2} + ... + \alpha_n p(u,v)_{t-n}, & t > n \end{cases} \tag{5.6}$$

where, $\alpha_m < \alpha_{m-1} \forall m \ni 2 \leq m \leq n$. Also, it holds that $\sum_{k=1}^{n} \alpha_k = 1$ for $n > 1$.

5.3.2.3 Exponential Smoothing

The third model that we present is also a short-range time series forecasting model used to estimate a performance metric for the next time period. In this model, the estimated performance metric for a period t is obtained by combining the estimated value of the previous period $t-1$ with a correcting error term. This latter term is obtained by subtracting the actual measured performance metric at time $t-1$ with the estimated value at $t-1$. This error is multiplied by a constant α to weight in

the relevance in this error term. Mathematically, we can express this model by the following recurrence:

$$\hat{p}(u,v)_t = \hat{p}(u,v)_{t-1} + \alpha \left(p(u,j)_{t-1} - \hat{p}(u,v)_{t-1} \right) \tag{5.7}$$

which also can be expressed with the alternative recurrence:

$$\hat{p}(u,v)_t = \alpha p(u,v)_{t-1} + (1-\alpha)\hat{p}(u,v)_{t-1} \tag{5.8}$$

where:

- $\hat{p}(u,v)_t$ is the performance metric estimator for period t (the current period)
- $\hat{p}(u,v)_{t-1}$ is the performance metric estimator for period $t-1$ (the previous period)
- $p(u,v)_{t-1}$ is the actual performance metric measured in the system por period $t-1$ (the previous period)
- α is the smoothing constant, from 0 to 1. Good values for this constant must be obtained tuning the system.

5.3.3 Using the Forescast Model in the Update Phase

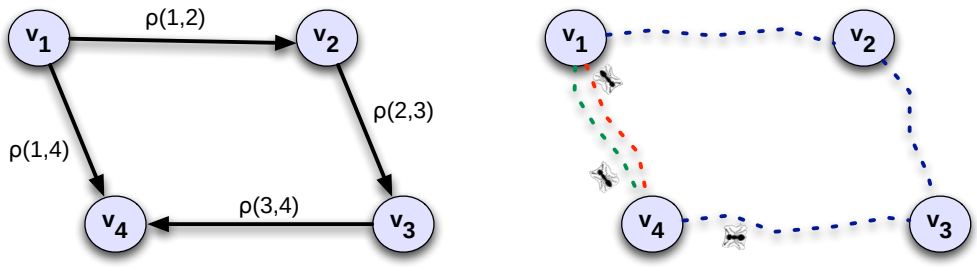
Given a node v_u, then for each destination node v_v in the system, we need to have an estimate of the cost of the path connecting v_u and v_v. Since there might be many possible paths, it is important to track not only the cost and structure of the best path from v_u to v_v but also a measure of the average cost of all paths connecting v_u and v_v. This latter statistic will be used in the algorithm to update the pheromone trail and measure the variability in the system. Notice that the ants are constantly being sent to explore the system. Thus, we need to have a mean estimator for cost of paths connecting each node v_u to a destination node v_v. Let us denote this sample mean between nodes v_u and v_v by $\hat{\mu}_{uv}$. Likewise, let us denote the variance of this

sample mean by $\hat{\sigma}^2_{uv}$. Then, the estimated cost mean $\hat{\mu}_{uv}$ and variance $\hat{\sigma}^2_{uv}$ give a representation of the expected cost from node v_u to node v_v and the stability of such cost.

$$\hat{\mu}_{uv} \leftarrow \hat{\mu}_{uv} + \alpha \left(o_{u \to v} - \hat{\mu}_{uv}\right), \tag{5.9}$$

$$\hat{\sigma}^2_{uv} \leftarrow \hat{\sigma}^2_{uv} + \alpha \left(\left(o_{u \to v} - \hat{\mu}_{uv}\right)^2 - \hat{\sigma}^2_{uv}\right), \tag{5.10}$$

Where: $o_{u \to v}$ is the new observed (measured) cost from node v_u to destination v_v. Thus, the value $\hat{\mu}_{uv}$ is updated based on its current value and the error observed from the current measured cost $o_{u \to v}$. This error is contained in the term $\alpha \left(o_{u \to v} - \hat{\mu}_{uv}\right)$. The factor α weights how much the number of most recent samples will affect the average. The same scheme applies to the update of the variance $\hat{\sigma}^2_{uv}$ [16, 23, 37]. In the next example, we illustrate these concepts. Consider a middleware configuration with four nodes v_1, v_2, v_3 and v_4, as shown in Figure 5.3a. Suppose that we are interested in paths connecting v_1 and v_4. From Figure 5.3a, you can observe that the ants could discover two different paths connecting source node v_1 and the destination node v_4.



(a) Initial Conditions (b) Ants Behavior

Figure 5.3. Example of the calculations for $\mu_{1,4}$ and $\sigma^2_{1,4}$

Now suppose we launch tree ants at different moments of time, represented

Table 5.1. Results for ants explorations in figure

Path Color	Explored Path	Associated Cost
green	(v_1, v_4)	3.2
red	(v_1, v_2, v_3, v_4)	4.8
blue	(v_1, v_4)	3.6

by different colors in figure 5.3b. As we show in Table 5.1, in the first attempt, represented by a green line, the ant found the path (v_1, v_4) and stores the cost during the trip (we could think of it as the performance cost associated with path or as a cost associated with freshness) and the nodes visited during the trip. After that, the ant chooses another path using a stochastic decision process, and this path is represented by the red line. In this case, the path was (v_1, v_2, v_3, v_4). Finally, the last ant chooses again the path (v_1, v_4) represented by blue line.

Based on this scenario, each ant travels backwards from destination node v_d to source node v_i along the path just explored. Each step of the way, the ant will update the pheromone and other state statistics (to be described shortly) contained in each node of path. For now we just illustrate the calculation of the mean $\hat{\mu}_{1,4}$ and variance $\hat{\sigma}^2_{1,4}$ over the path with source v_1 and with destination v_4 (the ants update the information for all the subpaths too). The first time, the system does not have any statistics, then, for the mean $\hat{\mu}_{1,4}$ the system assigns the value that the ant stores in the trip, and for variance $\hat{\sigma}^2_{1,4}$, according with the experience it assigns a constant, we will use $\hat{\sigma}^2_{1,4} = 1$ (since it is not possible calculate the variance with one value). Then, when the second and third ants go back, they use the information already collected, the weight factor α (for this case we will use $\alpha = 0.4$) and the stored information in the statistics module to calculate the new mean $\hat{\mu}_{1,4}$ and sigma $\hat{\sigma}^2_{1,4}$, as we show in theTable 5.2.

Table 5.2. Calculations for $\mu_{1,4}$ and $\sigma^2_{1,4}$

Path Color	*Explored Path*	*Cost*	$\mu_{1,4}$	$\sigma^2_{1,4}$
green	(v_1, v_4)	3.2	3.2	1
blue	(v_1, v_2, v_3, v_4)	4.8	3.2 $+0.4(4.8-3.2)$ $=3.84$	$1+$ $0.4((4.8-3.2)^2-1)$ $=1.62$
red	(v_1, v_4)	3.6	3.84 $+0.4(3.6-3.84)$ $=3.74$	$1.62+$ $0.4((3.6-3.74)^2-1.62)$ $=.99$

5.3.4 Java CSIM Simulation

We have implemented a detailed Java CSIM simulation of AntFinder based on the NetTraveler system architecture. Each QSB and IG site is an object server called the *ant host*, which serves as a hosting environment to receive ants. Each ant is an object that can be created, migrated or destroyed. Movement of an ant between nodes is implemented by having the ant create a copy of itself at the next node to be visited, and then transferring all its internal variable to this copy. The site statistics $\mathcal{M}_u$ and the pheromone trail $\mathcal{T}_u$ are implemented using adjacency lists, and the paths between nodes are implemented with linked lists. These structures are backed by database tables, which are accessed via Hibernate. For evaluation purposes, we choose Pareto Bounded as a probability distribution to simulate the service time at any node and to the travel time, since in [40] and [65] presented this option as better than exponential o regular Pareto. By other other hand, we sent the ants using the exponential distribution since it has been shown to successfully model the traffic over the network [40, 65, 53].

5.3.5 Feasibility

The first question that we must answer is whether our approach can actually find optimal or near optimal solutions to the problem of finding a path between a QSB v_u and and another QSB or IG v_v. To accomplish this we built several configurations

of NetTraveler nodes in our Java CSIM simulation. We developed different models for our database middleware system, each one having a different number of servers and connectivities between these. For example, one model had sixteen servers, the second had twenty two servers, the third one had thirty four servers, the fourth one had sixty servers, and the fifth one had five hundred servers. We will focus on the model with thirty four servers, which had twenty six QSBs and nine IGs as show in Figure 5.4.

The performance cost metric used was the cost of processing a page with 4KB worth of tuples, and then send these across the network. Intermediate nodes in the path acted as relays, but in a more complex setting these nodes would have query processing operators that must be applied to the page of tuples. Notice that the path forms a execution pipeline for tuples from an IG to a QSB. We used Exponential Smoothing as the mechanism to estimate costs and update both the pheromone matrix and the statistical model on each node in the system since has less storage necessities and shows better results during the setup process. Each node in the network was a modeled as a host with 1GB of RAM, 3GHZ CPU and 100 GB of disk. From each QSB we sent ants to discover how to reach IGs in the system. We sent ants concurrently, and we ran trials in which each QSB would sent 20, 40 and 80 ants in each trial, at random order. We repeated each trial 10 times and then took averages on the following percentages: a) success in finding the best path, b) consistency of pheromone trail to lead to the shortest path, c) number of routes in the system that were explored, d) amount of ants that were lost in loops and e) average amount of ants served by node.

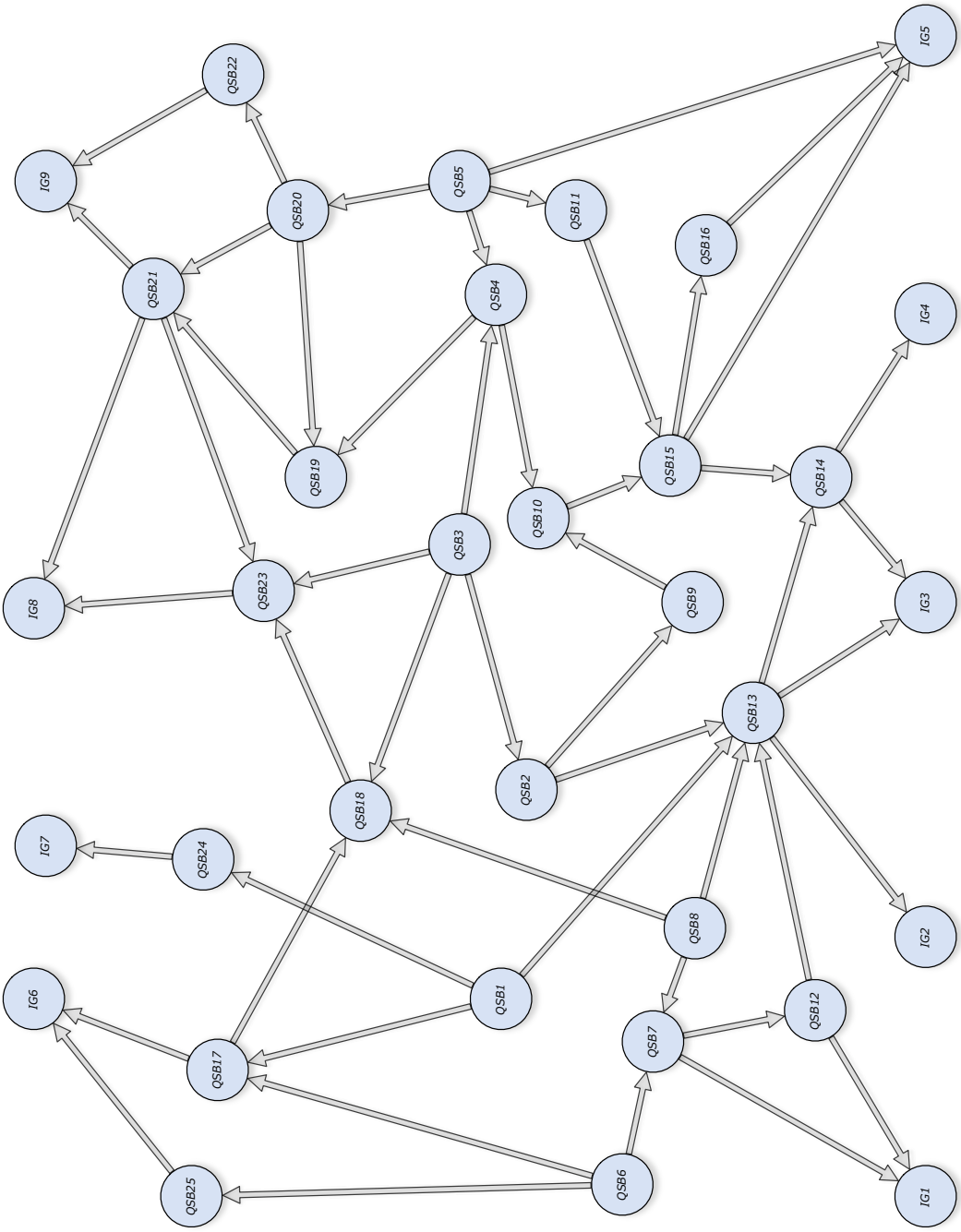


Figure 5.4. Example Model

Table 5.3. General Experiments Results (Part 1)

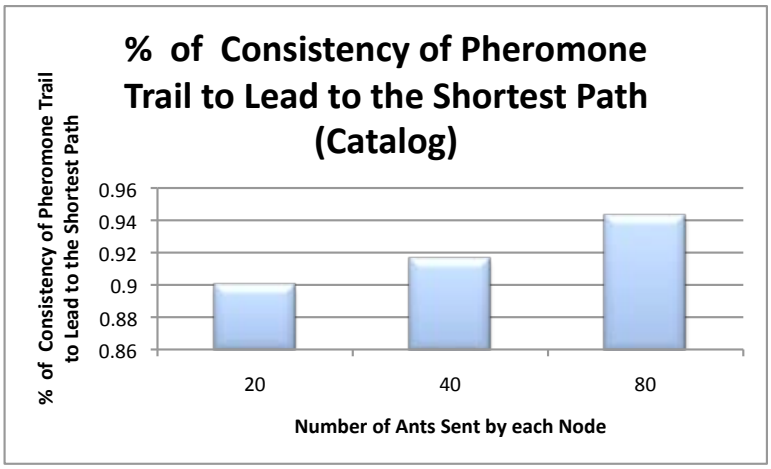
Ants	% Pheromone Consistency	% Success in Finding Best Paths	% Success in Finding Paths with Similar Cost
20	89.9	84.0	89.8
40	91.5	90.2	96.8
80	9.43	92.9	98.8

Table 5.4. General Experiments Results (Part 2)

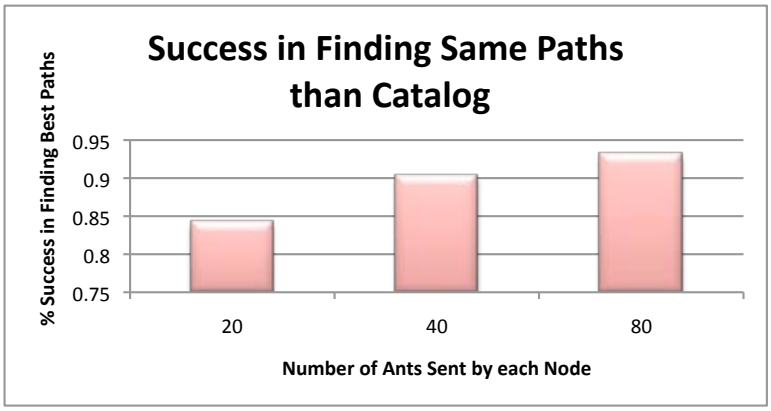
Ants	Average Number of Ants Visiting each Node	Percentage of Explored Routes
20	47	85.3
40	94	90.4
80	186	93.3

Tables 5.3 and 5.4 and Figures 5.5 and 5.6 show the general results of the experiments. As we can see from the figure, AntFinder can find the best path over 89% of time, and the pheromone trails are over 85% of time consistent with the shortest path. Notice that the pheromone consistency increases with the number of ants, since each node is accumulating more samples and hence its accuracy increases. Also, the percentage of the network that is explored improve as the number of ants increases. Notice also, that the average number of ants visiting each node increases, with the ants sent, as expected. In Chapter 7 we will discuss a improvement method to launch the ants.

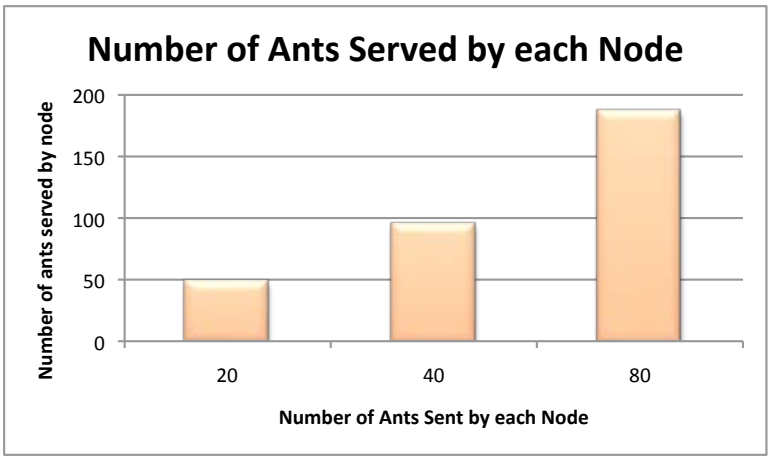
We chose to send a constant quantity of ants per node, regardless the number of destinations that can reach each one. For this reason, the results of nodes with fewer destinations as QSB_8 (see Figure 5.7a and 5.8a) are better than for the nodes that access more destinations as QSB_6 (see Figures 5.7b and 5.8b). In Chapter 7 we will discuss an improvement method to launch the ants, and in this way resolving the issue.



(a)

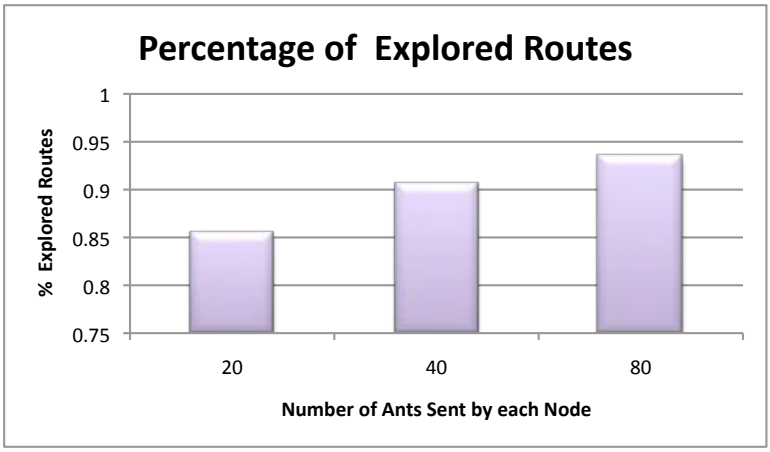


(b)

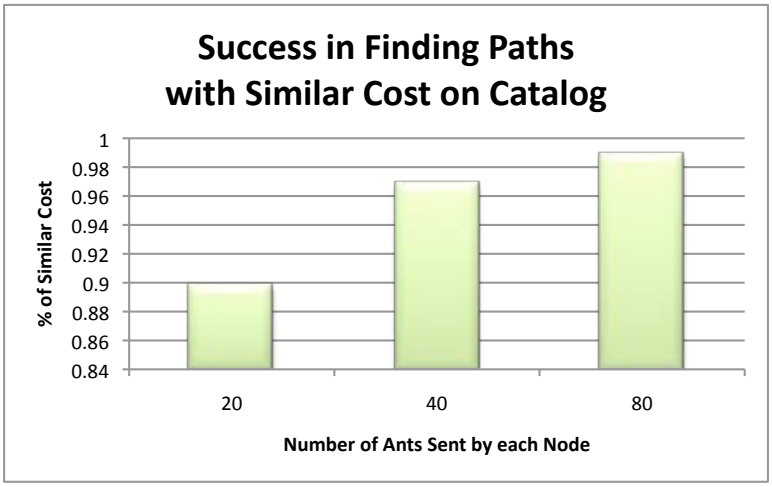


(c)

Figure 5.5. General Results (part 1).

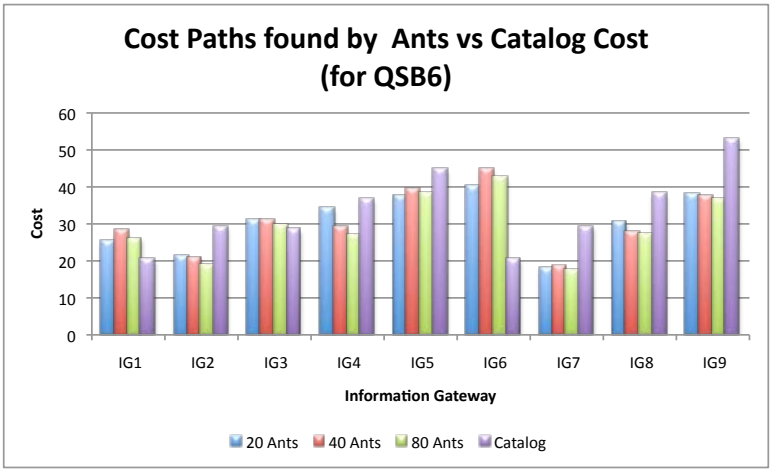


(a)

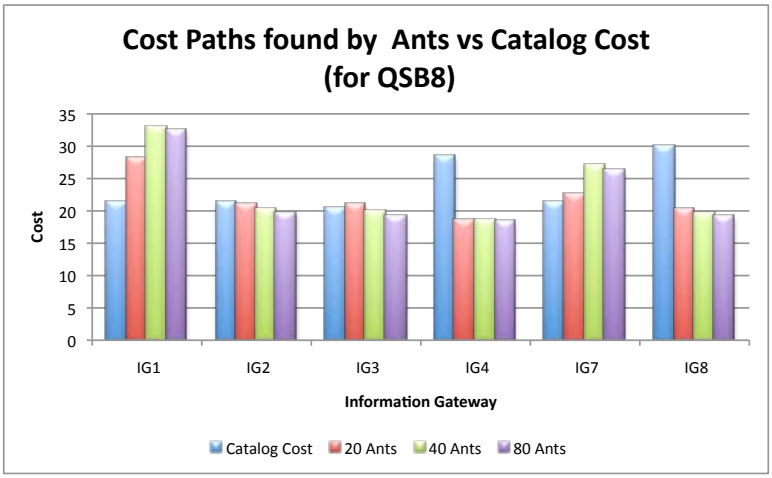


(b)

Figure 5.6. General Results (part 2).

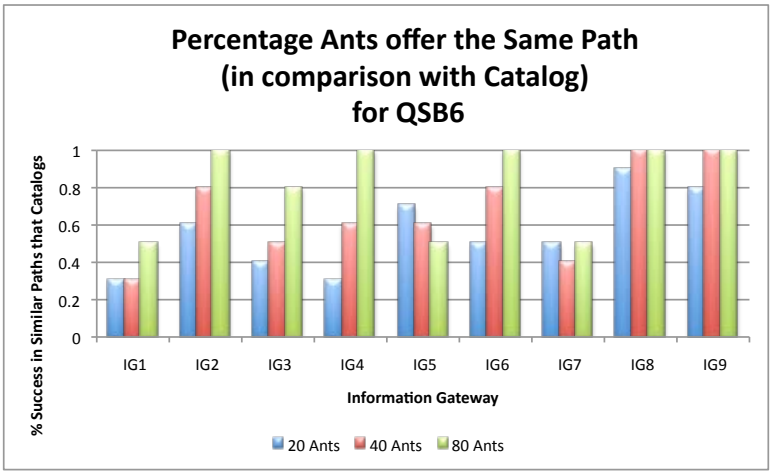


(a) QSB_6 Results

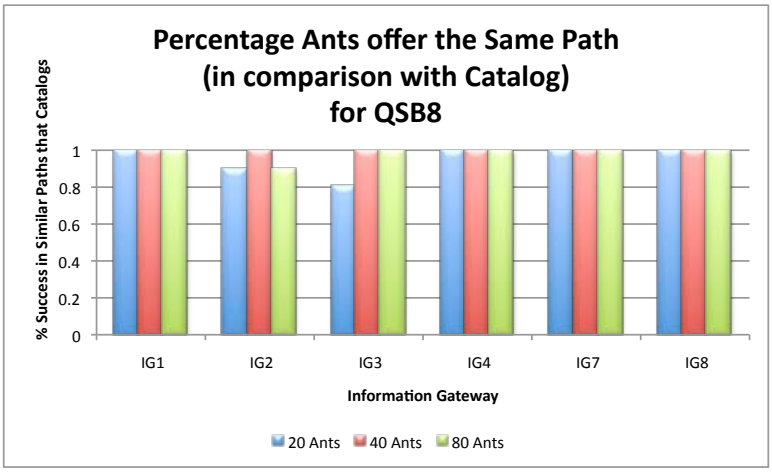


(b) QSB_8 Results

Figure 5.7. Cost Paths found by Ants vs Catalog Cost.



(a) QSB_6 Results



(b) QSB_8 Results

Figure 5.8. Percentage Ants offer the Same Path as Catalog.

5.3.6 Efficiency

The next question that must be raised is how efficient our approach is, when compared to the alternative of dynamically finding the paths at run time. For this purpose, using the same system than before, we compare our method with two variants based on Dijkstra's algorithm for shortest paths search. In the first variant, there is a central catalog with information about current cost of edges in the system, in our case, we use the theoretical expected value for a distribution as a estimator of the cost on the node and over the links. Then, having this catalog, each QSB could ask on-line for the each link cost of the system or for the average service time on every node. Once the statistics arrive, they run Dijkstra's algorithm to find the shortest path. In the second variant, the catalog is partitioned at each QSB. Each QSB could ask on line for the each link cost of the system to other node on the system or or for the average service time on every node (in parallel). Then, all the statistics are sent to the QSB, and this node will run Dijkstra's algorithm to find the shortest path. In our case, we just ask to Statistical Model about the best path for the desire destination, but this time, the calculation were done offline.

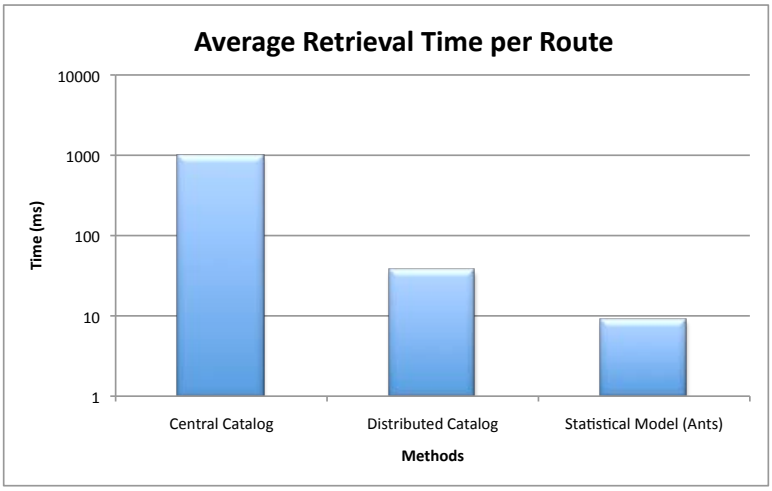
We ran experiments over the constructed system, and we obtained similar results overall. We present the results for the system used before, when we sent 80 ants by each node. We did the same with both variants that use Dijkstra's algorithm to find shortest paths at query time. We ran each experiment ten (10) times and we present average values from those runs. Figure 5.9a shows the general and specific result for one QSB (QSB_6). The figures show the average time to found the best path with AntFinder and using the variants based on Dijkstra's algorithm. The **Time** is the estimated number of milliseconds that are required to process a page of tuples of 4KB along the path. Note that the overhead incurred for both variants, is at least of an order of magnitude larger (for distributed catalog) for the variants based on

Dijkstra's algorithm. Figure 5.9b shows how the results on each node are consistency with the general results, having the same behavior and showing promissory results for our approach

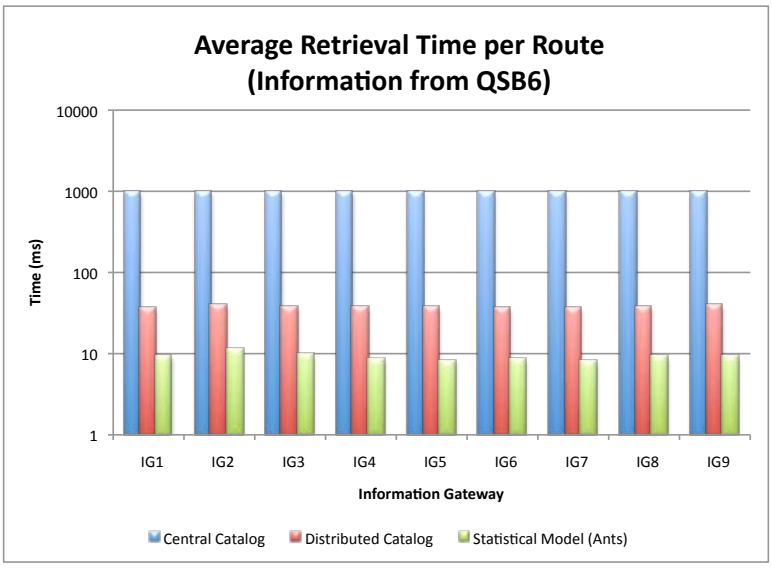
Although we just compared the first path offered by AntFinder with the variants based on Dijkstra's algorithm, our system may offer for the same source v_u more than one path to reach destination v_u instantaneously and neither a Centralized Catalog or Distributed Catalog could offer the same. Notice that since the ants work in parallel to the query execution process, a path lookup only required a local query in the catalog of the QSB. Thus, our approach is accurate with respect to path information and yet involves little extra overhead in the process. As the number of nodes increases, we can expect an increase in these overhead differences between our method and any method that finds paths at run time.

5.4 Summary

In this Chapter we proposed an autonomic framework for continuously discovering and ranking the sites in a database middleware system for mobile, wide-area environments. Our framework, called AntFinder, can help keep the catalog updated, and feed accurate information to a query optimizer about paths with sites for query operator placement. We proposed a framework for characterizing sites based on performance. Then, we presented an adaptation of the ACO algorithm to explore the system and rank the characteristic of each site. We used Exponential Smoothing as the mechanism to estimate costs and update both the pheromone matrix and the statistical model on each node in the system since has less storage necessities and shows better results during the setup process. The obtained values after a setup process is show in Appendix A. This framework is fully de-centralized and can scale to large number of nodes. Finally, we presented the results of a performance study carried out on a simulation of the system. These experiments show that our framework can find near



(a) General Results



(b) QSB_6 Results

Figure 5.9. Average Retrieval Time for All Methods

optimal paths in over 90% of the cases. In addition, we can say **that our method is far superior to approaches that try to determine shortest paths at run time**. Our solution combines de-centrelized behavior with accurate prediction of the paths, and accurate ranking the best sites to answer a query.

CHAPTER 6

Finding Fresh Data in Database Middleware Systems

6.1 Overview

This Chapter presents the AntFinder system which features autonomic and decentralized exploration of data freshness. We present the definition of a series of freshness metrics that capture different update events in the system. We provide a mechanism by which the ants can visit the nodes and gather freshness data. Also, we explain how these data structures can be used to track freshness and chose the data source(s) that comply with a freshness constraint. Additionally, we propose a notation to express freshness constraints in SQL. Finally, we discuss experiments carried out on an implementation of AntFinder in Java, deployed on a simulation built using CSIM for Java.

6.2 Introduction

Database middleware systems [57, 54, 68] enable the development of applications that integrate data from multiple data sources that have very different schemas and query processing capabilities. In the context of a wide-area network, a database

middleware system is a key element to realize data and component re-use while at the same time adding value by supporting new applications. Data replication and mirroring are widely used in database middleware systems (as well as in distributed databases) as a mechanism to improve query performance and data availability to a pool of users [29, 10]. By using replication, the data contained in a data source S_0 is copied into one or more replicas $S_1, S_2, ..., S_k$, as depicted in Figure 6.1. A client site can then fetch the data of interest from either the master site S_0 or from any replica $S_i \in \{S_1, S_2, ..., S_k\}$. This approach can be used to replicate many sites and data collections throughout the system. The result is a more robust system since a failure at the master site or any replica can be overcome by fetching data from any of the remaining sites. Recently, replication middleware has been studied as a mechanism to provide replication in: a) wide-area environments running a distributed database or a database middleware integration tool [10], and b) clusters of databases supporting OLAP workloads [56].

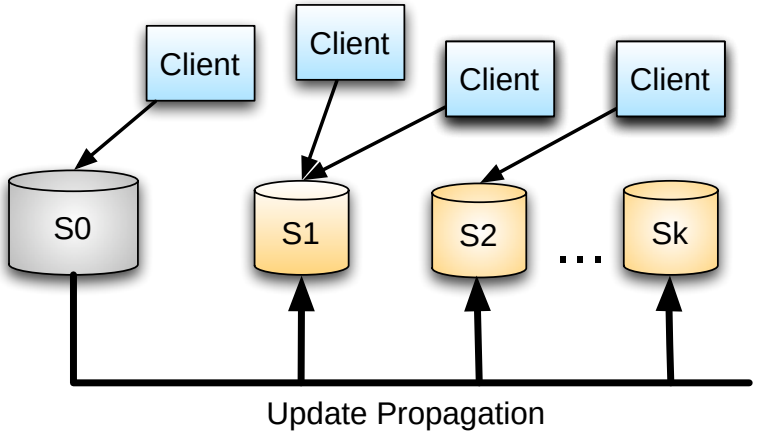


Figure 6.1. Client access to replicated data collections.

Notice, however, that replication comes at a price, since a query Q that is posed over a set of tables $R_0, R_1, ..., R_n$ might yield different results depending on the sources used to read the data. If the system uses *synchronous replication*, this is not

an issue since any update at the master copy S_0 is immediately propagated to all the replicas. But the overhead of this approach is far too high to be used in a wide-area network, and *asynchronous replication* would be the more common mechanism to manage the replicas. Thefore, the replicas will likely be out of sync with the master at various times during the daily operation of the system. This means that the same query Q can return different answers depending on which sources are used to extract the data. Ideally, the query should be routed to fetch data from either the master source S_0 or from replicas that are almost as good (up-to-date) as the master.

Given this scenario, the current state of the replicas and the *Quality of the Data (QoD)* they store now becomes an issue. *Data freshness* has been introduced as a metric to characterize the up-to-date status of the replicated data. The authors in [42, 43] use data freshness to guide the update process of materialized web views inside a web server. Meanwhile, the authors in [11] use data freshness to develop strategies to frequently update a local copy of a remote database. However, these approaches do not address the issue of extracting the data from a replica that is fresh enough to provide adequate answers to a query. The work in [56] explored an approach to route queries within a database cluster to the database node that has data fresh enough to satisfy the user request. The desired level of freshness is specified as a parameter of the query request. But this approach is more difficult to scale to a wide-area network since it uses a centralized scheduler to perform this routing. The piece that is still missing is a scalable mechanism that enables a database middleware to find replicas with data compliant with a required freshness value given by the user.

In this Chapter we explain how our approach finds the data collections that satisfty data freshness constrains specified by the user as part of the query submitted to a database middleware system. AntFinder is not a query optimization framework, but rather a *freshness management framework*. The information gathered by AntFinder can be used by a query optimizer to decide which nodes should be consid-

ered as alternatives from which data can be read to answer a query. In AntFinder, the user explicitly specifies the desired freshness level for each table as part of the SQL string sent to the system. Based on this freshness request, AntFinder locates the data sources that can participate in the query. This is done by performing a local lookup in the metadata catalog used by the server site that first receives the query from the client. After this step, query optimization can proceed as discussed in [30].

6.3 System Overview

In this case, we assume that the database middleware system is based on an architecture on which one or more integration servers (IS) connect to various wrappers (W) that take care of extracting the data from the sources, as shown in Figure 6.2. The integration server layer imposes a global schema on the heterogeneous data sources, and all queries posed by the user are expressed in terms on this global schema. We assume an *unstructured system*, where there is no central coordination site. The nodes form an overlay network for the purpose of exchanging tuples related to a given query. Each integration server contains a local catalog with metadata representing its own global schema, data source sites, users permissions, and so on. Without loss of generality, and to simplify our presentation, we assume that this schema follows the relational model and that all queries are expresses in SQL.

A client application sends its queries to one of the integration servers, and this server connects to other integration servers and wrappers to get the query solved. We assume that the integration servers have capabilities to either negotiate access to a query processing infrastructure or provide it altogether by means of a query execution engine. The integration servers rely on the wrappers to: a) extract the data from the sources, b) map the data from the local schema into the global schema, and c) execute some of the query operators necessary to generate the results. The wrappers deliver their results to the integrations server(s). Finally, the integration server originally

contacted by the client takes care of collecting all results and delivering them to the client.

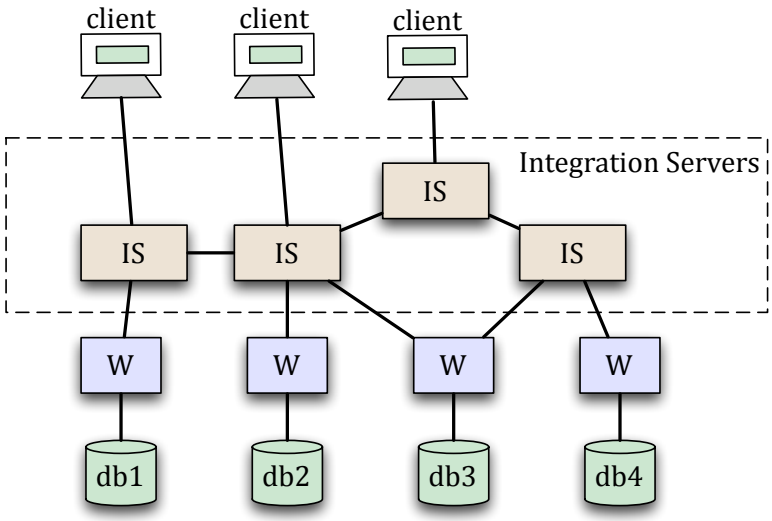


Figure 6.2. Typical Database Middleware Architecture.

6.3.1 Problem Description

The basic problem that we address in this Chapter is how to find the data sources that provide data that complies with a freshness value specified by the user for a query he/she needs to answer. Consider a query Q submitted to the system at time t_0. This query must access data from a set of n tables $\mathcal{R} = \{R_0, R_1, ..., R_n\}$ that are spread over the middleware system. Each table R_i can be read from a data source S_j^i in a collection of k sources $\mathcal{S}^i = \{S_0^i, S_1^i, ..., S_k^i\}$, where S_0^i is the master copy of the data collection and every other $S_j^i, j \neq 0$, is a replica. The system uses asynchronous replication to manage updates, and the master copy does not track the status of the replicas, making the system a pull-based replication scheme. Each replica S_j^i has an average update frequency U_j^i that denotes the average time interval on which the replicas visit the master copy to obtain updates. The following example illustrates this problem:

Example 6.3.1 *Let $R(a, b, c)$ be a global relational schema and consider the query SELECT R.a, R.b FROM R WHERE R.a >10. Table R can be located in three sites, a master copy in New York, and two replicas in Los Angeles and New Jersey. The replica in Los Angeles was updated 30 minutes ago, while the replica in New Jersey was updated two hours ago. The user is interested in accessing either the master copy or a replica of R that was last updated one hour ago. In this case, only the replica at Los Angeles meets the freshness constrain expressed by the user. Hence, the middleware system should only use data that comes from either the master copy in New York or the the replica in Los Angeles to answer the query.*

Our goal is to let the user specify this freshness constrain and provide the infrastructure that enables the system to automatically locate the source(s) that most closely match this freshness value. In this example we have expressed the freshness constrain in terms of time of last update. But there might be other types of freshness constrains. For example, the user might request that the replica might not differ from the master by more than one thousand tuples. In this case, freshness is represented as the number of updates yet to be applied to the replica. Likewise, the user's request might be expressed in terms of the percentage of tuples that must be up-to-date with respect to the master. In Section 6.4 we present three specific freshness metrics that are used in AntFinder.

Now let us turn our attention to possible solutions to the problem we have just described. One possibility is to build a catalog with freshness metadata that can provide freshness information to the integration servers performing query processing. If there are n data collections and if m is the average number of replicas per data collection, then the catalog must keep information records in the order of $O(nm)$. But this approach involves centralization of this catalog into a server or group of well-known servers. This scheme adds overhead to the query execution process since, for every query, an integration server must access this global catalog to find freshness

values for each table. In addition, it adds a single point of failure in the system. Notice also that there must be a mechanism to collect freshness statistics from all the replicas, and the catalog must track specific update frequencies for each replicated collection. This is necessary to maintain freshness statistics by frequently visiting each data source to fetch current freshness values. This means $O(nm)$ connections to get up-to-date statistics. As the number of sites increases, this approach becomes unfeasible because of its management complexity and performance overhead.

A second alternative is to simply query each candidate data source to ask its current freshness value for a given target table. That is, given a table R_i, we can contact each source S_j^i and ask for its current freshness value for table R_i. This approach guarantees accurate freshness information. However, it cannot scale to a large number of sites and replication schemes for a number of reasons. First, it would be necessary for each server in the system to track all possible replicas for each data source. Second, for each table R_i in each query Q and each data source S_j^i, the server that gets the query Q would have to establish a connection to ask each source S_j^i for its freshness value. This means $O(nm)$ connections to the target replication sites, as mentioned earlier. This approach would consume much of the server's network resources on ancillary activities and not on obtaining result tuples from the sources. In addition, it increases response time as the integration server that gets query Q needs to wait for the responses from each S_j^i before any optimization and query execution can begin.

A third approach would be to deploy an advertisement system to announce data freshness at a source S_j^i for table R_i. In this scheme, the servers would send advertisement messages $M = (R_i, S_j^i, F, t)$ indicating the freshness F at time t for table R_i at source S_j^i. Each server can cache some of these messages and use them for freshness information. The problem here is how to tune the advertisement frequency of the message for each sources. If the message are sent infrequently, the freshness

metric F becomes stale and useless to estimate current freshness. If the messages are sent too frequently, the freshness is more accurate but there is extra overhead by the excessive message passing. Moreover, some flooding technique would be needed to propagate freshness information throughout the system. In steady state, $O(nm)$ messages will be sent at regular intervals, and these need to be received and re-trasmitted (if necessary) by all nodes in the system.

The approach that we present here involves sharing freshness metadata by means of an autonomic operational scheme. In it, the integration servers of the database middleware continuously search for data freshness metadata and share these values with other servers autonomously and by an indirect method of communication. Although individual integration servers might need to store up to $O(nm)$ freshness records, they do not need to make $O(nm)$ connections to accomplish this. Instead, an integration server only looks for the metadata items that their client population needs but learns about most of the data collections and replicas from the social network build by all nodes in the system. The search for metadata occurs independent from and parallel to the query optimization/evaluation process. Hence, when a query request is received, the catalog already has accurate data freshness metadata. Our approach has several important properties that help it scale to a wide-area environment. First, the mechanism is decentralized since each integration server independently tracks the values of the freshness metrics for the data collections that its clients need. Second, each server explores the system at its own pace. Third, the addition or removal of data sources and replicas is handled gracefully by evolving the freshness metadata as time progresses. Fourth, accurate freshness metadata from the system is given to the user with a simple query lookup into the local catalog of the integration server. Thus, when a query Q is posed to a server, the freshness information can be found locally and handed immediately to the query optimization component. In the next Sections, we elaborate more on the overall operational mode of AntFinder.

6.3.2 Middleware Architecture

Applying the ACO methodology to computational problems involves, first of all, the development of a mapping of the problem to a shortest path problem in a graph. In addition, we need to define what are the food sources, the nest, the ants and the distance metric used to evaluate the goodness of each path. To accomplish this we need to have a system architecture that facilitates this mapping. In Figure 6.3 we present the architecture of the middleware system associated with AntFinder, which follows the system organization and components presented before. In AntFinder, the *nest* is the integration server that receives queries from the client. These integration servers are called Query Service Brokers (QSB) because their role is to get queries and find data and computational resources to solve these queries. Each QSB has a local catalog (denoted by the disk with the letter C in the figure) used to track metadata about the system. The QSB also has a query execution engine to process selection, projection and join queries.

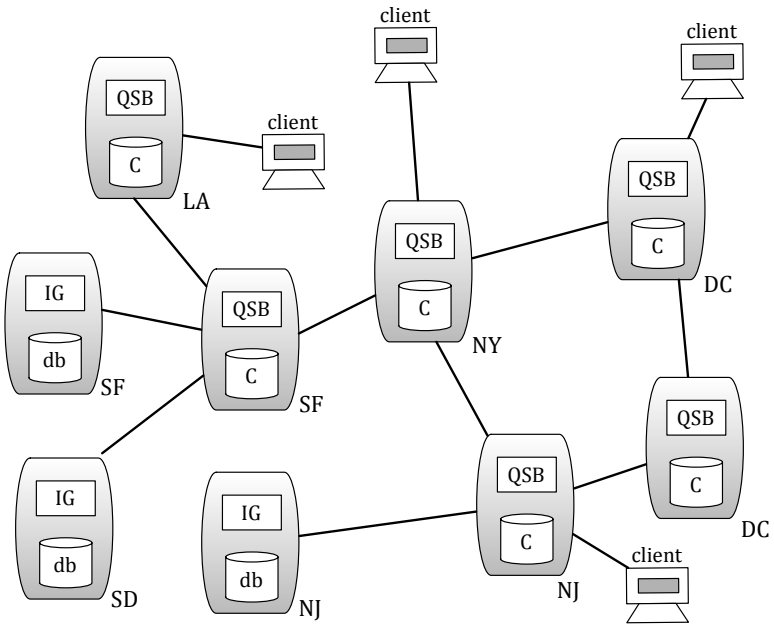


Figure 6.3. Database Middleware Architecture

Next, we have the *food sources* which are the data sources in the system. Each data source is composed of an Information Gateway (IG) and a database. The database can be a commercial relational engine, an XML store, or any other server capable of exporting data that can be mapped to a relational model. The IG has the role of the wrapper, mapping data from the local schema into the global schema used by system. In addition, the IG has query execution capabilities to help $QSBs$ execute queries. The IG also keeps track of freshness metadata related with the data collections of its associated database.

The IG and QSB are interconnected to form a Peer-to-Peer (P2P) social network for sharing data and computational resources. This network can be viewed as a graph $G = (V, E)$, where each $u \in V$ is either a QSB or IG, and each $(u, v) \in E$ indicates that $u \in V$ is a peer of $v \in V$. This peer relation indicates that node u can call node v to ask for tuples or computing time to process tuples. For example, the QSB at NY might talk with the QSB at SF to ask for help running a join or to facilite the extraction of tuples from the IG at SF. Likewise the QSB in DC might talk with the IG in NJ to get tuples for a given table. The interconnection of nodes provides the pipeline by which tuples are extracted from the local databases, transmitted and process to $QSBs$ and then delivered to the client. A *pipelining path* $P_z \in G, P_z =< u, w_0, w_1, ..., v >$, represents a possible pipeline on which tuples from IG v can reach QSB u. Figure 6.4 shows two alternative pipelining paths P_1, P_2 to reach sources for a table R from a QSB in DC. Each copy has a different freshness value, denoted by the different shades in the box representing table R. Each intermediate server w_k is a QSB that either simply forwards the tuples or applies some query operator to tuples before forwarding them to next stop node.

The cost of a pipelining path is evaluated in terms of the freshness metric F for the table R_i located at the IG S_j^i that ends the path. If two different paths P_1, P_2 lead to two different replicas of table R_i, AntFinder tells the integration server which path

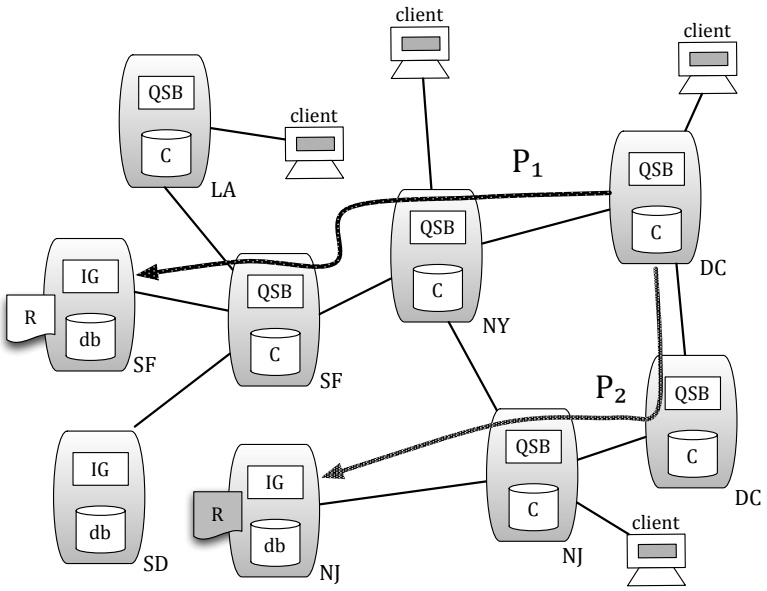


Figure 6.4. Paths explored by Ants.

is best, based on the freshness information. Hence, for each table R_i in a query Q, the QSB that first receives the query has to figure out three key pieces of information. First, the QSB must determine which paths are available to reach a data source S_j^i for table R_i. Second, the QSB must evaluate the cost of the path, which is equal to the freshness F of table R_i at source S_j^i. Third, if multiple sources are available to serve table R_i, the QSB must choose one that complies with the freshness constrains on table R_i. With this information, the QSB can determine which source has the most fresh data to satisfy the user request for each table in query Q. Notice that a path P might lead to two tables $R_i, R_{i'}$, which means that they are both stored in the same IG. But since freshness information is independently measured for each table, the paths are evaluated independently as well.

The final element in our mapping are the *ants*, which we shall call *artificial ants*. These artificial ants are software robots that are launched from a QSB B to explore the network and find the freshness metadata for each table R_i that the QSB typically accesses to serve queries to its clients. These ants also gather other information such as new data sources, and new nodes that lead to data sources.

An ants moves from QSB to QSB until it reaches an IG S_j^i that has the table R_i whose freshness is being sought, as shown in Figure 6.5. The movement of the ant is controlled by the pheromone it detects as it moves between QSBs. The pheromone is represented by a data structure already discussed in Section 4.5. Once the ant reaches the IG S_j^i, it reads the current freshness F for R_i and travels back along the same path used to reach the IG S_j^i, and stops when it reaches the QSB B. Once there, the ant will store the updated freshness information into the local catalog. But, as this ant moves back to its QSB B, it updates the pheromone at each QSB along its path to account for the current freshness value F of R_i at the IG S_j^i . Any follow up ant launched by any QSB will get to see this updated pheromone value and get influenced by the experience of previous ants at the IG S_j^i. Thus, if this IG has very fresh data for R_i, other ants looking for this table will get attracted to IG S_j^i, visit it and acquire its current freshness information.

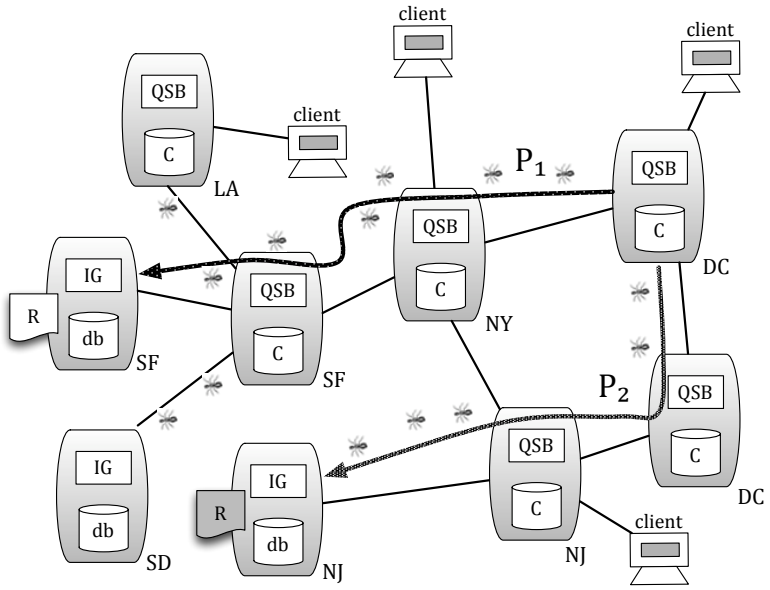


Figure 6.5. Ants exploring the system.

6.3.3 System Operation

We can break down the operation of the system in terms of two phases experienced by the QSB, namely initialization and steady-state operation.

Initialization: When the QSB gets initialized it has to be given a list of its initial peer $QSBs$ and IGs. Alternatively, some auto-discovery protocol can be used in which the QSB starts to advertise its information, schemas, etc.. The QSB is also given a set of initial tables that need to be used in queries, and the sources for these tables. The QSB then begins to send ants to investigate the freshness of these tables in each of their available data sources. This freshness information gets stored into the local catalog. During this phase, any query that is received by the QSB would have to be processed based on some initial freshness values that are configured (or guessed) for this purpose, much like a query optimizer guesses selectivity factors when such information is not available in the catalog [30]. Alternatively, the QSB might refuse queries until its catalog becomes "hot" with updated freshness information.

Steady-State Operation: Each QSB keeps track of the set of relations $\mathcal{R} = \{R_0, R_1, ..., R_n\}$ that are used by its clients. These can be all the relations defined in the portion of the global schema stored in the $QSB's$ local catalog. Another option is to track freshness for the top N percent of the tables, ranked by frequency of usage (e.g., only the top 10% in terms of requests). This can be inferred from the global schema defined in the catalog at the QSB and logs tracking target tables in the queries posed by the user. At various times $\{t_0, t_1, ..., t_k\}$ during its normal daily operation, a QSB would send an ant to find the current freshness value for table R_i at source S_j^i. The ant travels by following the pheromone trail that guides its movement between $QSBs$ until it reaches the IG associated with S_j^i. The ants get the freshness information, and returns back, updating both the pheromone trail and the local catalog of the originating QSB. When a query Q is posed to a QSB, it

will make a lookup into its local catalog to fetch the freshness information. For each table R_i, the source that has the freshness value that complies with the user request is chosen. This information is then passed to the query optimizer which takes care of generating the optimal query plan to run the query. After that, query execution begins and the clients receives tuples from the fresh data sources.

It is important to point out that a different ant is sent to visit each source S_j^i for table R_i. These ants are not necessarily sent at the same time from each node, and the frequency to launch ants depends on many factors such as the update frequency of the data sources, demand for the data, connectivity of the QSB, path length from QSB to the target IG, current system load, and so on. Moreover, a QSB might sent just a few ants and still get accurate information about freshness. Why? Because the ants use the pheromone trail built by other ants in the social network. Hence, if a QSB B has many peers and these peers sent many ants, then the ants from QSB B benefit from the exploration done these other ants. The key benefits of the ACO methods is that individual ants leverage on the experience and work of others to make intelligent desicions. For the time being, the reader should focus on the overall architecture and operation. The details of the how frequent ants are launched are described later on, and future directions on how to tune these one presented as well.

6.4 Data Freshness Metrics

The purpose of the data freshness metric is to measure the up-to-date state of the replica of table R_i at each source S_j^i. This information is then used to choose the replica that satisfies the freshness constrains of the query Q at hand. The most basic freshness metric F, presented in [11], indicates whether S_j^i is up-to-date with respect

to the master copy S_0^i at time t:

$$F(R_i, S_j^i, t) = \begin{cases} 0 & R_i \text{ is up-to-date at time } t, \\ 1 & \text{otherwise .} \end{cases} \tag{6.1}$$

This metric is a binary metric in the sense that that replica is either synchronized with the master or it is stale. However, in a wide-area network the most likely occurrence is that the replica will be somewhat out of synch with respect to the master. The key issue is to find a replica that is good enough to answer the query. If no such replica can be found, then the query must be submitted to the master source. We now present alternative freshness metrics, based on the work presented in [7]. We customize these metrics to the specific environments in which AntFinder operates. The treatment of the adequacy of these three freshness metrics for specific application domains and their relative strengths is beyond the score of this project.

6.4.1 Data Update Rate

We can define data freshness in terms of the update rate at which the replica S_j^i updates against the master S_0^i. In this sense, the replica S_j^i frequently connects to the master copy S_0^i to download the updates that must be applied to R_i. Let t_u denote the average time period between successive updates for table R_i at replica S_j^i. Similarly, let t_l denote the time of the last update for R_i at S_j^i. We can define data freshness based on update rate as follows:

$$F(R_i, S_j^i, t) = min\left\{\frac{t - t_l}{t_u}, 1\right\} \tag{6.2}$$

Notice that $F(R_i, S_j^i, t) \in [0, 1]$, with a value of 0 indicating a replica that is up-to-date with the master, and a value of 1 indicating a completely stale replica. This metric assumes a certain periodicity in the system, since new data is expected to be

available at certain periods of time. Let $t_0, t_1, ..., t_n$ denote a sequence of time stamps at which S_j^i gets updated. The behavior of the update-rate data freshness metric can be observed in Figure 6.6.

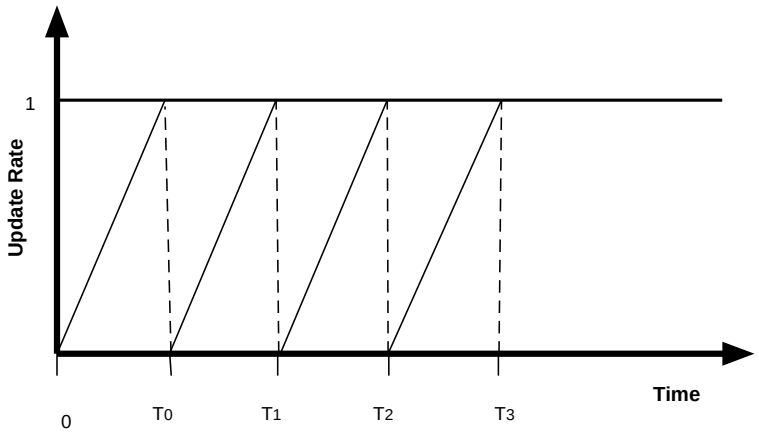


Figure 6.6. Changes in update rate freshness.

Clearly, this metric is adequate for applications in which the replicas frequently (but not necessarily within a fixed time period) visit the masters to get updates. Examples include data sources that distribute stock market quotes every two to three minutes, scores from sporting events, and temperature readings. The key issue in these examples is that updates are expected as time progresses and the replicas are expected to visit the master to get all the necessary updates to get back in sync.

6.4.2 Data Currency

Data currency measures the freshness of the data in terms of its age, which can be used by the application to determine if the data is good enough to answer the query at hand. For example, consider a football game between SF and NJ that is currently in progress. If a data source S_1 was updated a minute ago with the scores of the game, and data source S_2 was updated fifteen minutes ago, then S_1 is more suitable to answer the query: *Get the current score for the game between SF and NJ.* But if

the game ended thirty minutes ago, then either replica has suitable data to answer it.

As before, let t_l denote the time of the last update for R_i at S_j^i. We can define data freshness based on currency as follows:

$$F(R_i, S_j^i, t) = t - t_l \tag{6.3}$$

This is often called the age of the data [8]. In this case, the application needs to specify what is the maximum value that is acceptable.

6.4.3 Data Obsolescence

Another method to express data freshness relies on knowing how many tuples have changed at the master S_0^i. This is the case for many append-only data sources, such as weather trackers, movie clip databases (e.g., YouTube) or any other applications in which the important issue is to have as many updates as possible. For example, during an election night a user would like to know the results for a given candidate but only if no more than 3,000 votes are missing from the master in order to establish a winning tendency. Let ψ denote the average number of updates transactions that are applied to a source S_j^i per unit of time. Then we can define freshness at time t based on the value ψ and the last update time t_l as follows:

$$F(R_i, S_j^i, t) = (t - t_l) * \psi \tag{6.4}$$

In this case, the closer we are to t_l to more fresh the data source is.

6.4.4 Expressing Freshness in SQL

We can extend the syntax of SQL to support user constrains on the freshness of the tables by introducing the notation `FRESHNESS AT` to qualify the freshness required by the user. The proposed syntax is as follows:

```
SELECT [column list]
FROM [R1 FRESHNESS AT r], [R2 FRESHNESS AT s], ...
WHERE [predicate]
```

In this syntax, each table has an associated freshness value that must be met by the data source selected to serve tuples. The specific freshness metric must be configured as a system parameter. The following example illustrates this for the update rate metric.

Example 6.4.1 *A QSB at NY receives the following query:*

```
SELECT R.a, R.b
FROM R FRESHNESS AT 0.2
WHERE R.a > 10
```

The replica at SD has an update rate of 0.1, while the data at NJ has an update rate of 0.8. The master copy at SF has an update rate of 0. The middleware system can serve the data from the copy at SD since it satisfies the constrain, or go to the master copy. The replica at NJ does not satisfy the user request.

6.5 Experiments

In this Section we present a series of experiments that illustrate the benefits of using AntFinder to help find fresh data in a middleware system. There are several questions that must be answered before venturing into deploying a system like AntFinder into a real setting. The first question to be raised is whether AntFinder is worth the effort or if we are better off by simply asking each source for its current freshness. Using either a centralized solution or flooding for finding freshness metadata are not scalable options, so we focus on comparing our effort with the option of asking for the metadata at run time. Notice that this is the most accurate method to find the metadata. This leads to the second question, does our approach gives a QSB an accurate picture about the data freshness of the data collections reached by one IG?.

Next, we need to figure out if the freshness values of data collections at an IG are seen consistently across the system by each interested QSB. Finally, we need to consider if AntFinder can pick the data source that satisfies the freshness constrains expressed by the user. The goal of this Section is to show initial evidence on the affirmative for each these questions, and lay down the foundation for future research studies to gain a more in-depth understanding of the system.

6.5.1 Simulation Environment

We implemented the code for AntFinder as a set of Java libraries (packages) independent of the specific platform that is used for the database middleware system. We used CSIM for Java to construct a detailed simulation environment for a database middleware system, based on the architecture presented in Section 6.3.2. The main reason for using this simulation is to better understand and control system dynamics, under an environment that allows us to easily change critical parameters, repeat tests, and quickly generate data for analysis. As part of our future work we plan on coupling the system with a real prototype for a middleware system.

We developed five different models for our database middleware system, each one having a different number of servers and connectivity between these. The first model had sixteen servers, the second had twenty two servers, the third one had thirty four servers, the fourth one had sixty servers, and the fifth one had five hundred servers. Due to lack of space and in the interest of showing easy-to-follow charts we will focus on the model with thirty four servers. This model has twenty six $QSBs$ and nine IGs.

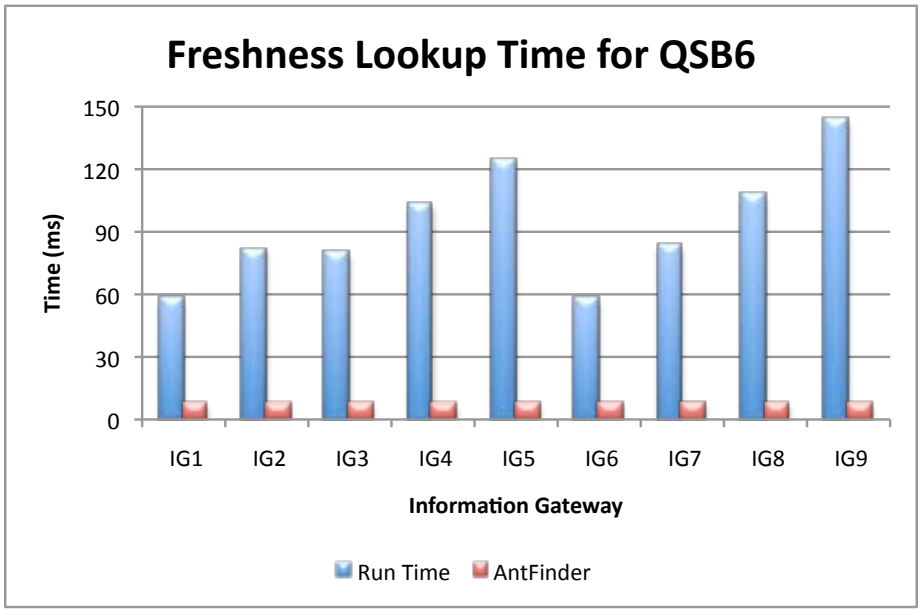
We wrote all code in Java 1.5 SE using the Eclipse 3.4 IDE. The local catalog used to store data freshness metadata was implemented as a relational database managed with MySQL 5.0. The communication between the AntFinder code and MySQL 5.0 was done via JDBC. The machine used to carry out these experiments

was an Apple iMac running MacOS 10.5 (Leopard) with a 1.83 GHz Intel Core 2 Duo CPU and 2GB of RAM.

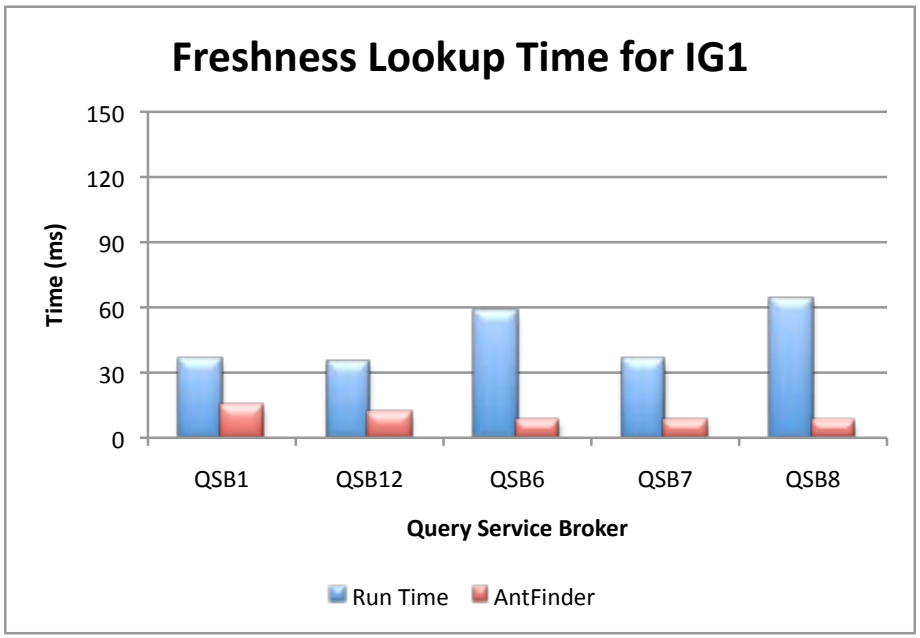
6.5.2 Overhead in Freshness Lookup

The first set of experiments was designed to measure the overhead of simply asking for the freshness from the each data source versus using the estimated freshness obtained from the local catalog managed by AntFinder. For this purpose, we configured the system to maintain a set of ten tables dispersed at random over the IGs, such that each table was kept by at least two different IGs. Each IG would register an update to each table at least once per hour, and not all tables were updated at the same time. Each QSB selected at random the table whose freshness had to be inspected and one IG that replicated that table. Then, the QSB sent ants to find the freshness value for that table at that IG. We let the simulation run for a simulation time of five weeks, and we present average values from ten independent trials.

Figure 6.7a shows the overhead in freshness lookup seen by QSB_6, which we chose to present here to illustrate what happens to a QSB that is located right in the middle of the network. We present the update rate metric as the freshness metric. The option labeled **Run Time** represents the average time (milliseconds) it takes to directly ask for the freshness information for tables in each of the IGs targeted by QSB_6. The option labeled **AntFinder** corresponds to the time is takes to make a local lookup. As we see, the option of using AntFinder is almost negligible since we are simply making a local catalog lookup. As the size of the system grows and the number of tables increases, and the distance between the QSB and target IG increases and has more hops, the difference between these two options increases and we start to get into the seconds range. Figure 6.7b shows similar information, but this time as measured from the perspective of IG_1. The results for the other metrics were similar. Not surprisingly, making a local catalog lookup is the better option.



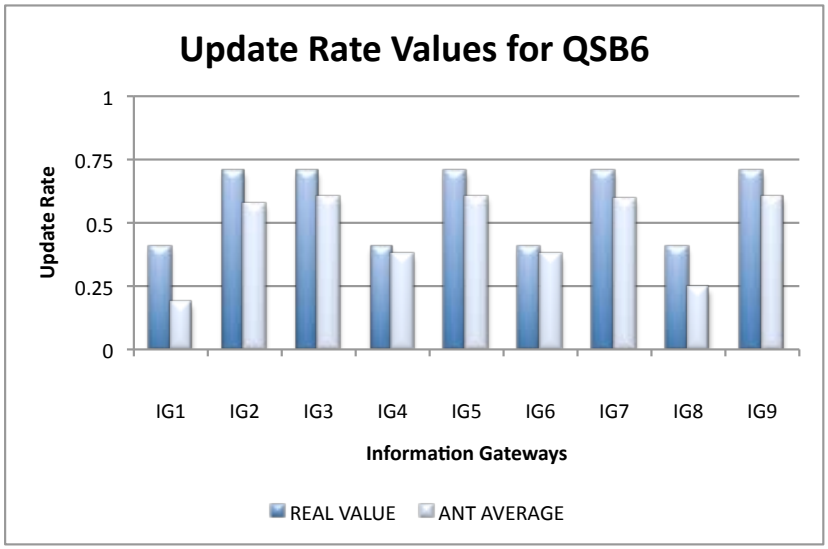
(a) Look up time as seen by QSB_6.



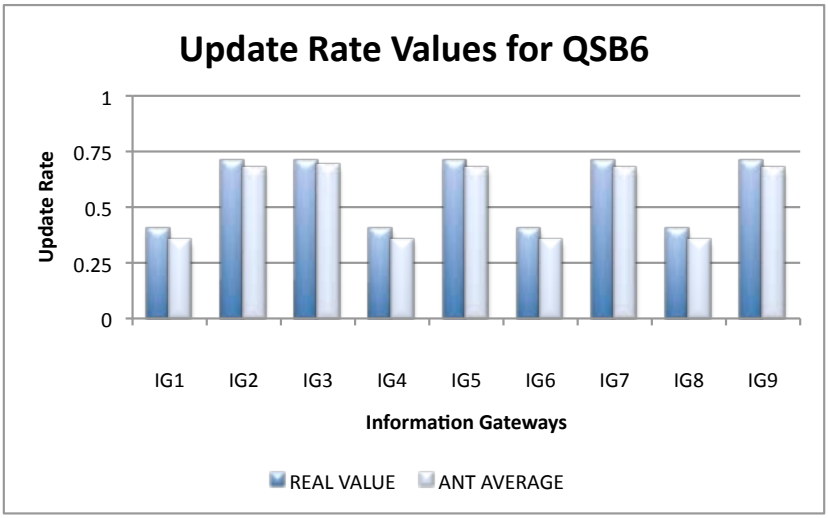
(b) Look up time as seen by IG_1.

Figure 6.7. Overhead incurred in looking for freshness.

But, is this freshness information accurate? The next Sections address this issue.



(a) One ant every two hours.

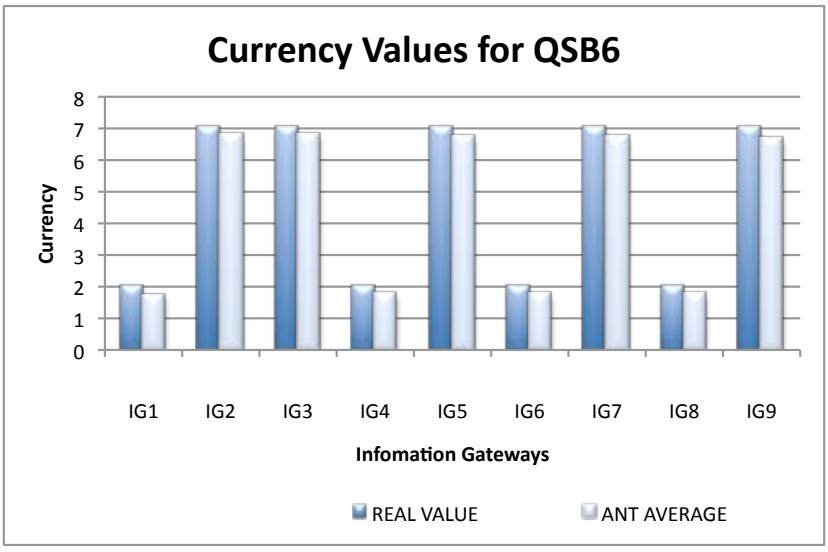


(b) One ant every half-hour.

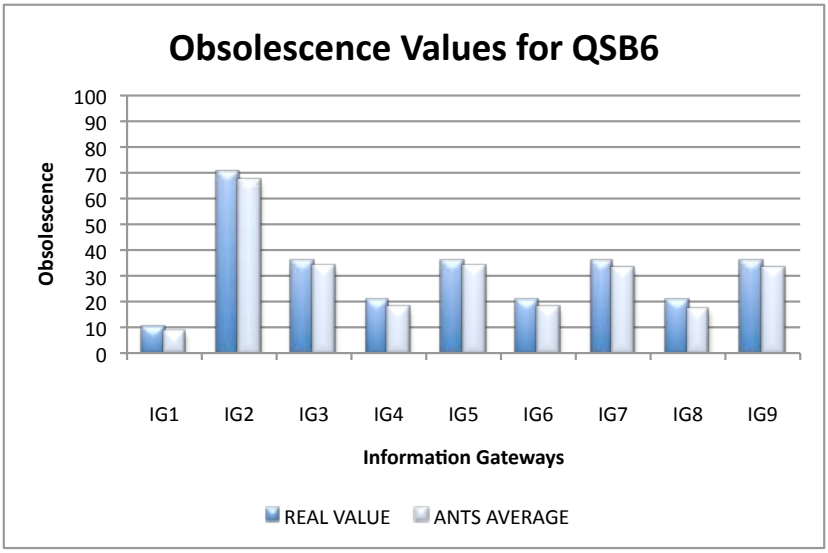
Figure 6.8. Data Freshness behavior seen by QSB_6.

6.5.3 Accuracy: the QSB Perspective

The second set of experiments was designed to compare the freshness value obtained by ants sent from a QSB to a given IG, versus the real freshness value at that IG obtained from a direct lookup. Figure 6.9 and Figure 6.8 show the results seen by



(a) One ant every half-hour.

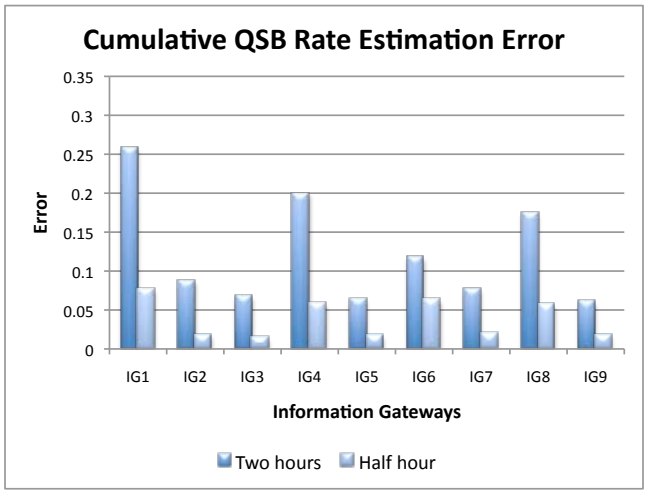


(b) One ant every half-hour.

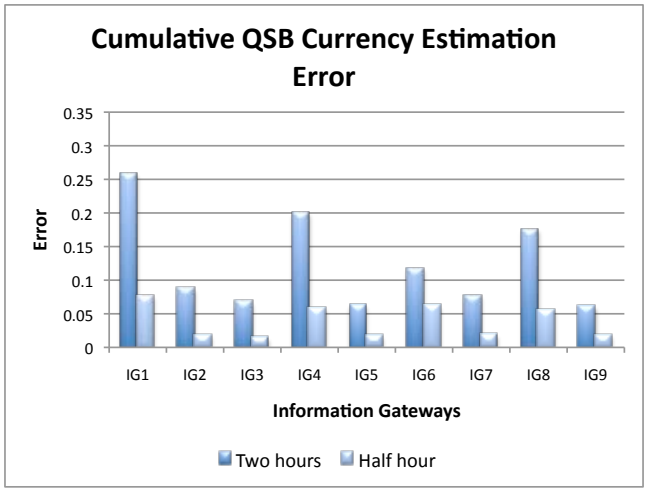
Figure 6.9. Data Freshness behavior seen by QSB_6.

QSB_6. Figure 6.8a presents the update rate metric value for each target table R_i in their associated IG S^i_j. We ran experiments in which ants were launched every hour and ever half hour. The dark bar depicts the real update rate value for the target table at each IG and the pale bar shows the freshness given by the ants. Figure 6.8b illustrates the results for the same setting but this time, the ants are sent every half-hour. As we can see, when more ants are sent the accuracy of the estimates increases. But notice the behavior with IG_4 and IG_6 in these two figures. There is little difference in the two scenarios. The reason for this has to do the with cooperative behavior of the ants. IG_4 and IG_6 are connected to $QSBs$ located in the middle of the network, having many peers. As a result, they are visited by many ants from many other $QSBs$. Despite the fact the QSB_6 sent ants less frequently in the first case, its ants benefited from the trails formed by ants from other $QSBs$, and were able to get a better picture of the freshness at these two IGs. This raises an interesting issue in terms of how to tune the number of ants sent by each QSB. It becomes clear that the update frequency of the sources, the number of hops to reach a source, and the number of peers of the servers along the path have an influence in this parameter. Figures 6.9a shows the results of the same experiments using the currency metric and ants launched every half-hour.

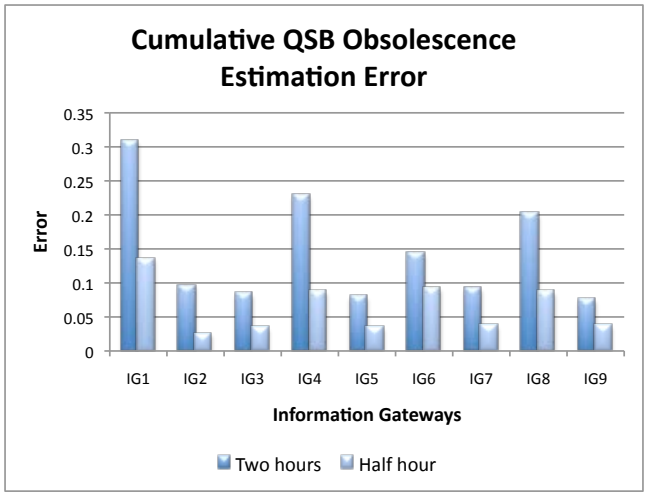
Figure 6.10 shows the percentage of error in the estimates for the freshness metrics in all the $QSBs$ in the model. The dark bars show the error for the cases in which ants were sent every two hours, and the pale bars corresponds to the case in which ants were sent every half-hour. We can see from Figure 6.10a that the percentage of error goes down without having to send ants at the level of minutes or seconds. In fact, by sending ants every half-hour we can bring the error below 10%. The same pattern is observed for the currency metric in Figure 6.10b and obsolescence metric in Figure 6.10c.



(a) Update rate error.



(b) Currency error



(c) Obsolescence error.

Figure 6.10. QSB Cumulative Estimation Error.

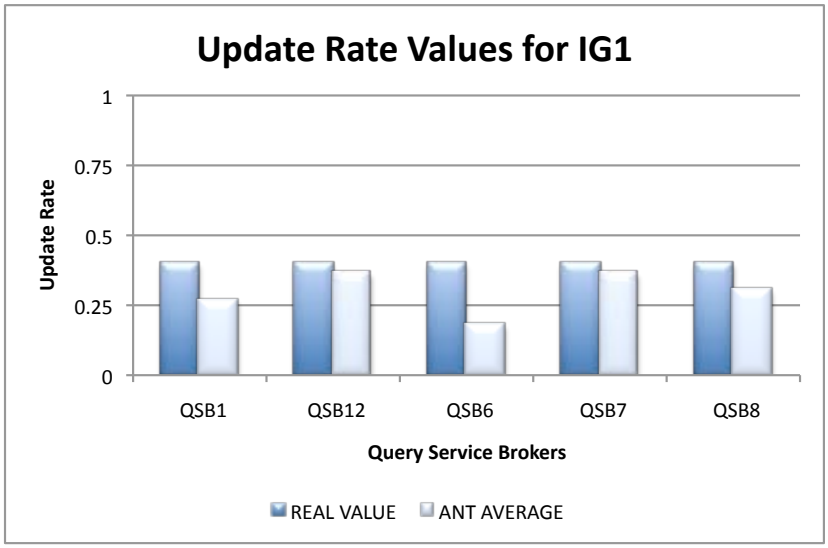
6.5.4 Uniformness: the IG Perspective

The next experiments aim at determining if the freshness of a given IG is seen consistently by a group of $QSBs$ that have interest on it. In this case, we chose a table stored at an IG and then sent ants from a set of $QSBs$ to find out its freshness. Like in the previous example, the IG would register an update to each table at least once per hour, and each QSB sent its ants at different times. We let the simulation run for a simulation time of five weeks, and we present average value from ten independent trials.

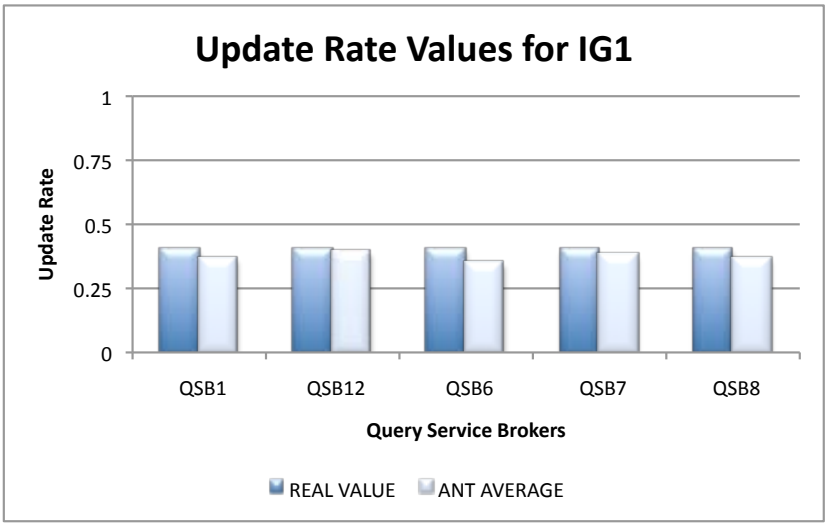
Since the trends are similar in every IG, we only show here the results for IG_1 in Figure 6.11 and Figure 6.12, the other ones can be found in Appendix A. Figure 6.11a shows the results for the update rate metric when the $QSBs$ sent one ant every two hours, and Figure 6.11b corresponds to the case in which ants are sent every half-hour. Notice how consistent is the picture about the freshness of our target table among all five $QSBs$. The values in Figure 6.11b are more accurate than those in Figure 6.11a because more ants are sent to explore freshness. But as in the results from Section 6.5.3, we see from Figure 6.11a that some QSB see a very accurate freshness despite sending few ants. These are QSB_{12} and QSB_7. Once again, the cooperative behavior of ants benefits these $QSBs$. Figures 6.12a and 6.12b show the results for the currency and obsolescence metric, respectively.

6.5.5 Choosing the Right Replica

The final set of experiments was made with the goal of finding out if AntFinder would tell a QSB the correct data source to use to accommodate freshness requirements. For each QSB that had access to tables that were replicated, we generated lookups into the local metadata generated by AntFinder to ask for the data with smallest freshness values that satisfy the specific freshness constrains in a randomly generated query. This processes generated a total of twenty nine different requests throughout

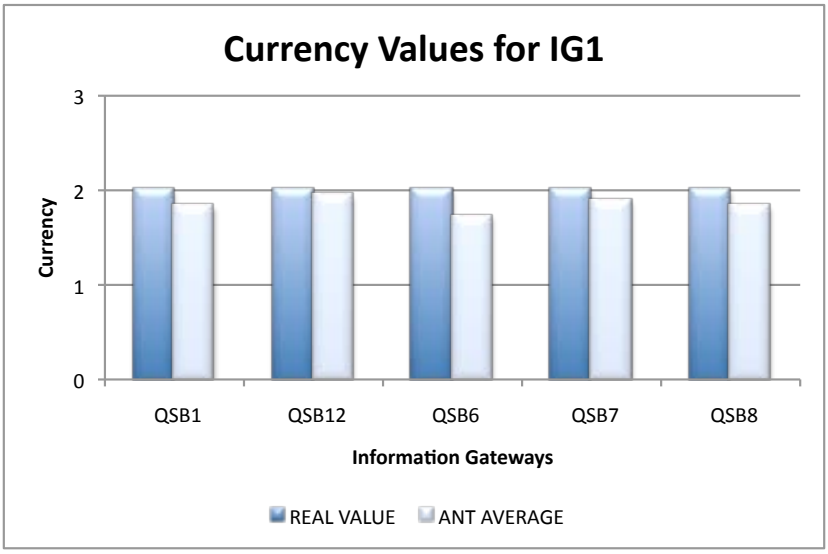


(a) One ant every two hours.

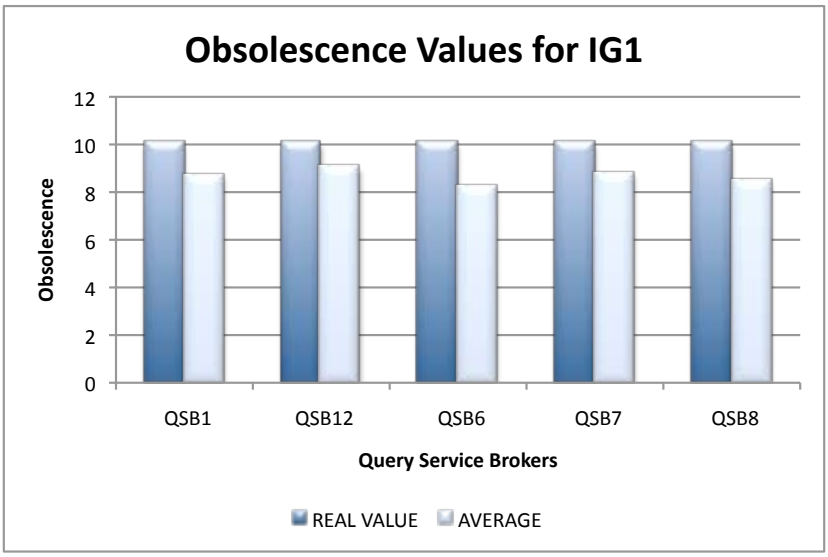


(b) One ant every half-hour.

Figure 6.11. Data Freshness for IG_1 as seen by the system (Part 1)



(a) One ant every half-hour.



(b) One ant every half-hour.

Figure 6.12. Data Freshness for IG_1 as seen by the system (Part 2).

the system. For each query, we recorded the answer provided by the AntFinder lookup operation and compared it the with right answer, which we obtained by inspecting the actual freshness value at the target IG. We repeated this process ten times for each of the three metrics. The results are presented in Figure 6.13. As before, we present results for cases in which ants were sent once every two hours and once every half-hour. Notice that the system is more successful when the freshness metric is update rate or currency. Also notice that even if the actual freshness in the QSB is somewhat off with respect to the actual value (see Section 6.5.4), the important issue is to be able to choose the source with the smallest freshness that satisfies the request. Thus, precise freshness values appear not to be as important as the getting right which source is the one with most fresh data.

Scenario	Metric		
	Rate	Currency	Obsolescence
Two Hours	99%	99%	98%
Half Hour	100%	100%	99%

Figure 6.13. Success rate for choosing the right replica.

6.6 Discussion

There are several issues that must be addressed in order to improve the capabilities of AntFinder. This discussion provides some perspective of these issues and raises possible research that should be conducted.

Strategies to launch ants: The number of ants that reach an IG influences the accuracy of the freshness metrics. Our results indicate that by increasing the number of ants sent by the QSB the value of the freshness is much better approximated. But this is not the only factor to take into consideration. A QSB that has many peers benefits from the work done by the ants sent from its peers. In addition,

the number of hops between an IG and QSB influence the travel time of the ant. The longer this path is, the more time is takes the ant to get back to the QSB to update the freshness statistics and accuracy is affected. There is a need for an adaptive method to control the process that launches ants from each QSB. One approach would be to monitor the error between the current observation of an ant that reaches an IG S_j^i from a QSB B and the value stored in the catalog at B. If this error goes above a threshold ϵ then the rate at which ants are sent from QSB B to IG S_j^i gets increased. Conversely, if the error is very small, then the QSB might opt to reduce the rate of launching and see from the next round of ants if the error is still within acceptable limits.

Overhead of the ants: An individual ant carries little overhead since it only inspect in-memory data structures (i.e., the pheromone matrix and statistical model) and updates a few tuples in the catalog of the QSB. But still, this overhead is not zero and it can accumulate rather fast if way too many ants are launched on the system. By sending ants to inspect the current freshness state of the system we end up consuming network, CPU, memory and disk resources that could otherwise be used to process queries. Care must be taken not to overflow the network with too many ants, resulting in reduced system throughput. The tradeoff we are seeking is to incur in a small overhead in order to access fresh data to satisfy user requests. If we run queries on stale data then we are also wasting resources since the query results have little value for the user. We need to characterize the overhead incurred by the ants and compare it with the cost of giving the wrong answer to figure out a control mechanism to hit the break on the processes of launching ants.

Space requirements for the pheromone matrix: We have assumed that the pheromone matrix is an in-memory data structure. Still, we need to investigate the space requirements of this matrix as the number of tables, data sources and $QSBs$ increases to determine if this is a feasible solution. If not, we need to explore

techniques to only load into memory those portions of the matrix that are most frequently used and read the rest from disk as needed.

Integration with performance metrics: In addition to using fresh data, the user most likely wants to get his/her query resolved quickly. Hence, performance must be taken into consideration at the moment of choosing the data sources that will be handed to the optimizer for plan generation. There must be a balance between the freshness of the data sources and their performance. Moreover, we must also take into consideration the performance of the $QSBs$. It would be unfortunate to give the optimizer a list of data sources with very fresh data but severely overloaded with query requests. We are currently investigating this issue by defining a cost metric that weights in the freshness of the data with the performance of the candidate sites to run the queries. We expect to report on this issue in a future paper.

Ability to meet desired freshness: The master copy always has the latest updates, and could be used to answer queries if no other replica meets the freshness constrains imposed by the user. However, if the replicas are slow to get updated, then care must be taken not to route all queries to the master copy since it will get overloaded. In fact, if every request has to go to the master, then the replication effort is not useful at all. We need a protocol to control when can the freshness guarantee be met, and when should the system simply return an empty result set indicating that no sources could be found to meet the freshness constrains. Alternatively, the system might enqueue the query to be executed at a later time when the sources get updated or the master gets unloaded. In any event, the user should receive feedback and the application must be designed to handle this situation.

Security: Having ants move between $QSBs$ and IGs opens the door for a malicious ant to damage the pheromone matrix or consume local resources at the host QSB. In our implementation, the ant moves by simply creating a copy at the destination and transferring its memory state. Still, the potential for security breaches

is there. We need to develop a security model and sandbox architecture to make sure ants do not pose a threat to the $QSBs$ and IGs. Right now, AntFinder runs as a separate thread inside the QSB or IG. Alternatively, we can run AntFinder as a separate process form the QSB or IG. A third approach is to run AntFinder in a machine different from the QSB or IG but located in the same LAN. Admittedly, we need more research into the security aspects of AntFinder.

6.7 Summary

In this Chapter we have explored the problem of developing a scalable mechanism that enables a database middleware to find replicas with data compliant with a required freshness value given by the user. As solution to this problem, we have presented AntFinder, a decentralized framework for finding the data collections that satisfty data freshness constrains. When a query Q is submitted into an integration server, this server makes a local catalog lookup to find the freshness of the candidate data sources and picks the one that satisfies the freshness constrains in the query. To maintain freshness information in their local catalogs, the data source servers and query processing servers form a social network that empowers them to share freshness metadata from multiples sites in the system. The freshness metadata are collected by autonomous software robots called *artificial ants*, that mimic the behavior of real ants, as modeled by the Ant Colony Optimization (**ACO**) paradigm. The behavior of these ants enables autonomic operation because the ants autonomously move around the system discovering data source sites and data freshness values associated with target data collections. We implemented AntFinder as a Java library and tested it with a CSIM for Java simulation of a database middleware system. We have conducted experiments that indicate that the freshness values obtained by the ants are accurate when compared with the real values at the sources. In fact, we can obtain an error of less than 10% for the estimate of freshness values. Moreover, AntFinder picks the

right data source to serve the fresh data for a given table in at least 98% of the cases we tried. This shows promise as a solution for a wide-area environment and more research shall be conducted to improve the capabilities of AntFinder.

CHAPTER 7

Strategies for Launching Ants and Sharing System Knowledge

7.1 Overview

This Chapter presents the issues related with the problem of how to find a good strategy to launch the artificial ants over a middleware system. First, we explain every strategy used and explain how could each one be improved. Then, we discuss experiments carried out on an implementation of AntFinder in Java, that was deployed on a simulation built with CSIM for Java.

7.2 Introduction

Finding metadata about the performance characteristics of servers and data freshness of the data sources involves overhead since one must visit the servers to inspect the appropriate parameters. This means that the process of finding the metadata steals computational resources and network bandwidth from the jobs that perform query processing. The tradeoffs that one must explore is how to collect these metadata while being as little intrusive as possible. In the area of performance evaluation this effect is called *perturbation*, and methods are proposed to reduce the level of system

intrusion while instrumenting a system to collect performance data [48, 31, 66].

In the work on AntFinder that we presented so far, we have not considered the effect of this overhead. However, our experimental results indicate that by increasing the number of ants we get a more accurate picture of the system. But, how many ants should be sent to ensure accurate results yet not stall query processing? Recall, that in the Chapter 6 we presented results that indicate that a QSB located in the middle of the network does not need to send as many ants as others located on the edges. What are the strategies needed to launch ants?

Another question that must be raised in whether an ant needs to complete its journey from site u to site v to collect the metadata or if it can simply read this information from some cache located in the network. If so, how accurate is the metadata compared with the ants that complete a full trip? In this Chapter, we investigate these two questions.

7.3 Ant Launching Strategies

In this Section we discuss several strategies that can be used to launch ants to the system from each QSB $u \in V$. The basic idea is to discretize the process of launching ants in terms of a sequence of rounds $\rho_0, \rho_1, ..., \rho_{t-1}, \rho_t, \rho_{t+1}, ...$, such that χ number of ants are launched during each round ρ_t. This scheme is illustrated in Figure 7.1. Recall that AntFinder is a decentralized system. Hence, each QSB has its own set of rounds occurring at different intervals. That is, given two QSB $u, v \in V$, the rounds of u are not synchronized with the rounds of v. Each one is sending ants at its own pace without knowledge of ants from rounds belonging to other $QSBs$. Rounds can occur in serial mode and concurrent mode. When the QSB operates in *serial mode* a round ρ_t cannot begin until the round ρ_{t-1} completes, as shown in Figure 7.1. Thus, the ants from round ρ_t cannot be launched until the ants from the round ρ_{t-1} complete their trip to the target destinations. This schemes simplifies the implementation, but

its major drawback is that it limits the speed at which the system can be explored. For this reason, we used a *concurrent mode*, where two rounds ρ_t and $\rho_{t'}$ can overlap, as illustrated in Figure 7.2. In fact, more than two rounds can overlap and these need not be consecutive. This enables the metadata from rounds that complete quickly to be incorporated into the system as soon as possible, while rounds from ants that visit remote locations take longer to be completed. One of the advantages of this scheme is that ants that follow an efficient path quickly influence the pheromone matrix and the statistical model of the nodes on that path.

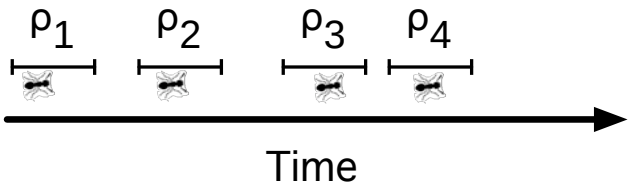


Figure 7.1. Rounds for launching ants (serial mode).

There are two options to determine the number of ants χ launched from each QSB site u. The first option is to let χ be a constant value c, which can be a global parameter for all $QSBs$, or can be locally defined parameter tuned specifically for each QSB. The second option is let χ be a number drawn from an exponential random variable X with a mean μ, emulating ants as packets travel through the network [40, 65, 53]. In this case, the mean μ can be a global parameter, or a local one specific to each QSB. In this work, we use the second option with μ being a global parameter. Developing a methodology for finding the right value for constant c or mean μ is beyond the scope of this dissertation.

7.3.1 First strategy: One Ant per Round

In this first strategy, illustrated in Figure 7.3, given a QSB u we sent one ant to a destination IG v that is chosen randomly from a pool of b possible destination IGs. Note that if u can reach b destinations, then we need to run at least b rounds

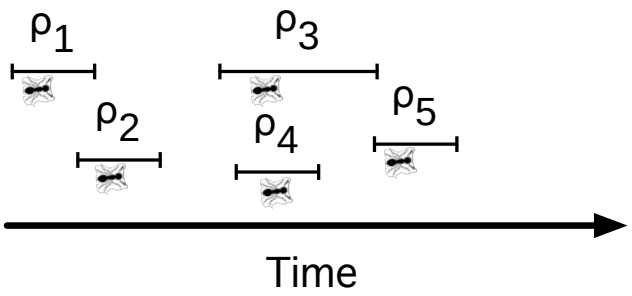


Figure 7.2. Concurrent mode for rounds to launch ants.

to obtain information about all these destinations. Clearly, the information about the system could be of poor quality if multiple changes in freshness or remote server performance metrics occur during the current time window. When we implemented this strategy, we were concerned about possible bias resulting if the system sent ants in a particular order that could lead to a cyclic order of visits. Also, we were concerned about repeating the same server multiple times if the selection was totally random. For these reasons, our approach was to plan a set of b rounds, execute them, and then plan the next b rounds. During each planning phase, a given destination v is associated with a given round ρ_t, such that every destination is visited once in each set of b rounds. The order in which a destination v gets visited is different between successive groups of rounds.

Table 7.1 contains an example explaining the ideas presented above. Consider a situation as illustrated in Figure 7.3, where QSB_1 is sending ants with this strategy to three IGs: IG_1, IG_2, IG_3. First, the QSB_1 finds all known destination IGs at that moment in time, extracting the column identifiers from the pheromone matrix $\mathcal{T}_u$. Then it plan a series of rounds to visit the system. In this case, obtaining the metadata (about a desired metric) from all destinations would take three rounds. Thus, it first plans three rounds, executes them, plans the next three rounds, executes them, and so on. Table 7.1 show the process for rounds divided in groups of three, required to visit the IGs IG_1, IG_2, and IG_3. As we see in Table 7.1, the order used

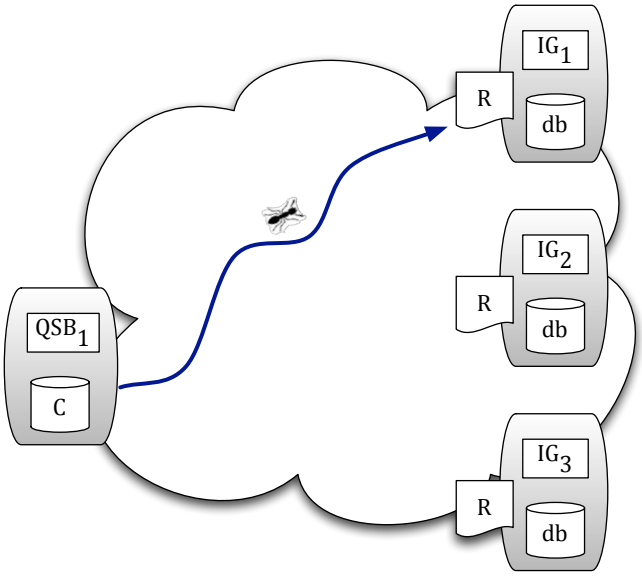


Figure 7.3. One ant per round.

by the QSB to send ants over the rounds is changing, and by doing this, we can ensure that the ants are trying to reach all possible destinations during a number of b cycles, 3 in this case.

Table 7.1. Example of Ant Launcher Schedule using One Destination per Round

Round	Destination
ρ_t	IG_1
ρ_{t+1}	IG_2
ρ_{t+2}	IG_3
ρ_{t+3}	IG_2
ρ_{t+4}	IG_1
ρ_{t+5}	IG_3
ρ_{t+6}	IG_1
ρ_{t+7}	IG_3
ρ_{t+8}	IG_2

We consider this strategy to be the least invasive, having the smallest level of intrusiveness and impact on the query workload of the system. However, it has its drawbacks, particularly if the system is large and each node has many destinations. For nodes that can reach many destinations this strategy is particularly weak because

many rounds must occur before reliable metadata are obtained. In contrast, nodes with fewer destinations can trust their metadata earlier. This means that the quality of the metadata can be quite variable between different nodes at a given point in time t.

7.3.2 Second Strategy: One Ant per Destination per Round

In this strategy, given a QSB u we send one ant, in random order, to every destination IG v that can be reached from u (i.e., those IGs with data that QSB u needs to access). Figure 7.4 illustrates the strategy. In this case, in just one round we get metadata about all the nodes that can be reached from u. Of course, an ant might die in the her trip, so getting metadata from all target sites in a given round ρ_t is a best case scenario. Notice that for each combination of nodes $u, v \in V$ we are exploring one possible path in any given round ρ_t. Successive rounds will enable the system to explore alternative paths for each combination of nodes $u, v \in V$. Thus, during the normal operation, the first round ρ_t for a QSB u would yield metadata about its target IGs, but the consistency of the metadata is uncertain. Successive rounds will make the metadata more consistent with the real behavior of the system parameter (i.e., performance, freshness).

Table 7.2 depicts the process to send an ant per destination in a given round. As before, the first step is to inspect the pheromone matrix $\mathcal{T}_{QSB_1}$ to build a list with the identifiers of all target destination IGs. Then, the QSB schedules a different ant to visit each destination v in this list. We have two options to control this process. In this first option, the QSB can send all ants at the same time. Since AntFinder is a multi-threaded system, the QSB would need to allocate one thread of execution to each ant. This means that a QSB with many destinations (IGs) will have to spend many threads to complete this process, and a lot of computing resources will be spent. A second option, is to group the ants into smaller groups and send these at

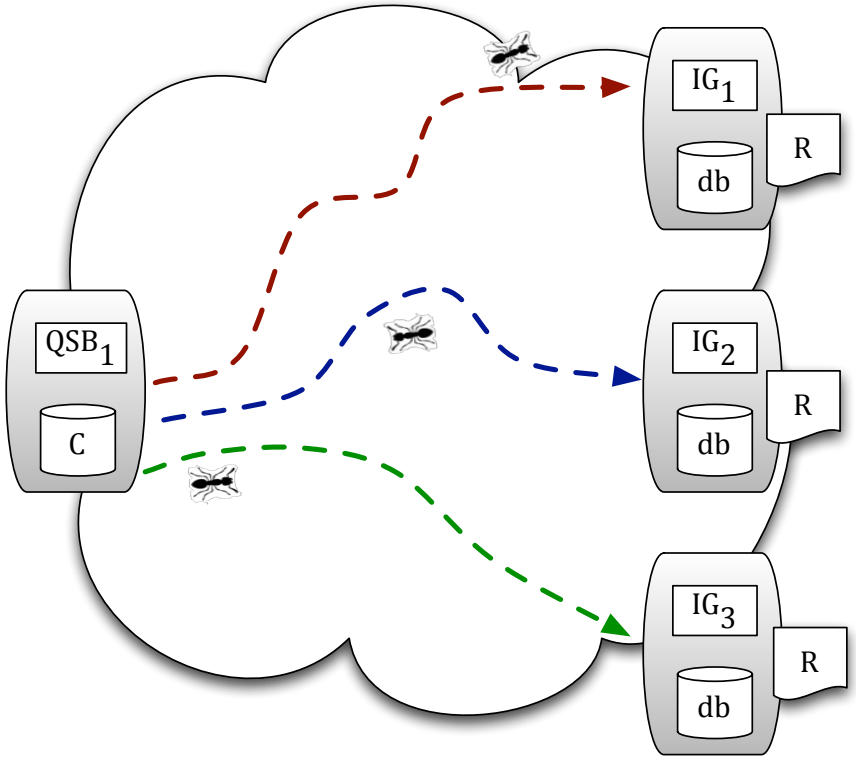


Figure 7.4. One ant per destinations per round.

the same time. For example, suppose we are currently in round ρ_2 at a QSB u with twenty destination IGs. Rather than sending all twenty ants at the same time, we can divide the process into first sending ten ants and then sending the remaining ten. This way, we control the amount of threads of executions consumed by AntFinder. In our implementation, we used the second option. In fact, we had to used it because we were getting errors at run time because the Java Runtime Environment (JRE) ran out of threads for $QSBs$ with many destination IGs.

Table 7.2. Example of Ant Launcher Schedule using One Ant per Destination per Round

Round	Destination
ρ_t	IG_1, IG_2, IG_3
ρ_{t+1}	IG_2, IG_1, IG_3
ρ_{t+2}	IG_3, IG_1, IG_2
ρ_{t+3}	IG_2, IG_2, IG_1

This strategy is more invasive than the first one and therefore the level of intrusion of the ants in the system workload increases. However, it has the advantage that in a few rounds, it obtains information on every IG. Notice that since many $QSBs$ are launching ants from different location in the system, the pheromone matrices at these $QSBs$ get updated more frequently and their consistency gets better in a shorter period of time.

7.3.3 Third Strategy: Multiple Ants per Destination per Round

In the third strategy we randomly sent $\mathcal{K}_u$ ants to every destination v that can be reached from a QSB u in each round ρ_t. Recall that the ants choose the paths in the forward direction using the pheromone trail. Hence, we can only guarantee that the number of ants that are sent during a round ρ_t from a node u to a destination v is proportional to the number of possible neighbors that can reach node v, and which have been previously discovered by the ants and stored in the pheromone matrix.

Now we will explain this methodology in detail. This strategy requires two steps: preparation phase and shipping phase. During the preparation phase we must define the destinations to be visited by the ants. Also, we need to establish the number of ants that should be sent to each destination. The destinations to which the ants can go are given by the identifiers of each column of the matrix of pheromone $\mathcal{T}_u$. Furthermore, we define the number of ants sent to a destination v as the number of positions that have the column v of the pheromone matrix $\mathcal{T}_u$ greater than zero. This actions complete the first phase, then we know how many ants to sent to each destination during the round. During the second phase, there are two options: to send all ants at the same time, or to sent the ants concurrently in random order during the round. In our implementation we used the second method to reduce the number of required threads of execution, reduce congestion during shipping, and avoid the

negative impact on the workload of the node.

Figure 7.5 illustrates what would happen in a full round when we use this strategy. In the preparation phase the system generates the information shown in Table 7.3. Meanwhile, Table 7.4 shows an example of how the ants would be sent during the round. In this particular example we have three potential outcomes for a destination v after the round:

- There is only one path available to reach destination v, and only one ant is sent along this route, as in the case of IG_3.
- There are multiple paths to reach destination v, and ants are sent to all possible available neighbors to reach node v from node u. This is the case of the IG_1, where two ants were sent and they took different paths (QSB_1, IG_2) and (QSB_1, QSB_3, IG_2) to be explored.
- There are multiple paths to reach destination v, and ants are only sent to several but not all of the them. In fact, it might be the case that two different ants follow the same route. This situation happens in the case for IG_2, where two ants were sent during this round and they will generate information about the path (QSB_1, QSB_2, IG_1) but not about the path (QSB_1, QSB_3, IG_1) .

This strategy launches many ants per round, and we can expect the pheromone matrix and statistical model at each node u to get accurate values in a short period of time. But the strategy is the most invasive of all and therefore the level of interference may affect the service quality for the workloads of the system.

Table 7.3. Phase 1 Results using Multiple Ants per Destination

Destination	IG_1	IG_2	IG_3
Number of Ants	2	2	1

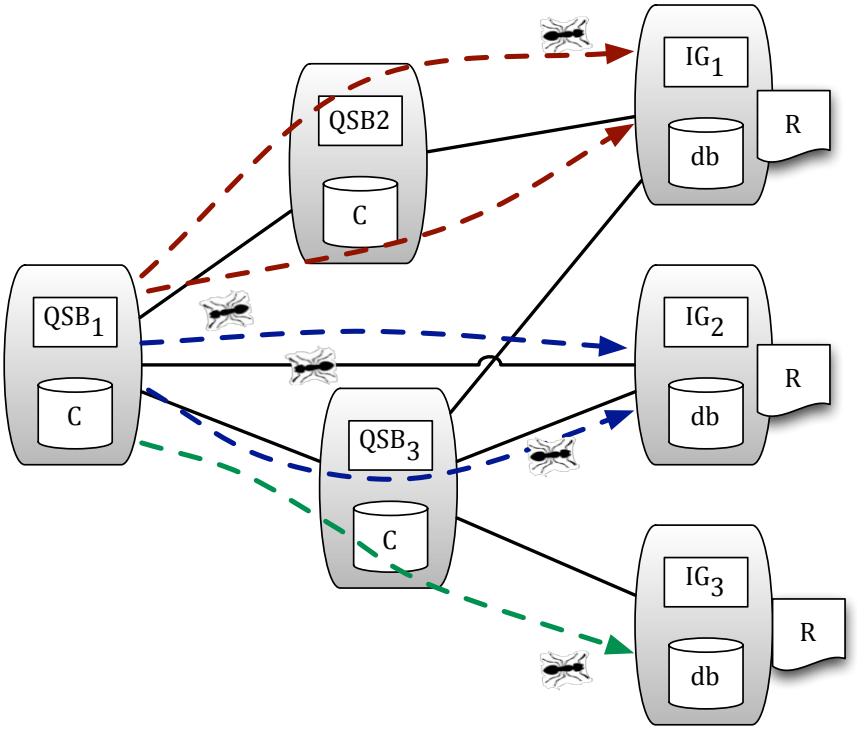


Figure 7.5. Multiple Ants per Destinations per Round.

Table 7.4. Example of Ant Launcher Schedule using Multiple Ants per Destination

Round	Destination
ρ_t	IG_1, IG_2, IG_3, IG_1, IG_2
ρ_{t+1}	IG_2, IG_1, IG_1, IG_3,IG_2,
ρ_{t+2}	IG_1,IG_3, IG_2, IG_1, IG_2

7.3.4 Launching Frequency

The final item to be discussed in the Section is the frequency by which we begin rounds. As mentioned before, was cannot guarantee that a round ρ_t will have a specific time duration T. Instead, we can control how frequent rounds are started at each QSB u. We call this parameter the *round frequency*, which is measured in time and denotes the frequency for starting new rounds. For example, a QSB u might have a round frequency of thirty minutes. This means that new rounds are started every thirty minutes. Other QSB u' might have a round frequency of one hour. Notice that it might be the case that a round ρ_{t+1} gets started before a previous round ρ_t

completes. This is a natural consequence of the fact that the paths between nodes have variable size and the time it takes for the ants to traverse these paths is variable as well.

7.4 Collecting Parameters from Caches in the Statistical Model

We have already discussed how AntFinder can be used to find metrics such as performance and freshness on the middleware system. We recognize that the number of ants to be launched by every node u per unit time and the strategy used to send them greatly influences the accuracy of the estimated metrics.

Our results show that by increasing the number of ants sent by each QSB we get an increase in the expectations for estimators approximating the reality (i.e., reduced error). However, we observed as well that some nodes had more visitants and launched more ants than others. For example in Figure 7.6, QSB_1 has many connections and hence a heavy load situation unlike other nodes with less connections such as QSB_9 that has a lighter load situation. Consequently, we must be careful to not overflow some nodes with too many ants, resulting in reduced system throughput, while in other cases underflowing some nodes, diminishing the quality of estimators for some areas on the middleware system. Next we explain our approach to deal with this issue and the possible scheme to improve the our basic ACO-based approach for estimating freshness and performance parameters.

7.4.1 Cooperation between ants with Polydomy

The basic biological model used during the development of this thesis was based on a colony of ants in one nest or *monodomy*. The system consisted on several independent colonies, each having one nest, located at a QSB. Cooperation between the ants was limited to influencing the pheromone matrix and statistical model in each QSB.

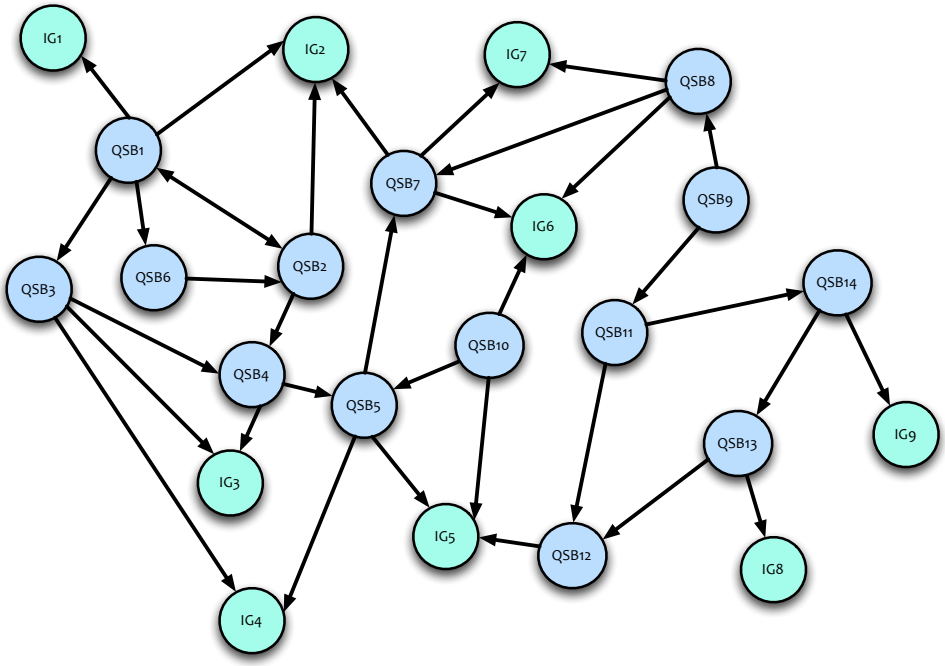


Figure 7.6. Variable Loads Example

However, our computational problem, in this case seeking metadata throughout a system of distributed databases, is considered more broad and complex. The question that must be raised is whether we can add more cooperation between the ants without having direct communications between individual ants. When searching for a similar biological model to help us find better solutions, we found a model for ant behavior known as *polydomy*.

The authors in [18] define *polydomy* as an arrangement of an ant colony in at least two spatially separated nests, as illustrated in Figure 7.7. By acquiring a polydomous structure, a colony may increase its rate for capturing of resources (food or nest sites) by the expansion of its foraging area and increased efficiency of foraging. Ants from one nest might visit another nest belonging to the same colony to get food. In fact, ants from one nest that has little food might go to another nest (from the same colony) with plenty food and transport the food back to their food deprived nest.

By allowing the colony to forage over a greater area, polydomy also allows the

diversification of food resources, and thereby strengthens opportunism in foraging and the stability of the colony's food supply. Social insect colonies have frequently been considered as central-place foragers, similar to some solitary animals [36]. However, polydomous ant societies diverge from the classic central-place model because the different nests of a colony are often not aggregated in one central place. This real behavior maps better our middleware architecture, and provides a framework to extend our notion of food. Now the food will not only be the metadata at the IGs, but also cached metadata located at $QSBs$ belonging to the same federation of cooperative $QSBs$. This cached metadata will be used by ants to avoid making a full trip to the target IGs. Instead, the ant will inspect the cached metadata record and if it is still valid, the ant will use this metadata record to update the pheromone trails and statistical model of the nodes in its path back to the QSB that launched it.

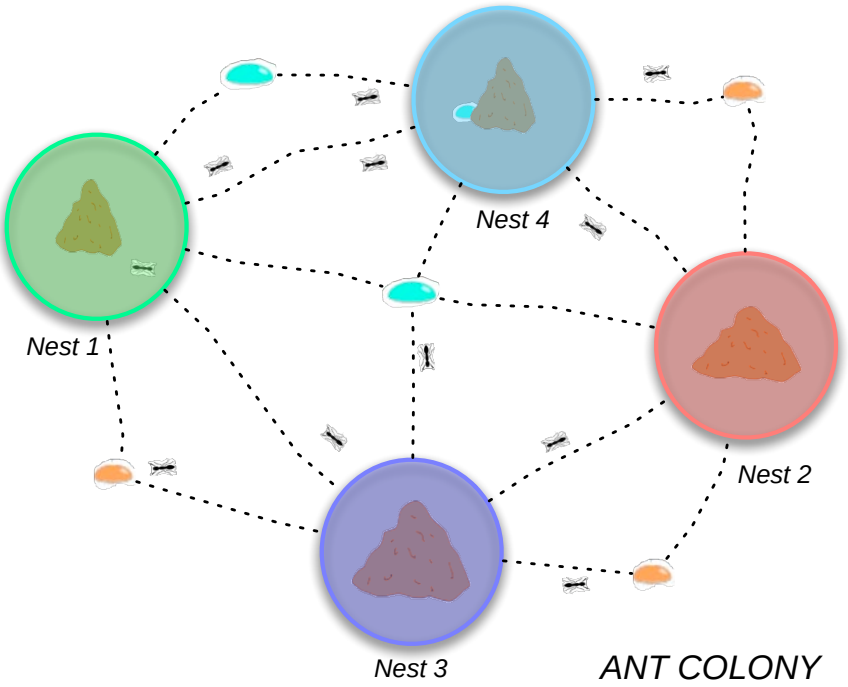


Figure 7.7. Polydomy in Ants

7.4.2 Lazy Ants: Improved Search Algorithm

Polydomy enables us to enhance the ants' cooperative behavior without breaking the indirect communication assumption, and then we can take advantage not only of the pheromone matrix $\mathcal{T}_u$, but also explode the statistical model $\mathcal{M}_u$ as a cache tool, in an effort to improve the ant launching strategies. Specifically, the behavior that we desire is for an ant X traveling from node u to node v to reach intermediate node w and inspect the statistical model $\mathcal{M}_w$. In this object $\mathcal{M}_w$ there will be a cache containing recent observations from the system. If this cache has an entry for the value parameters of w and this entry is not expired, then the ant X takes this value as good, and then generates the backward ant Y to begin the process of updating the pheromone trail and statistical model along the return path. If the cache item is expired, the forward ant X will continue the journey as usual until it reaches destination v, as shown in Figure 7.8. But, if the cache item is valid the trip is shortened and the ants returns earlier, as shown Figure 7.9. We call this method *lazy ants* since the ants attempt to avoid doing the whole trip to the destination.

This new methodology produces changes in the original algorithm in both phases: the *Solution Construction Phase* and on *Update Phase*. Figure 7.10 presents the original search algorithms, while Figure 7.11 shows the lazy ants approach. In the original algorithm, an ant going from node u to node v only stops searching when it reaches the node v, as shown in Figure 7.10, line 3. In the new algorithm, the ant that goes from node u to node v stops searching when:

- it reaches the node v or,
- it stops earlier if one of the nodes w found during the trip has unexpired cached information about the destination v

This is illustrated in Figure 7.11, line 3. If this ant finds information that is not expired, she stores in its memory and then she starts the *Update Phase* and refreshes

the Pheromone Matrices $\mathcal{T}$ and the Statistical Models $\mathcal{M}$ over all path, as would occur when the ant reaches the destination v. At this stage there are also differences between the algorithms, although not seen directly on Figure 7.11. as would occur when the ant reaches the destination We explain the differences next, using an example.

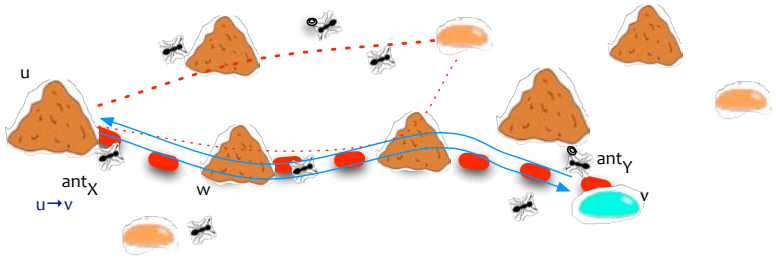


Figure 7.8. Traditional Ants Behavior

Figure 7.12a shows an example in which the forward ant with destination node v_4 moves along the path $v_1 \rightarrow v_2 \rightarrow v_3$ and arrives at node v_4. Then it generates a backward ant from node v_4 to v_1, following the path $v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$, as we can see from figure 7.12a, we explain the details of the Update Phase before in Section 4.5. Now, we illustrate one example with the Lazy Ants Algorithm. Figure 7.12b shows an example in which the forward ant with destination node v_4 moves along the path $v_1 \rightarrow v_2$ and there, she find information not expired about node v_4. Then it generates a backward ant from node v_2 to v_1, following the path $v_2 \rightarrow v_1$, as we can see from figure 7.12b.

At each node v_i, $i = 2, 1$, the backward ant uses the stack $S_{1\rightarrow 4}(v_i)$ to update the values $\mathcal{M}$ and $\mathcal{T}$. Specifically, at node v_2, the backward ant updates the

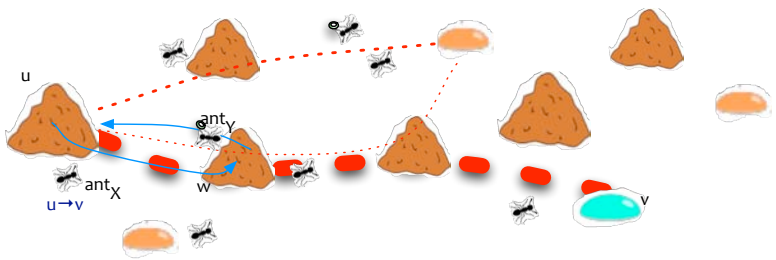


Figure 7.9. Cooperative Ants Behavior

```
LAUNCH FORWARD ANT(u, v);
1  for each(ActiveForwardAnt(u, w, v))
2        do
3           while (w ≠ v)
4                 do
5                    next ← SELECT NEXT(w, v);
6                    MOVE(w, next);
7                    memory ← MEMORIZE(next, cost);
8                    w ← next;
9           LAUNCH BACKWARD ANT(v, u, memory);
```

Figure 7.10. Initial Approach

```
LAUNCH FORWARD ANT(u, v);
 1  for each(ActiveForwardAnt(u, w, v))
 2        do
 3           while ((w ≠ v)||(newInfo == NULL))
 4                 do
 5                    newInfo ← SEARCH INFORMATION(w, d);
 6                    if (NEWINFO == NULL)
 7                       then
 8                             next ← SELECT NEXT(w, v);
 9                             MOVE(w, next);
10                             memory ← MEMORIZE(next, cost);
11                             w ← next;
12                       else
13                             memory ← MEMORIZE(newInfo);
14  LAUNCH BACKWARD ANT(v, u, memory);
```

Figure 7.11. Lazy Ants Approach

statistics $\mathcal{M}_2\,(\mu_{24},\ \sigma^2_{24},\ \mathcal{P}_{24},\ \mathcal{F}_{24})$ and the pheromone trail $\mathcal{T}_2$ directly at position τ_{234} and all other τ_{2j4}, where $j \in \mathcal{N}_2$ and j can connect to node v_4. Notice that in the first case (i.e., $w = 2$), the algorithm just updates the tuple $(\mu_{24},\ \sigma^2_{24},\ \mathcal{P}_{24},\ \mathcal{F}_{24})$ of statistical model $\mathcal{M}_2$. In the second case, the entry τ_{234} of $\mathcal{T}_2$ is update directly, and all other τ_{2j4} are adjusted because of the normalization necessary to make the sum of all τ_{2j4} equal to 1. In the original schema, the backward ant should have updated $\mathcal{M}_2\,(\mu_{23},\ \sigma^2_{23},\ \mathcal{P}_{23},\ \mathcal{F}_{23})$ and the pheromone trail $\mathcal{T}_2$ directly at position τ_{233}, and all other τ_{2j3} are adjusted because of the normalization necessary to make the sum of all τ_{2j4} equal to 1, but she does not have this information, since they just have aggregate information about the path between v_2 and $v4$.

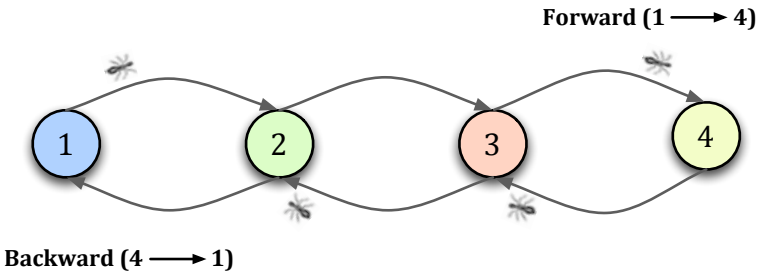
The same explanation applies for the case node v_1, in which the update is done to:

- $\mathcal{M}_1\,(\mu_{14},\ \sigma^2_{14},\ \mathcal{P}_{14},\ \mathcal{F}_{14})$ and the pheromone trail $\mathcal{T}_1$ directly at position τ_{124} and indirectly all other τ_{1j4}, where $j \in \mathcal{N}_1$, and j can connect to node v_4.
- $\mathcal{M}_1\,(\mu_{12},\ \sigma^2_{12},\ \mathcal{P}_{12},\ \mathcal{F}_{12})$ and the pheromone trail $\mathcal{T}_1$ directly at position τ_{122} and indirectly all other τ_{1j2}, where $j \in \mathcal{N}_1$, and j can connect to node v_2.

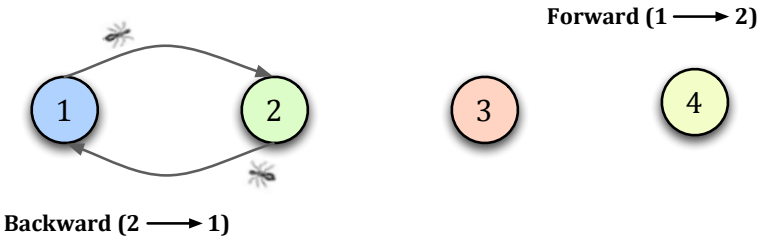
  And because lack of information, the update is not done to:

- $\mathcal{M}_1\,(\mu_{13},\ \sigma^2_{13},\ \mathcal{P}_{13},\ \mathcal{F}_{13})$ and the pheromone trail $\mathcal{T}_1$ directly at position τ_{123}

This new algorithm can launch ants using any of the strategies presented in Section 7.3 and then the question here is: when we use it, is it the quality of the results the same as when we used the original ACO-based algorithm? Is it the new amount of ants enough to reduce congestion and interruptions on the workload in the system? We shall answer these questions in the next Section.



(a) Original



(b) Improved

Figure 7.12. Structures Update Phase.

7.5 Experiments

In this Section we present a series of experiments that validate the ideas presented in Sections 1,2 and 3 of this Chapter. Initially we will explore the differences between the three methodologies to send ants by using a fixed round frequency. Next, we show the behavior of our approach when the round sending times are changing and finally label we want to explore how work the algorithm of cooperative ants. The first question is whether the quality of the metadata found by the ants using our original algorithm is superior to the metadata discovered by the new algorithm? The second question, there is a significant reduction in sent ants when they new algorithm? The last question is whether the ants cooperating to reduce congestion and the intrusion into the system without drastically deteriorated the quality of metadata obtained. In all cases, also we using this simulation tool, before venturing into deploying a system like AntFinder into a real setting.

7.5.1 Simulation Environment

We implemented the code for AntFinder as a set of Java libraries (packages) independent of the specific platform that is used for the database middleware system. We develop five different models for our database middleware system, each one having a different number of servers and connectivity between these. The first model had sixteen servers, the second had twenty two servers, the third one had thirty four servers, the fourth one had sixty servers, and the fifth one had five hundred servers. We will focus on the model with thirty four servers. This model has twenty six $QSBs$ and nine IGs.

7.5.2 Basic Ant Launchers

The first set of experiments was designed to compare the three basic strategies to launch the ants throughout the network. For this purpose, we let the simulation run for a simulation time of four weeks, and we present average values from ten independent trials. In this case, as in Section 5.3.5, we choose Pareto Bounded as a probability distribution for simulate the service time at any node and to the travel time and we sent the ants using the exponential distribution [40, 65, 53]. In all trials we used one hour as frequency round.

One variable to explore is the average number of ants that visit each node $u \in V$. Our hypothesis on this variable is that average number of ants visiting each node is statistical different for each strategy, and warrants further investigation to determine different tradeoffs in the presence of different update rates for the freshness and performance statistics. Figure 7.13 shows how the first strategy sends fewer ants during each round, and therefore is the least invasive. On the other hand, the third strategy is the more invasive since with it, we send considerable more ants during each round. Using One-Way Analysis of Variance (ANOVA) as a way to test the equality of three means at one time by using variances, we can conclude that this

difference has statistical significance, given that the p value is less than 0.05 at the 95% of confidence level. (see Figure 7.14).

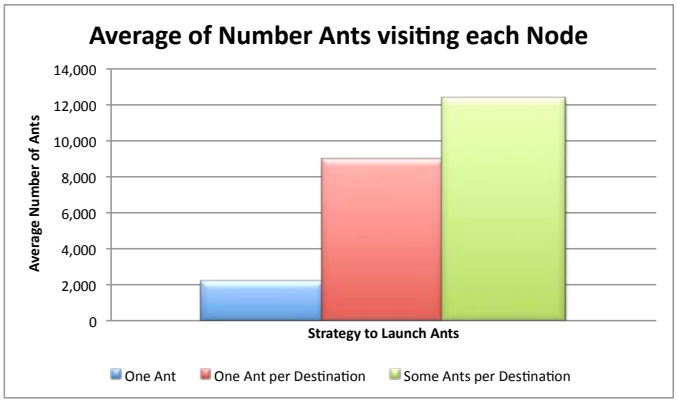


Figure 7.13. Average Attended Ants by Node

Another variable that we want to explore is the quantity of nodes that suffer some level of congestion. To explore this situation we store the number of nodes (QSB or $IG's$ where the ant needed to wait for service (when the queue length was greater than zero). Our hypothesis on this variables is that average of this measure is different for each strategy. As expected, the first strategy presents the lowest value and the third one highest. Figure 7.15 validates our reasoning and using a Anova One Way we can conclude that this difference has statistical significance at the 95% of confidence level, given that the p value is less than 0.05. (see Figure 7.16). Then, again we conclude for this scenario that the first strategy is the less intrusive and the second and third are the more intrusive. Additionally, we did not find differences between the second and third strategies. In this case, all the runs for this two strategies presents the same number of nodes with queue length greater than zero.

In terms of the quality of the metadata obtained by the ants, we next measure the pheromone matrix consistency, which measures how often the pheromone trail leads to the shortest path that is found using Dijsktra's Shortest Path algorithm. This measure is a percentage of the how frequent these two paths coincide. We expect

```
Analysis of Variance for Attended Ants by Node
Source     DF        SS        MS        F        P
Strategy    2 535216481 267608240  2.6E+05    0.000
Error      27     27779      1029
Total      29 535244260
                                  Individual 95% CIs For Mean
                                  Based on Pooled StDev
Level       N      Mean     StDev  ---+---------+---------+---------+---
0          10    2137.1      16.0  *
1          10    8878.4      41.6                  *
2          10   12304.8      33.2                            *
                                  ---+---------+---------+---------+---
Pooled StDev =     32.1            3000      6000      9000     12000
```

Figure 7.14. One-way ANOVA: Attended Ants versus Strategy

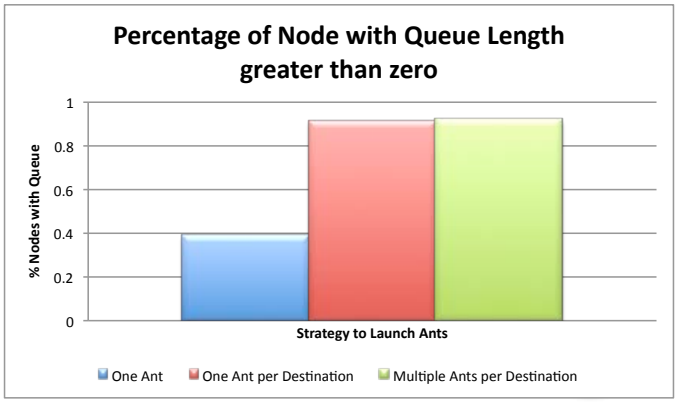


Figure 7.15. Nodes with Queue Length greater than zero.

```
Analysis of Variance for Node with Queue Length greater than zero
Source     DF        SS        MS        F        P
Strategy    2  2148.067  1074.033  7631.29    0.000
Error      27     3.800     0.141
Total      29  2151.867
                                  Individual 95% CIs For Mean
                                  Based on Pooled StDev
Level       N      Mean     StDev  -----+---------+---------+---------+-
0          10    13.100     0.568   *)
1          10    31.000     0.000                                      *
2          10    31.100     0.316                                      *)
                                  -----+---------+---------+---------+-
Pooled StDev =    0.375              15.0      20.0      25.0      30.0
```

Figure 7.16. One-way ANOVA: Node with Queue Length greater than zero.

on this variable that the average of this measure is different for every strategy, and we anticipate that the first present the the lowest value and the third one highest. Figure 7.17 shows the consistency for each of the strategies. Meanwhile, Figure 7.19 shows similarity between in the total cost of paths found by the ants and the shortest path given by Dijsktra. Figure 7.17 and Figure 7.19 validate our ideas but the difference between them is less than expected, since in this case this diference is the less than 1.5 % in average. Using a Anova One Way we can not conclude that this difference has statistical significance, given that the p value is more than 0.05. (see Figures 7.18 and 7.20).

We expected the results from the first strategy to be worst than the rest. But these results contradict our assumption. This lead us to further investigate the issues, and we found that the average measurements were favoring the first strategy. Specifically, we found that nodes that had many destinations had poor values while nodes with few destinations did when we used the first strategy. Thus, the good results from nodes with few destinations were hiding the problems at nodes with many destinations.

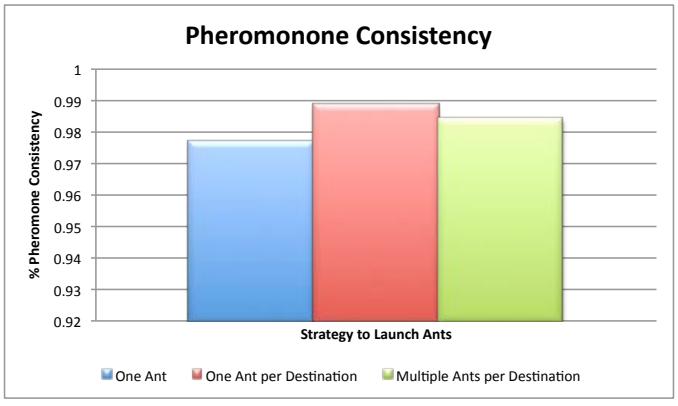


Figure 7.17. Pheromone Consistency for Basic Launch Strategies

```
Analysis of Variance for Pheromone Consistency
Source     DF        SS        MS        F        P
Strategy    2  0.000702  0.000351     2.89    0.073
Error      27  0.003282  0.000122
Total      29  0.003984

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ---------+---------+---------+-------
0          10   0.97660   0.01208   (--------*--------)
1          10   0.98830   0.01369             (--------*--------)
2          10   0.98404   0.00561          (--------*--------)
                                    ---------+---------+---------+-------
Pooled StDev =  0.01103                  0.9760    0.9840    0.9920
```

Figure 7.18. One-way ANOVA: Pheromone Consistency versus Strategy.

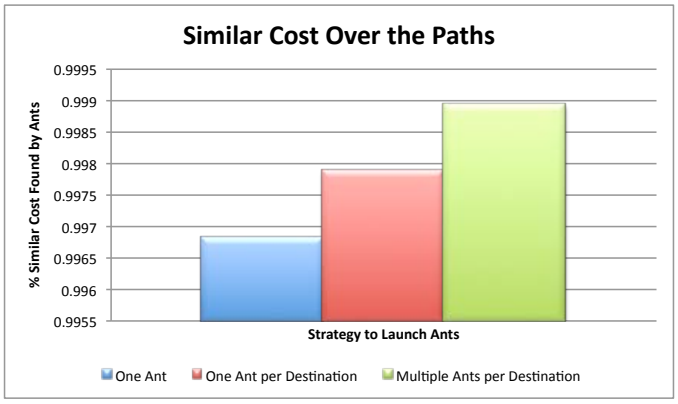


Figure 7.19. Similar Cost Paths for Basic Launch Strategies

```
Analysis of Variance for Percentage of Cost lest than expected
Source     DF        SS        MS        F        P
Strategy    2  0.000845  0.000423     1.18    0.323
Error      27  0.009676  0.000358
Total      29  0.010521

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  --------+---------+---------+--------
0          10   0.93936   0.01946   (---------*---------)
1          10   0.94787   0.01769          (---------*---------)
2          10   0.95213   0.01958             (---------*----------)
                                    --------+---------+---------+--------
Pooled StDev =  0.01893                 0.936     0.948     0.960
```

Figure 7.20. One-way ANOVA: Percentage of Cost lest than expected.

7.5.3 Effects by Changes on the Round Frequency

All the experiments presented until now use one value for round frequency, but we want to explore the impact of this variable over each strategy. When rounds frequency is varied, we observed the following changes:

- the quality of metadata decreases as the round frequency decreases (using the same measures as before), although all them are above 90%. (see Figure 7.21).
- the number of ants that visit each node increases, but strategies two and three show similar numbers (see Figure 7.22). It appears that the method of capping the number of ants based on the available threads controls the number of ants that can be sent, and we do not observe
- the nodes with queue length greater than zero decrease far rapidly for strategy one, but the other two take longer to reach zero (see Figure 7.23).

The ANOVA plots for each of these three figures that we just presented can be found in Appendix C.

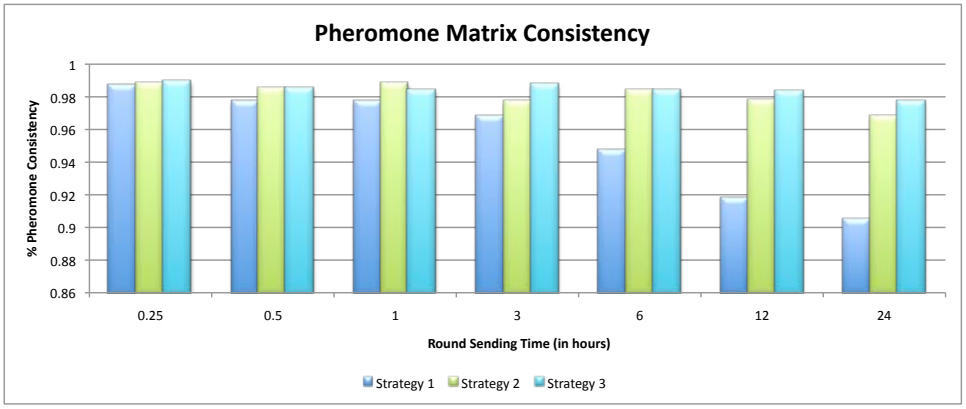


Figure 7.21. Pheromone Consistency for Basic Launching Strategies using Multiples Round Frequencies.

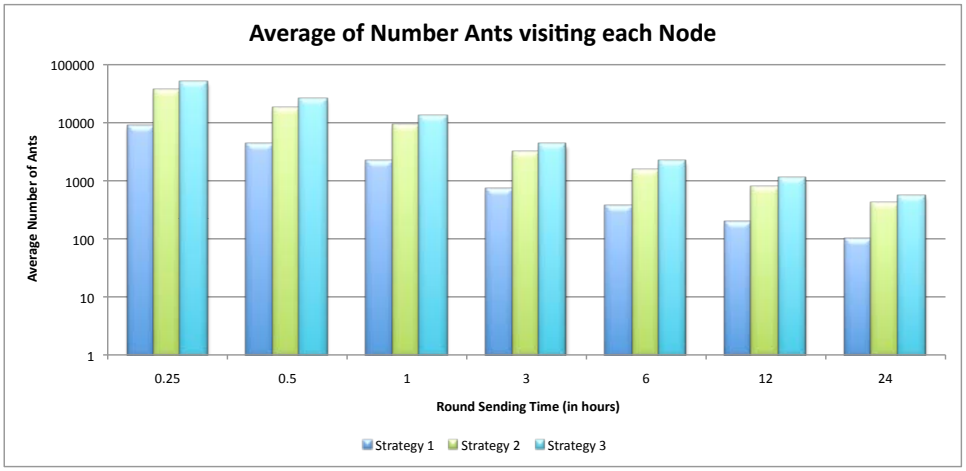


Figure 7.22. Average Ants Visiting each Node for the Basic Launching Strategie using Multiples Round Frequencies

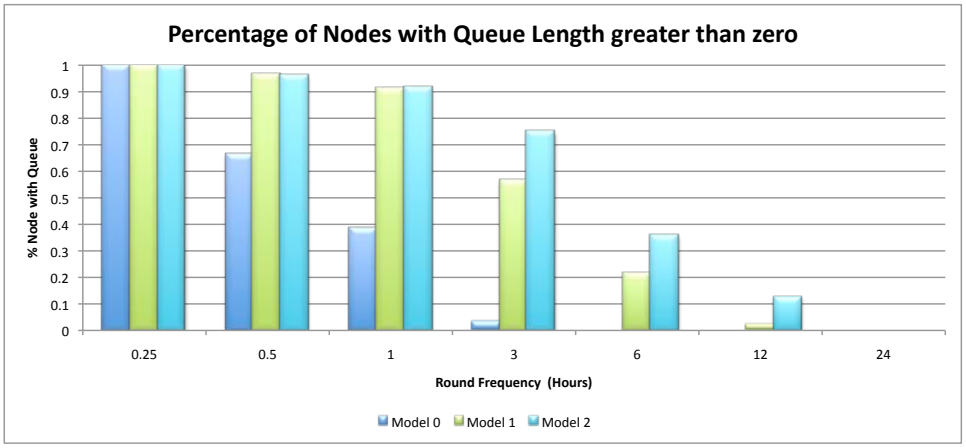


Figure 7.23. Nodes with Queue Length greater than zero for Basic Launching Strategie using Multiples Round Frequencies

7.5.4 The Lazy Ants Evaluation

This set of experiments were designed to compare the three basic strategies to launch the ants throughout the network with the new improved algorithm presented in Section 7.4. For this purpose, we let the simulation ran for a simulation time of four weeks for both versions, after a warm up period of twelve hours and we present average values from ten independent trials. Besides, we run these algorithms changing the round frequency.

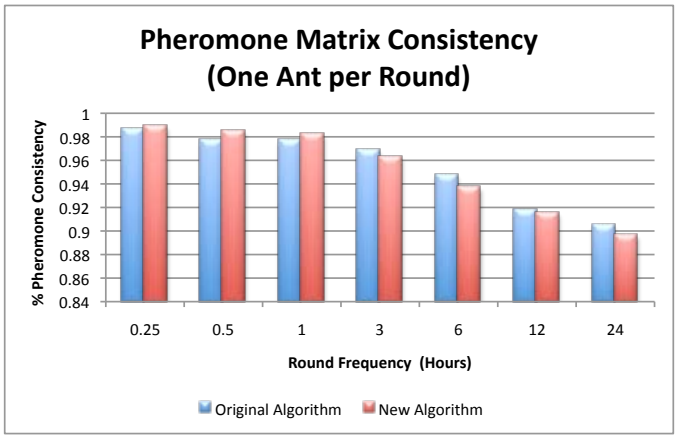
Our hypothesis about congestion(i.e., nodes with queue length greater than zero) was that this measure is significantly less for the lazy ants approach. The new algorithm reduces the average but this measure does not have the same behavior for all strategies. This results is expected, given we do not evaluate the size of the queue. In this case, we suggest new experiments as future work to investigate more details about it.

Our hypothesis about the average ants visiting each node, was that this measure is less for for the new algorithm. The new algorithm reduces the average attended Ants by Node in the first strategy approximately 40% of the attended Ants by Node, in the second one approximately to 65% and in the third one approximately 67% , as show figures 7.25a, 7.27a and 7.29a. This results is quite promissory, since the level of intrusion is reduced too, independently of the ant strategy to be use.

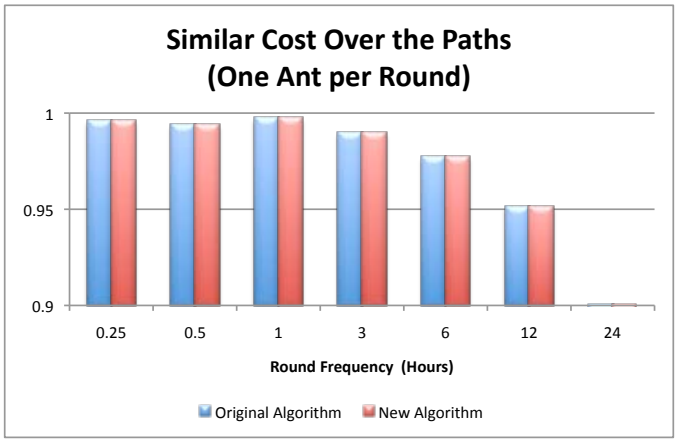
Our hypothesis about the quality of metadata, was that this measure is similar both algorithms. The new algorithm reduces the quality of the metadata, base in the used metric for this experiments in the first strategy approximately in 0.5%, in the second and third case one less than to 1.5% , as show figures 7.24b, 7.26b, 7.28b, 7.24a, 7.26a and 7.28a. This results is quite promissory, since the decrease on quality metric is acceptable, independently of the ant strategy to be use.

From these results we see that the lazy ants approach reduces the overhead

in the system while keeping good quality for the pheromone consistency. Meanwhile, strategy two appears to provide the best tradeoff between number of ants sent, pheromone consistency and congestion in the system.



(a) Pheromone Consistency

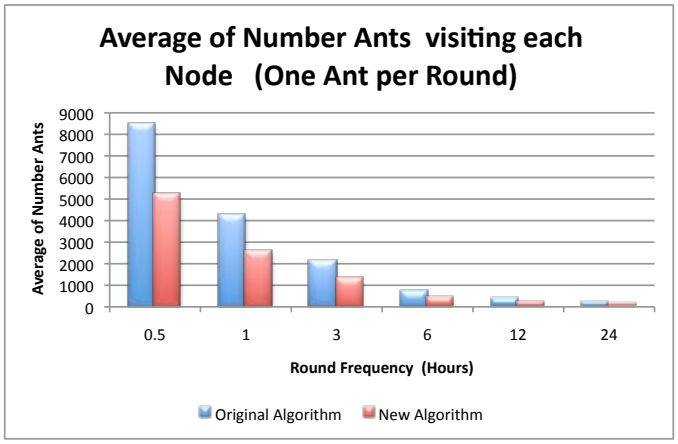


(b) Cost

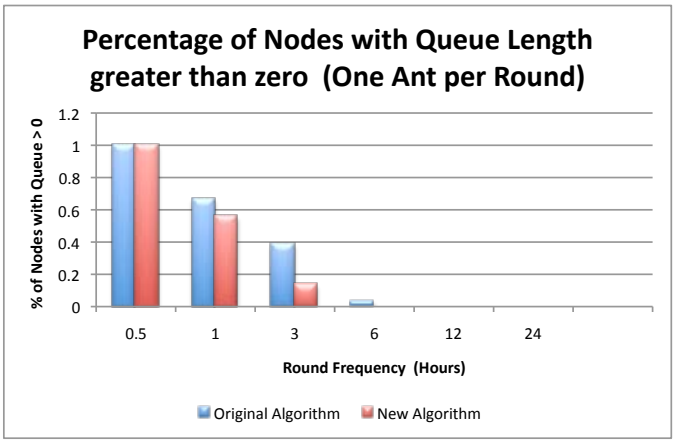
Figure 7.24. Experiments Results for One Ant per Round Strategy

7.6 Summary

In this Chapter we have presented several alternative strategies to launch ants to the system. We have introduced the concept of a round as a mechanism to schedule the ants to be launched from a source node u to each destination node v. Based on the idea of rounds, we defined the following strategies to launch ants: a) one ant per

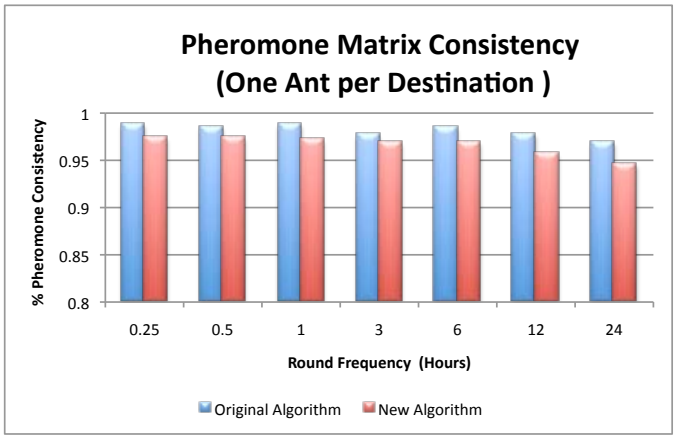


(a) Average ants visiting each node

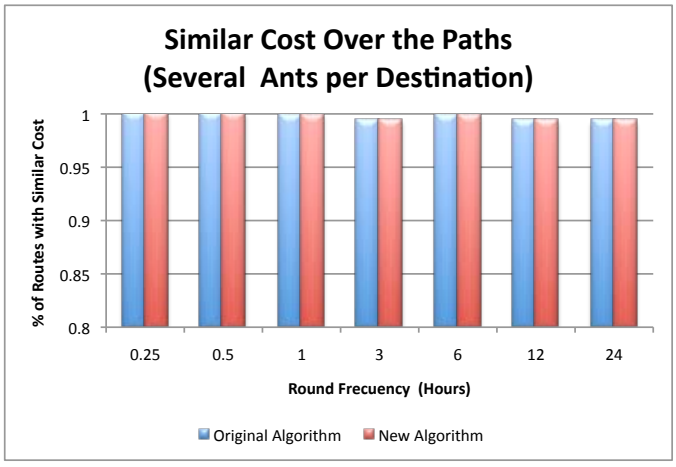


(b) Nodes with Queue Length greater than zero

Figure 7.25. Experiments Results for One Ant per Round Strategy

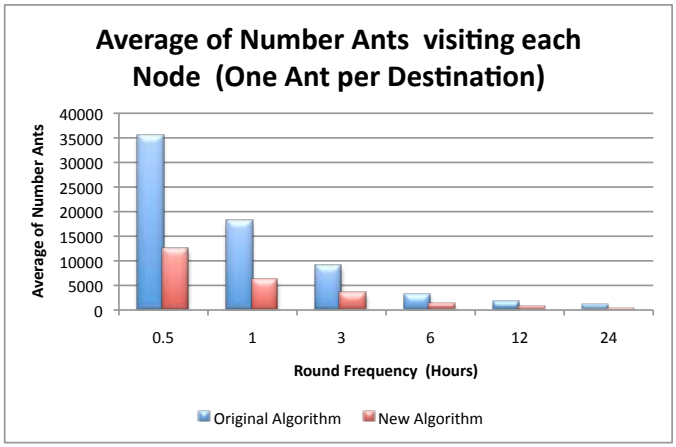


(a) Pheromone Consistency

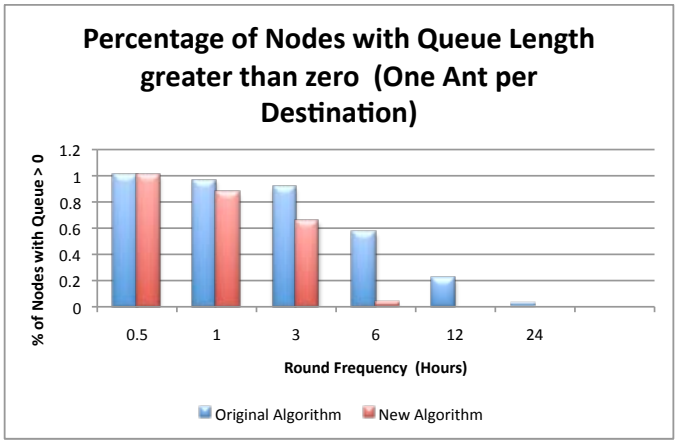


(b) Cost

Figure 7.26. Experiments Results for One Ant per Destination per Round Strategy

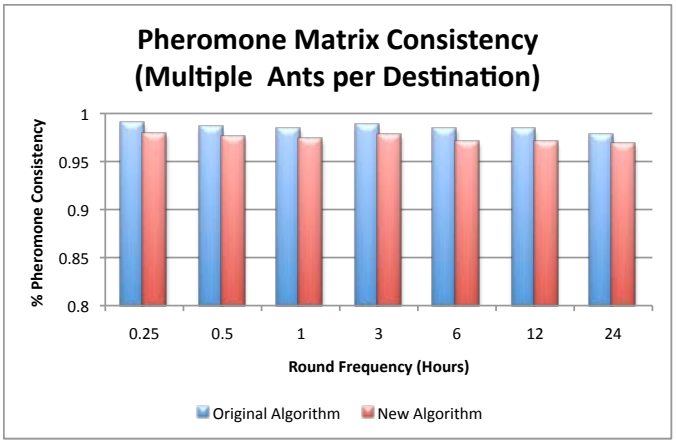


(a) Average ants visiting each node

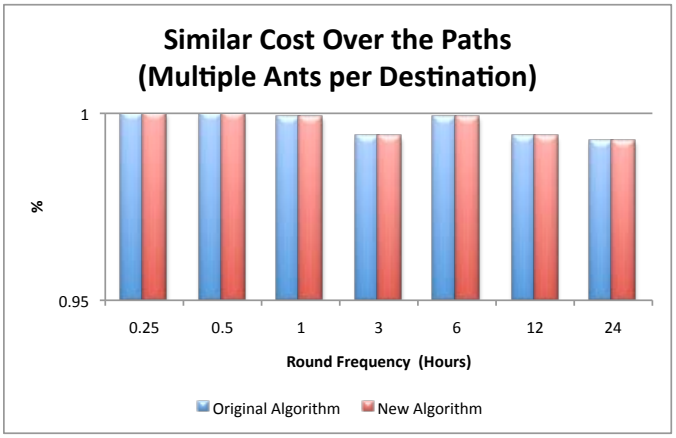


(b) Nodes with Queue Length greater than zero

Figure 7.27. Experiments Results for One Ant per Destination per Round Strategy

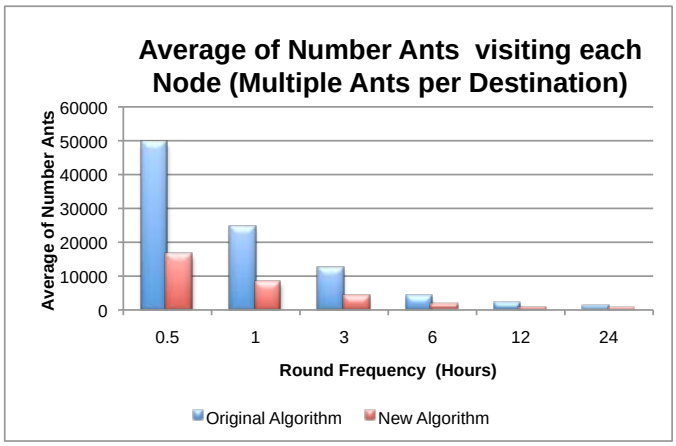


(a) Pheromone Consistency

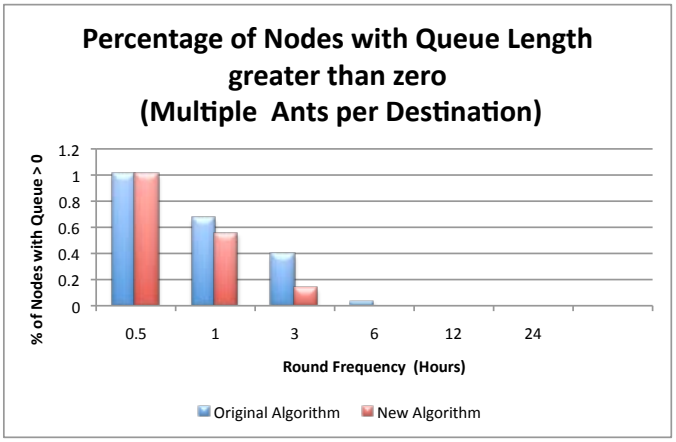


(b) Cost

Figure 7.28. Experiments Results for Multiple Ants per Destination per Round Strategy



(a) A average ants visiting each node



(b) Nodes with Queue Length greater than zero

Figure 7.29. Experiments Results for Multiple Ants per Destination per Round Strategy

round, b) one ant per destination per round, and c) multiple ants per destination per round. We also presented the concept of polydomy, where ants can have a colony with multiple nests in different locations. Ants from one nest might visit another nest from the same colony to get food and transport the food to their original nest. We use this feature to introduce the idea of lazy ants, whereby the statistical model at a node caches performance or freshness information from target IGs. Ants that visit a node u look for this metadata as if it was food. If the cached metadata is still valid, the ants cut their trip short and use the metadata to begin the backward trip updating the pheromone trails and statistical modes of the nodes in the path. Our experimental results show that the strategy for launching one ant per destination per round provides the best tradeoff between good pheromone consistency and low overhead. By adding the lazy ants we improve things by further reducing the overhead of the ants while keeping the pheromone consistency at over 90%.

CHAPTER 8

Ethical Considerations

Computer Ethics is an important topic in computer applications. As researchers, we must be aware of the wide range of ethical responsibilities that come with doing any type of research. In this Chapter we explore some concepts and issues related with ethic in computer sciences, databases and autonomic agents that would be relevant to this dissertation.

8.1 Computer Ethics

Computer Ethics is a branch of practical philosophy which deals with how computing professionals should make decisions regarding professional and social conduct. The term "computer ethics" was first coined by Maner [49]. He expressed that Computer ethics is an academic field in its own right with unique ethical issues that would not have existed if computer technology had not been invented. He explained six levels of justification for the study of computer ethics:

- We should study computer ethics because doing so will make us behave like responsible professionals.
- We should study computer ethics because doing so will teach us how to avoid computer abuse and catastrophes.

- We should study computer ethics because the advance of computing technology will continue to create temporary policy vacuums.
- We should study computer ethics because the use of computing permanently transforms certain ethical issues to the degree that their alterations require independent study.
- We should study computer ethics because the use of computing technology creates, and will continue to create, novel ethical issues that require special study.
- We should study computer ethics because the set of novel and transformed issues is large enough and coherent enough to define a new field.

All of this levels are important and we deal with all of these during the execution of this dissertation. Is important to mention that one of our motivations was try to produce accurate metadata that follow certain level of quality and in this way, the middleware could do a better job during the optimization of any query received.

8.2 Ethics in Computer Sciences

According with Wright [73], science and engineering are commonly distinguished as two different kind of activities. Sciences produces more theorical results, while engineering seeks to apply those results through the creation and refinement of technology. Computer Science, as it generally presented, taught, and researched, stands in a unique position, intimately involved in both theoretical and applied. Given that engineering requires an ethical position beyond codes of conduct [9], then a discipline that spans both theory and application, an that impact so many aspects of life, should be grounder ethical foundation. The social contract between modern scientific research and society rest upon a threefold foundation: the responsible conduct of research; clear and complete recording of research procedures, results, and analysis;

and respect for those that may be affect by that research.

8.2.1 Responsible Conduct of Research

Empirical science relies on measurements and observation to corroborate or disprove research hypotheses. In computer science, metrics and analysis methods are the primary instruments for measuring software systems. The uses of these instruments is complicated by the diverse and dynamic nature of computing, and often by the necessity under integrating the measurement tool in to the system under observation. This is further complicated by a lack of consensus on the topic and minimal attention to the quality of measurement results [1].

This ethical concern was a point of motivation in our work, since we want to provide a tool inside the middleware that has the potential to create accurate metadata relate with the query processing site and the data sources itself and in this form to pull out the responsibility on the system administrator to feed the catalog with with recent information about the network and sites behavior.

Additionally, we use simulation as a tool that allow us to manage bias and variability of the system and has a controlled situation.

Another area of bias in software is the actual implementation of the software itself. Different programing languages have features and semantics that offer more efficient ways to implement certain operations. While these features are generally obvious to the research, the optimizations that compilers can perform are often far less visible, and can vary among computer platforms (hardware and operating systems). Some studies applied over different benchmarks, shown different results depending on the language and compilers on different platforms [45]. Other author found that compiler can create differences in performance using the same program source code as well as compiling and execution on the on the same platform [32]. In our case, we avoid this issue using always the same type of computers to do out experiments and

our programs to evaluate the performance and to do conclusions.

8.2.2 Documenting and Reporting Research

The ability to duplicate the work of other researchers is perhaps the most fundamental principle and responsibility of science. Repeating an experiment allow new result to corroborate or refuted, as well as providing the means to restate or refine the problem under consideration. Duplicating previous work of other research is often more than simple recreating earlier experiments: the later research should also be looking for new results that extend or clarify early work, or be seeking some case where there exist theory does not apply.

In out work, we used previous knowledge and background from various scopes, and these with this information we construct a framework applied to the middleware system. We also document all the theoretic aspects in this dissertation related with our formulation necessary that cannot be included in our research papers, due to space limitations. We also make proper references to the ideas and work of previous researchers thorough all the prepared documents.

8.2.3 Human Participation in Computer Research

In computer science, this topic is probably most closely associated with the study of human-computer interaction, with the evaluation of computer science educational methods and software development techniques following closely behind.

Most research institutions have some form of a review board that evaluates research proposals involving human subjects. Key requirements for approval include obtaining uncoerced informed consent to participate from research subjects, and providing the option to withdraw from the study at any time and for any reason without penalty. However, there are areas of computer science research that involve humans in more subtle ways but that can still put individuals at risk to harm. Some of the aspect that Wright [72] mention and can be applied to our project are:

1. Knowledge Discovery and Data Mining (KDDM): Large and diverse data sets are examined and manipulated for the purpose of extracting hidden knowledge and patterns not otherwise observable, often using agents as faster and tireless human surrogates. The potential exists for the invasion of personal privacy. There is also the possibility of misleading results (agents or pro-grams are looking for particular kinds of patterns that a researcher expects to nd in a particular data set, etc.) that could cause harm to individuals or groups.

2. Networking: There is also the risk of potential invasion of privacy when monitoring live networks. An ethicist might also ask if the users of a computer network that is a subject of study are themselves subjects in the study? After all, it is their usage that creates the patterns of trafic that are of interest to a researcher.

3. Software Engineering: A signicant amount of research in this area is very human-involved, since it is humans that design, implement, use, and maintain software systems. Research often involves observing individuals, groups, and entire organizations, with the associated risks of harm.

We already considered this aspect during the execution. Our external agents (the artificial ants) never have access to the data sources or the processing data and work in autonomic form. Although, the ants store information on run time about the network, this kind of data is anonymous and no body can be identified using it. Also every agent is deleted after reach the source site. In relation with the data sources, we had assumed that all of that accomplish ethical aspects related with privacy. Finally, we explored codes of ethics and professional conduct ([3, 2]) during the realization of this research work.

CHAPTER 9

Conclusions

In this Chapter we provide a summary of the conclusions and contributions of this dissertation. In addition, we point to directions in future work.

9.1 Summary of Results

- *Chapter 1 Introduction*: In this Chapter we presented the motivation to this project and a problem statement.
- *Chapter 2 Literature Review*: In this Chapter we presented relevant work in the areas that form the basis of this dissertation, which include: Distributed Database Systems, Database Middleware Systems, and Data Replication.
- *Chapter 3 NetTraveler System*: In this Chapter we presented an overview of NetTraveler, the model database middleware system on which our framework operates.
- *Chapter 4 Ant Colony Framework*: In this Chapter, we discussed relevant aspects on Ant Colony Theory, which include: Basic Social Networks Concepts, the mapping between real and artificial ants in our middleware framework, the original and the adapted algorithm. We also review an extensive amount of work that has been carried out prior to this dissertation as well a work done as part of it.

- *Chapter 5 - Autonomic Ranking of Data Sources and Query Processing Sites using Ant Colony Theory*: In this Chapter we proposed an autonomic framework for continuously discovering and ranking the sites in a database middleware system for mobile, wide-area environments. Our framework can help keep the catalog updated, and feed accurate information to a query optimizer about paths with sites for query operator placement. We proposed a framework for characterizing sites based on performance. Then, we presented an adaptation of the ACO algorithm to explore the system and rank the characteristic of each site. This framework is fully de-centralized and can scale to large number of nodes. Finally, we presented the results of a performance study carried out on a simulation of the system with twenty six query processing sites and nine information gateways. These experiments show that our framework can find near optimal paths in over 90% of the cases. In addition, they demonstrate that our method is far superior to approaches that try to determine shortest paths at run time. Our solution combines de-centrelized behavior with accurate prediction of the paths, and accurate ranking the best sites to answer a query.

- *Chapter 6 - Finding Fresh Data in Database Middleware Systems*: In this Chapter we explored the problem of developing a scalable mechanism that enables a database middleware to find replicas with data compliant with a required freshness value given by the user. As solution to this problem, we presented AntFinder, a decentralized framework for finding the data collections that satisfty data freshness constrains. When a query Q is submitted into an integration server, this server makes a local catalog lookup to find the freshness of the candidate data sources and picks the one that satisfies the freshness constrains in the query. To maintain freshness information in their local catalogs, the data source servers and query processing servers form a social network that empowers them to share freshness metadata from multiples sites in the system. The

freshness metadata are collected by autonomous software robots called *artificial ants*, that mimic the behavior of real ants, as modeled by the Ant Colony Optimization (**ACO**) paradigm. The behavior of these ants enables autonomic operation because the ants autonomously move around the system discovering data source sites and data freshness values associated with target data collections. We implemented AntFinder as a Java library and tested it with a CSIM for Java simulation of a database middleware system. We have conducted experiments that indicate that the freshness values obtained by the ants are accurate when compared with the real values at the sources. In fact, we can obtain an error of less than 10% for the estimate of freshness values. Moreover, AntFinder picks the right data source to serve the fresh data for a given table in at least 98% of the cases we tried. This shows promise as a solution for a wide-area environment and more research shall be conducted to improve the capabilities of AntFinder.

- *Chapter 7 - Strategies for Launching Ants and Sharing System Knowledge*: In this Chapter we have presented several alternative strategies to launch ants to the system. We have introduced the concept of a round as a mechanism to schedule the ants to be launched from a source node u to each destination node v. Based on the idea of rounds, we defined the following strategies to launch ants: a) one ant per round, b) one ant per destination per round, and c) multiple ants per destination per round. We also presented the concept of polydomy, where ants can have a colony with multiple nests in different locations. Ants from one nest might visit another nest from the same colony to get food and transport the food to their original nest. We use this feature to introduce the idea of lazy ants, whereby the statistical model at a node caches performance or freshness information from target IGs. Ants that visit a node u look for this metadata as if it was food. If the cached metadata is still valid, the ants cut their trip

short and use the metadata to begin the backward trip updating the pheromone trails and statistical modes of the nodes in the path. Our experimental results show that the strategy for launching one ant per destination per round provides the best tradeoff between good pheromone consistency and low overhead. By adding the lazy ants we improve things by further reducing the overhead of the ants while keep the pheromone consistency at over 90%.

9.2 Summary of Contributions

The main contributions presented in this dissertation can be summarized as follows:

- Development of a de-centralized approach to dynamically characterize data sources and query processing sites in a distributed database system. Evidence is presented about its potential benefits in supporting the query optimization process in distributed and replicated systems that do data integration via middleware technology.
- Definition of a quality metric for data sources and query processing sites. This quality metric can be defined in terms of performance or data freshness. The goal of this quality metric is to establish a rank for the sites and system from a the perspective of a particular site. Using this rank, the candidate sites for data extraction and query processing can be chosen and fed to a query optimizer to generate a query plan. To the best of our knowledge no other middleware system performs such assessment of data sources.
- The development of heuristic techniques based on Ant Colony Theory [22] to implement the site characterization process. This techniques have been shown to provide good solutions to problems in other areas in computer science and Networking, and we expect to explode this experience in our research project.
- The study of different techniques to launch the ants from each node to explore

the system, based on the idea of rounds.

- The development of the lazy ants approach to send ants to explore the system, which reduce the number of ants in the system but keeps a high quality of the metadata.
- The implementation of a system prototype using Java and CSIM. This implementation shows that our prototype version of the ACO algorithm is able to find good path between the nodes, in a set up where the cost between nodes changes over time. This implementation experience will be very useful in our future integration with the NetTraveler [68] Prototype.

Also, we have a publication of initial results of AntFinder in the 2009 Asia Modeling Symposium [67].

9.3 Future Work

We conclude this dissertation with a series of task that can be carried out as future work to complement this work.

9.3.1 Autonomic Discovery and Assessment of Metrics

AntFinder enables a database middleware to find critical metrics for scheduling query operators in a decentralized and automated fashion. This behavior shall result in better performance and data freshness for the queries submitted by the user. There are three main issues that we wish to explore to better understand the impact of ants on query processing:

- What are the metrics that can be accurately measured by the ants? How can we balance performance and data freshness? Should we model this issue as a multi-objective optimization problem, or should one metric be optimize before the other?

- Can the ants provide accurate information on system dynamics? How can we control the rate at which ants are send to prevent them to cause undesired overhead? What would be the impact on the pheromone matrix if an existing site leaves the system or a new one arrives? How quickly is the pheromone matrix stabilized?
- Since the ants are moving around the system, can they be used to carry the tuples and ask nodes to run operators on these tuples? How can the ants quickly establish a pheromone trail for query processing? How can query operators be scheduled along different paths?

9.3.1.1 Task 1: Balancing the Performance and Data Freshness Metrics

Users are interested in getting their data from fresh sources, but they also wish these data to be delivered fast and efficiently. Thus, performance must be taken into consideration to provide a set of data sources that are fresh but also powerful enough to solve the query in a timely manner. Likewise, the query processing services that are chosen to participate in the query must be efficient as well. In short, the query optimizer must be provided with a set of good data sources and query services to be used in the optimization process for operator placement. Since the system is dynamic, we need to make sure that the performance estimates are accurate enough and we must adapt them as time progresses to reflect current dynamics.

9.3.1.2 Task 2: Ant-based Query Processing

Despite the fact that ants continuously provide system metadata in decentralized fashion, the optimization process is done by the QSB that first gets the query, and it decides on operator placement once and in a centralized fashion. Since the ants are traveling around the system, the question that must be raised is whether the ants can do query optimization and processing on-the-fly. The work in [12] explored how to change query plans on the fly, and the work in [47] developed methods to

continuously adapt query operators. But, these methods rely on either the having a point in time in which operators are reorder, or the addition of synchronization operators that route the tuples to destination sites. Can we use the ants to find the sources and the best place to put the operators? Can the ants find the proper order of evaluation for the operators? These are the issues we shall start to explore in this task.

9.3.2 System Modeling and Self-Steering

Distributed database systems behave as complex systems and their intrinsic operation resemble the behavior of grid computing systems. Obtaining a general view of the behavior of the collective components of the middleware system and tuning its dynamic behavior to accomplish a set of performance goals poses an interesting challenge and a set of questions arise:

- How to model the system behavior? What is the best model for distributed systems that will allow us to understand the general behavior of the system? Should it be a centralized or decentralized model or maybe a combination of both?
- How to measure and optimize this behavior to meet a particular performance goal? What type of instrumentation should we use? Should measurements from the database status be collected event in an event-based mode or sampled at regular intervals?
- Once the general behavior of the system is assessed, how to automatically steer the system so that it autoregulates to meet the objectives? Here steering refers to performance steering [69], that is monitoring the state of a system and enabling parameter changes to provoke a change in the behavior of the system.

9.3.2.1 Task 1: Modeling the Dynamics of the System

The query plans used for completing a query are based on the information that the middleware knows about the basic performance measurements from the distributed system. Simple to calculate metrics are typically used to ensure fast response. Initial decisions on the query plan should change with the dynamics of the system. On the other hand, a performance measurement does not provide information on itself but rather based on the model of the system it is resembling. Which model is the best one to ensure that an operational database middleware system is adequately represented? Diverse applications may require a different model. Since this type of application entertains an observational model, should it be a function fitting model, a time series, or a probabilistic model [25]?

9.3.2.2 Task 2: System Intrusion Measurement/Perturbation Analysis

The use of software tools or agents (e.g., the ants) to identify the status of a software system creates contention for the resources that are available for the database middleware system. Perturbation or intrusion on the behavior of the database may cause undesirable behavior. AntFinder parameters such as the rate at which the ants are sent may adversely affect the performance.

9.3.2.3 Steering the Behavior of the Database Middleware System

Vetter et al classify computational steering into application and performance steering [69]. In the former, parameters of an application are modified to change the outcome of the program. Performance steering on the other hand refers to monitoring the state of a system and enabling parameter changes to provoke a change in the behavior of the system [31]. For example, a new replicated service might be deployed by the system to maintain the throughput at an acceptable level. Vetter designed Magellan, a computational steering system to monitor and control parallel computing applications and systems [69]. However, the software architecture of Magellan may not appropriate

for distributed systems. In these systems a combination of distributed and centralized data collection and actuation may be more appropriate.

REFERENCES

[1] A. Abran and A. Sellami. Measurement and metrology requirements for empirical studies in software engineering. In *STEP '02: Proceedings of the 10th International Workshop on Software Technology and Engineering Practice*, page 185, Washington, DC, USA, 2002. IEEE Computer Society.

[2] ACM Council. *ACM Code of Ethics and Professional Conduct*, 1997.

[3] ACM/IEEE-CS. *Software Engineering Code of Ethics and Professional Practice (Version 5.2)*, 1999.

[4] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. A. Ketabchi, W. A. Litwin, A. Rafii, and M.-C. Shan. The pegasus heterogeneous multidatabase system. *IEEE Computer*, pages 19–27, 1991.

[5] M. Astrahan. System r: Relational approach to database management. *ACM Transactions of Database Systems*, 1(2):97–137, 1976.

[6] C. Blum, A. Roli, and M. Dorigo. HC–ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 2, pages 399–403, Porto, Portugal, 2001. Also available as technical report TR/IRIDIA/2001-16, IRIDIA, Université Libre de Bruxelles.

[7] M. Bouzeghoub. A framework for analysis of data freshness. In *IQIS '04: Proceedings of the 2004 international workshop on Information quality in information systems*, pages 59–67, New York, NY, USA, 2004. ACM.

[8] L. Bright and L. Raschid. Using latency-recency profiles for data delivery on the web. In *VLDB*, pages 550–561, 2002.

[9] G. Bugliarello. Machine, modifications of nature, and engineering ethics. *National Academy of Engineering*, 2005.

[10] E. Cecchet, G. Candea, and A. Ailamaki. Middleware-based database replication: the gaps between theory and practice. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 739–752, New York, NY, USA, 2008. ACM.

[11] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. *SIGMOD Rec.*, 29(2):117–128, 2000.

[12] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 150–160, New York, NY, USA, 1994. ACM.

[13] I. Corporation. Virtual table interface programmer's guide.

[14] O. Corporation. Oracle transparent gateways.

[15] S. Corporation. Data access migration, 1999.

[16] C. L. Culp. *The risk management process: business strategy and tactics*. John Wiley and Sons, 2001.

[17] S. B. Davidson, C. Overton, and P. Buneman. Challenges in integrating biological data sources. *J. Computational Biology*, 2:557–572, 1995.

[18] G. Debout, B. Schatz, M. Elias, and D. Mckey. Polydomy in ants: what we know, what we think we know, and what remains to be done. *Biological Journal of the Linnean Society*, 90(2):319348, 2007.

[19] A. Delis and N. Roussopoulos. Performance and scalability of client-server database architectures. In *18th International Conference on Very Large Data Bases*, pages 610–623, Vancouver, British Columbia, Canada, 1992.

[20] D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J.-B. Yu. Client-server paradise. In *20th International Conference on Very Large Data Bases*, pages 558–569, 1994.

[21] M. Dorigo. *Optimization, Learning and Natural Algorithms (in Italian)*. PhD thesis, Politecnico di Milano, Dipartamento di Electtronica, 1992.

[22] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[23] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. The MIT Press, 2004.

[24] A. Eickler, A. Kemper, and D. Kossmann. Finding data in the neighborhood. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 336–345, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[25] N. Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, 1998.

[26] F. Glover and M. Laguna. *Tabu Search*. Kluver Academic Publisher, 1997.

[27] C. A. Goble, R. Stevens, G. Ng, S. Bechhofer, N. W. Paton, P. G. Baker, M. Peim, and A. Brass. Transparent access to multiple bioinformatics information sources. *IBM SYSTEMS JOURNAL*, 40(2), 2001.

[28] P. P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: essai dínterprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[29] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 173–182, New York, NY, USA, 1996. ACM.

[30] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. Lorie, and T. G. Price. Accesspath selection in a relational database management system. In *ACM SIGMOD Conference*, pages 23–34, Boston, Massachusetts, USA, 1979.

[31] W. Gu, J. Vetter, and K. Schwan. An annotated bibliography of interactive program steering. *ACM Sigplan Notices*, 29(9):140 – 148, 1994.

[32] S. T. Gurumani and A. Milenkovic. Execution characteristics of spec cpu2000 benchmarks: Intel c++ vs. microsoft vc++. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 261–266, New York, NY, USA, 2004. ACM.

[33] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 276–285. Morgan Kaufmann, 1997.

[34] L. M. Haas, P. M. Schwarz, E. K. P. Kodali an, J. E. Rice, and W. C. Swope. Discoverylink: A system for integrated access to life sciences data sources. *IBM SYSTEMS JOURNAL*, 40(2):489, 2001.

[35] T. Hernandez and S. Kambhampati. Integration of biological sources: current systems and challenges ahead. In *ACM SIGMOD Conference*, volume 33, pages 51–60, Sept. 2004.

[36] B. Holldobler and C. J. Lumsden. Territorial Strategies in Ants. *Science*, 210(4471):732–739, 1980.

[37] P. Jorion. *Financial risk manager handbook*. John Wiley and Sons, 2007.

[38] Karimi and Rouhani. A new ant colony optimization based algorithm for data allocation problem in distributed databases. *Knowledge and Information Systems*.

[39] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 134–143, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[40] J. Knowles and D. Corne. *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*, chapter Bounded Pareto Archiving: Theory and Practice, pages 39–64. Springer, 2004.

[41] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, Dec. 2000.

[42] A. Labrinidis and N. Roussopoulos. Update propagation strategies for improving the quality of data on the web. In *VLDB*, pages 391–400, 2001.

[43] A. Labrinidis and N. Roussopoulos. Balancing performance and data freshness in web database servers. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 393–404. VLDB Endowment, 2003.

[44] G. T. Lakshmanan and R. E. Strom. Biologically-inspired distributed middleware management for stream processing systems. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 223–242, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[45] H. M. Levy and D. W. Clark. On the use of benchmarks for measuring system performance. *SIGARCH Comput. Archit. News*, 10(6):5–8, 1982.

[46] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In W. W. Chu, G. Gardarin, S. Ohsuga, and Y. Kambayashi, editors, *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*, pages 149–159. Morgan Kaufmann, 1986.

[47] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 49–60, New York, NY, USA, 2002. ACM.

[48] A. D. Malony and D. A. Reed. Models for performance perturbation analysis. pages 15 – 25, 1991.

[49] W. Maner. The unique ethical problems in information technology. *Science and Engineering Ethics*, 2:137–154, 1996.

[50] V. M. Markowitz and O. Ritter. Characterizing heterogeneous molecular biology database systems. *Journal of Computational Biology*, 2(4):547–56, 1995.

[51] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw Hill, 3 edition, 1991.

[52] M. Patino-Martinez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Middle-r: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.

[53] T. G. Robertazzi. *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1994.

[54] M. Rodriguez-Martinez and N. Roussopoulos. Mocha: A self-extensible database middleware system for distributed data sources. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 213–224. ACM, 2000.

[55] M. Rodrguez-Martnez. Nettraveler project description, 2005.

[56] U. Röhm, K. Böhm, H.-J. Schek, and H. Schuldt. Fas - a freshness-sensitive coordination middleware for a cluster of olap components. In *VLDB*, pages 754–765, 2002.

[57] M. T. Roth and P. M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 266–275. Morgan Kaufmann, 1997.

[58] K. M. Sim and W. H. Sun. Multiple ant-colony optimization for network routing. pages 277–281, 2002.

[59] K. M. Sim and W. H. Sun. Ant colony optimization for routing and load-balancing: survey and new directions. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 33(5):560–572, Sept. 2003.

[60] K. Socha and C. Blum. An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. *Neural Comput. Appl.*, 16(3):235–247, 2007.

[61] L. D. Stein. Integrating biological databases. *Nature Reviews Genetics*, 4(5):337–345, 2003.

[62] M. Stonebraker. The design and implementation of distributed ingres. pages 187–196, 1986.

[63] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063, 1996.

[64] M. Tamer Özsu and P. Valdurie. *Principles of Distributed Database Systems*. PrenticeHall, 1991.

[65] K. S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications*. John Wiley and Sons Ltd., Chichester, UK, UK, 2002.

[66] H.-L. Truong, T. Fahringer, and S. Dustdar. Dynamic instrumentation, performance modeling and analysis of grid scientific workflows. *Journal of Grid Computing*, 3:1 – 18, 2005.

[67] E. Valenzuela-Andrade and M. Rodriguez-Martinez. Autonomic ranking and characterization of data sources and query processing sites using ant colony theory. In *Third Asia International Conference on Modelling and Simulation*, Bali, Indonesia, 2009.

[68] E. A. Vargas-Figueroa and M. Rodriguez-Martinez. Design and implementation of the nettraveler middleware system based on web services. In *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, page 138, Washington, DC, USA, 2006. IEEE Computer Society.

[69] J. Vetter and K. Schwan. An annotated bibliography of interactive program steering. *IEEE Concurrency*, 7(4):63 – 74, Oct.-Dec. 1999.

[70] R. Williams, D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, and R. Yost. R*: An overview of the architecture. Technical report, IBM Almaden Research Center, San Jos, California, 1981.

[71] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Trans. Database Syst.*, 22(2):255–314, 1997.

[72] D. R. Wright. Research ethics and computer science: an unconsummated marriage. In *SIGDOC '06: Proceedings of the 24th annual ACM international conference on Design of communication*, pages 196–201, New York, NY, USA, 2006. ACM.

[73] D. R. Wright. Motivation, design, and ubiquity: A discussion of research ethics and computer science. *CoRR*, abs/0706.0484, 2007.

[74] V. Zadorozhny, A. Gal, L. Raschid, and Q. Ye. Arena: Adaptive distributed catalog infrastructure based on relevance networks. In *VLDB*, pages 1287–1290, 2005.

APPENDICES

APPENDIX A

Simulation Parameters

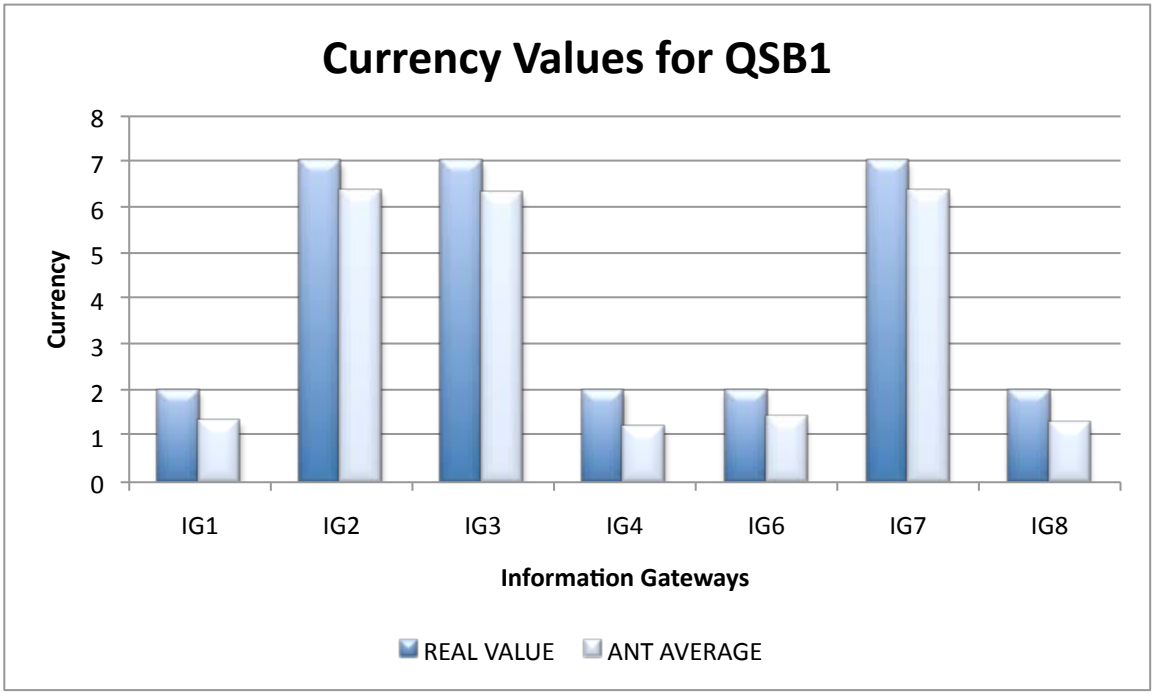
```
/*************************************************************************
 *     NetTraveler Project - Ant System Approach
 *                          Copyright (C) 2007
 *    Eliana Valenzuela-Andrade                Manuel Rodriguez-Martinez
 *                 University of Puerto Rico, Mayaguez
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
 * 02110-1301, USA.
 ********************************************************************/
package edu.uprm.admg.nettraveler.ants.CSIM;

public class Constants_CSIM {
public final static String DATABASE = "PAPERMODEL";
public final static String DATABASELOGIN = "root";
public final static String DATABASEPASS = "";

public static final String runNumber = "1";
public static long seed = 19721106;
//public static final String runNumber = "2";
//public static long seed = 91287364;
//public static final String runNumber = "3";
//public static long seed = 20002008;
//public static final String runNumber = "4";
//public static long seed = 2345678;
//public static final String runNumber = "5";
//public static long seed = 324234;
//public static final String runNumber = "6";
//public static long seed = 47854151;
//public static final String runNumber = "7";
//public static long seed = 1247825;
//public static final String runNumber = "8";
//public static long seed = 9874521;
```

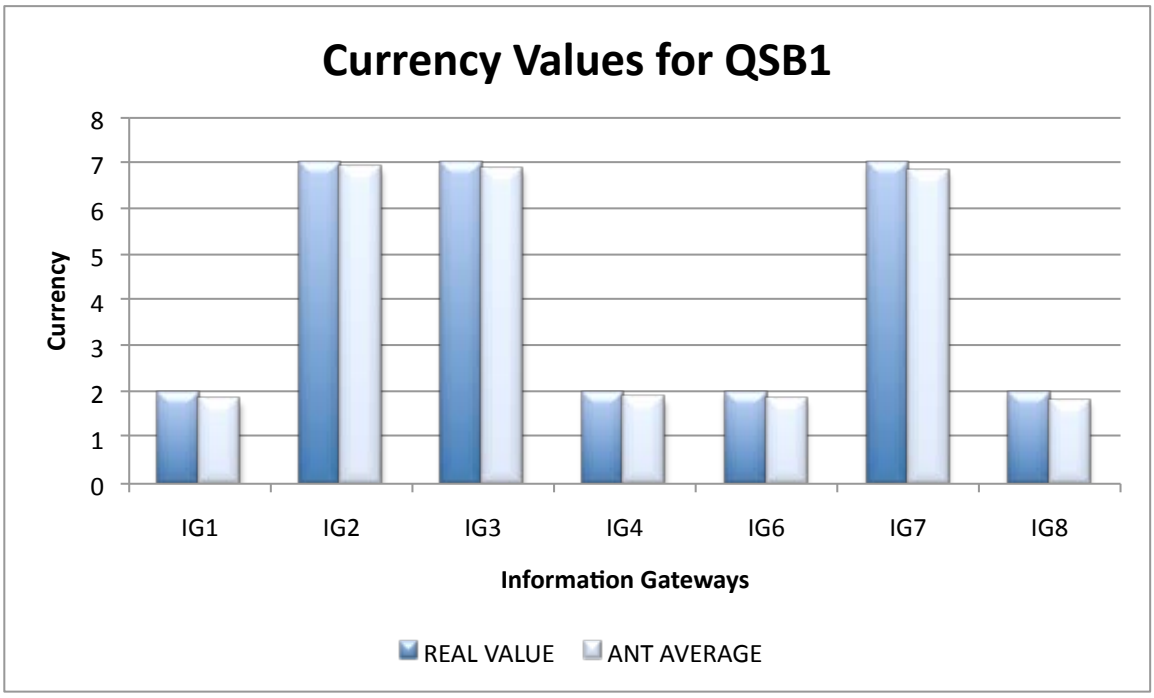
```
//public static final String runNumber = "9";
//public static long seed = 541247;
//public static final String runNumber = "10";
//public static long seed = 0611172;

// Define initial values for statistics
public static final double INITIAL_VARIANCE = 1.0;
static final double ALPHA = 0.5;
static final double ALPHA_IND = 0.15;
public static final double C1 = 0.35;
public static final double C2 = 0.25;
public static final double CONFIDENCE_LEVEL = 0.90;
public static final double CONFIDENCE_LEVEL_IND = 0.90;
public static final double A = 10.0;
public static final int MAX_LOOPS = 5;
public static final double EVAPORATION = -0.35;
public static final double VARHPEXP = 2.0;
public static final double VAR_PATHS = 1.0;

public static final boolean LazyAnts =false;
public static final double THFresh =0.5;
public static final double kparameterNetwork_pareto = 0.58;
public static final double kparameterCPU_pareto = 0.60;
public static final double kparameterNetwork_paretobounded = 1.1;
public static final double kparameterCPU_paretobounded = 1.2;
}
```

APPENDIX B

Other Freshness Results

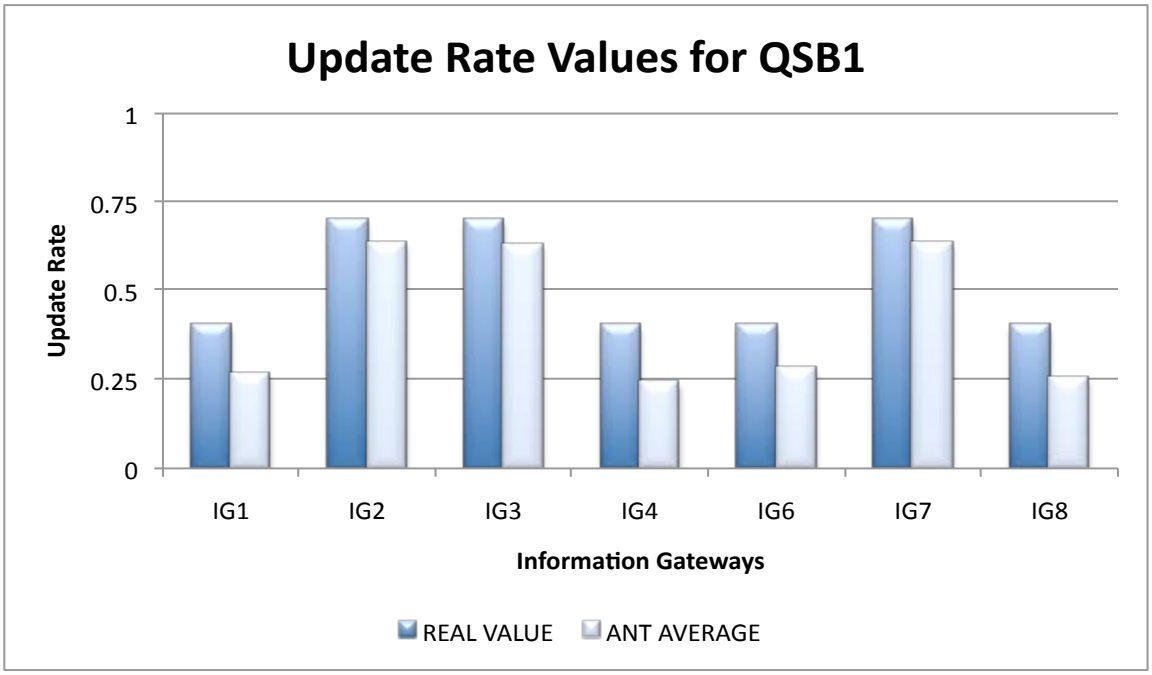


(a) One ant every half-hour.

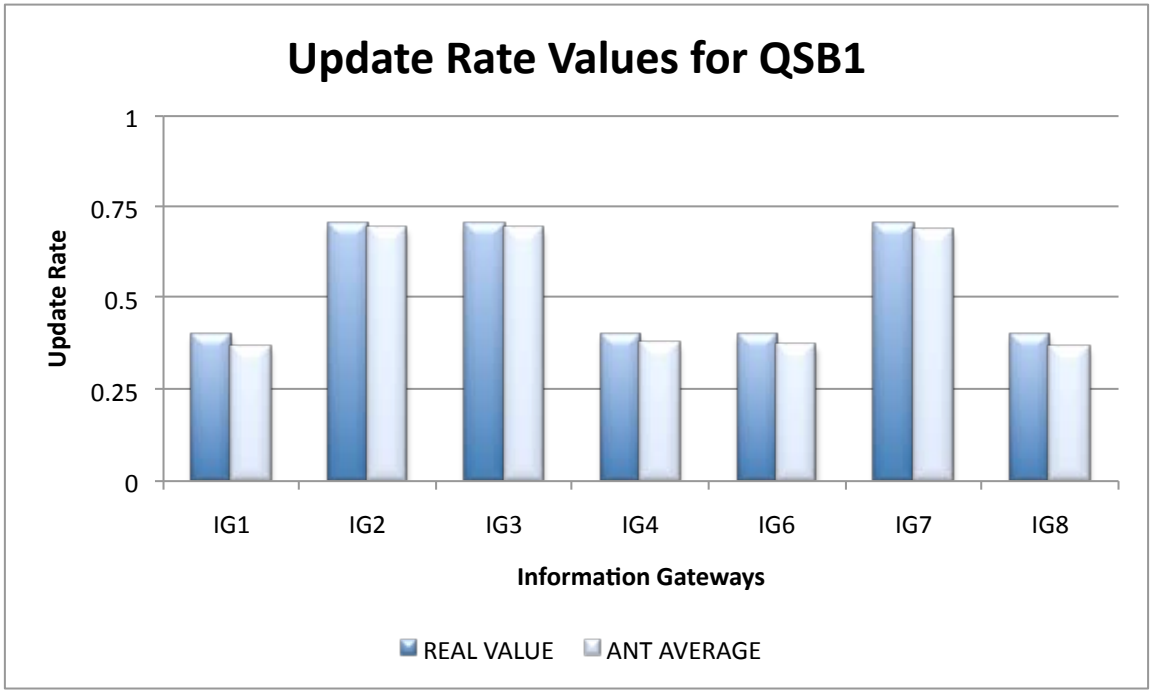


(b) One ant every two hours.

Figure B.1. Currency Metric behavior seen by QSB_1.

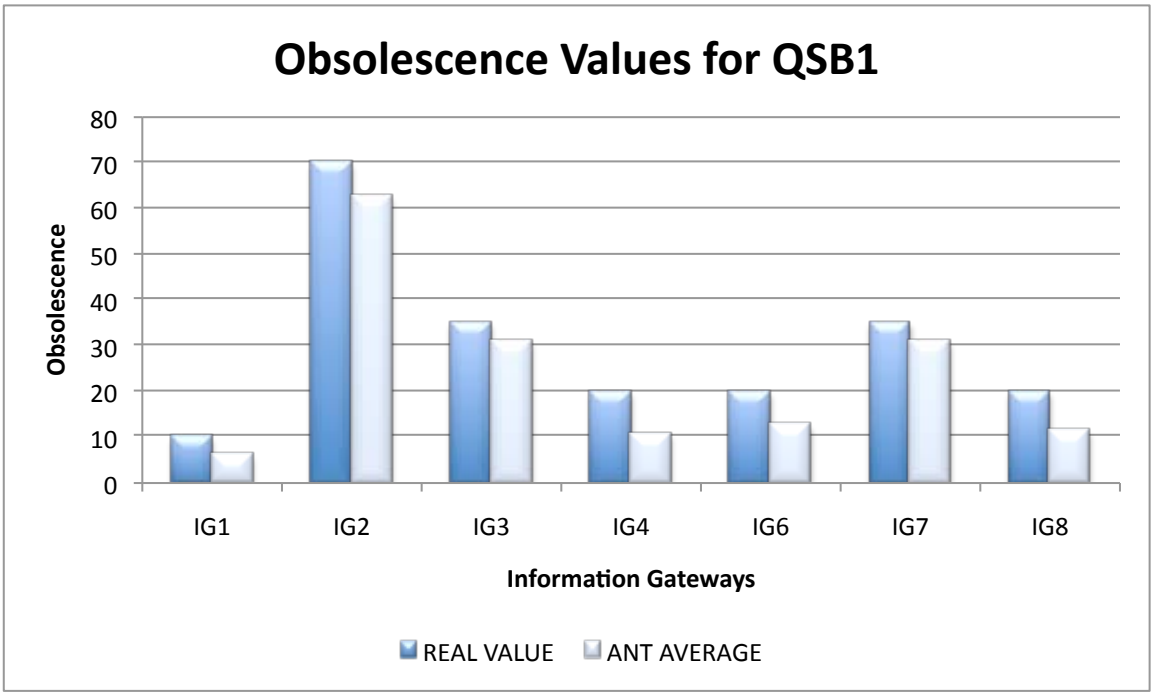


(a) One ant every half-hour.

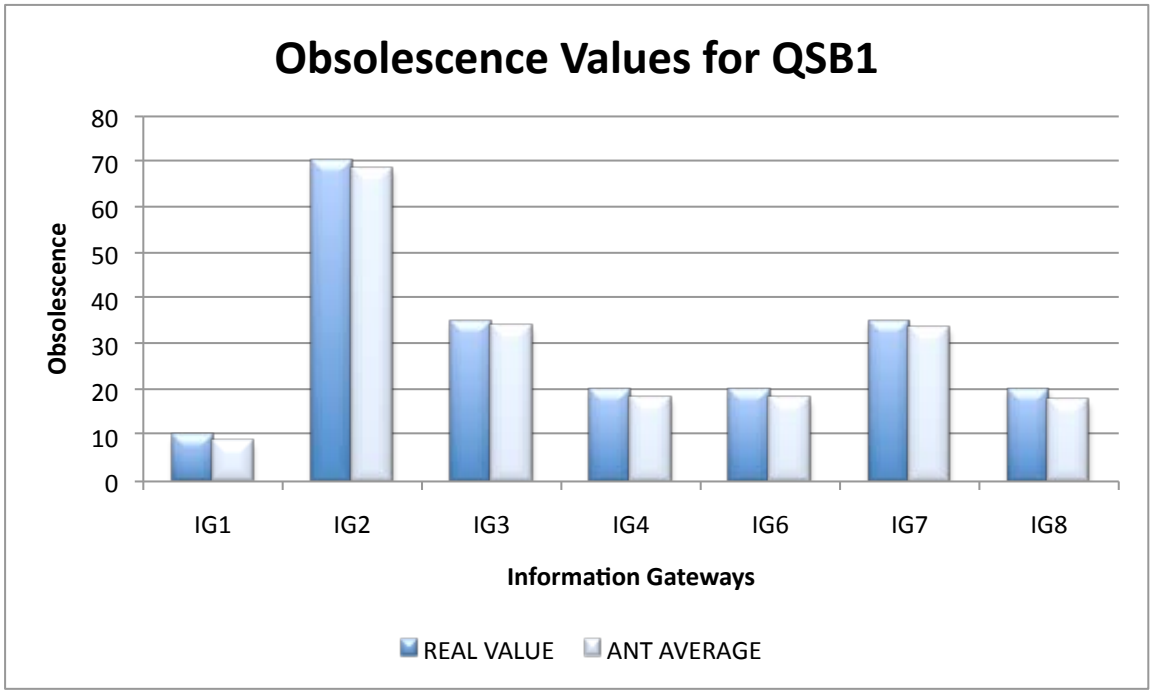


(b) One ant every two hours.

Figure B.2. Rate Metric behavior seen by QSB_1.

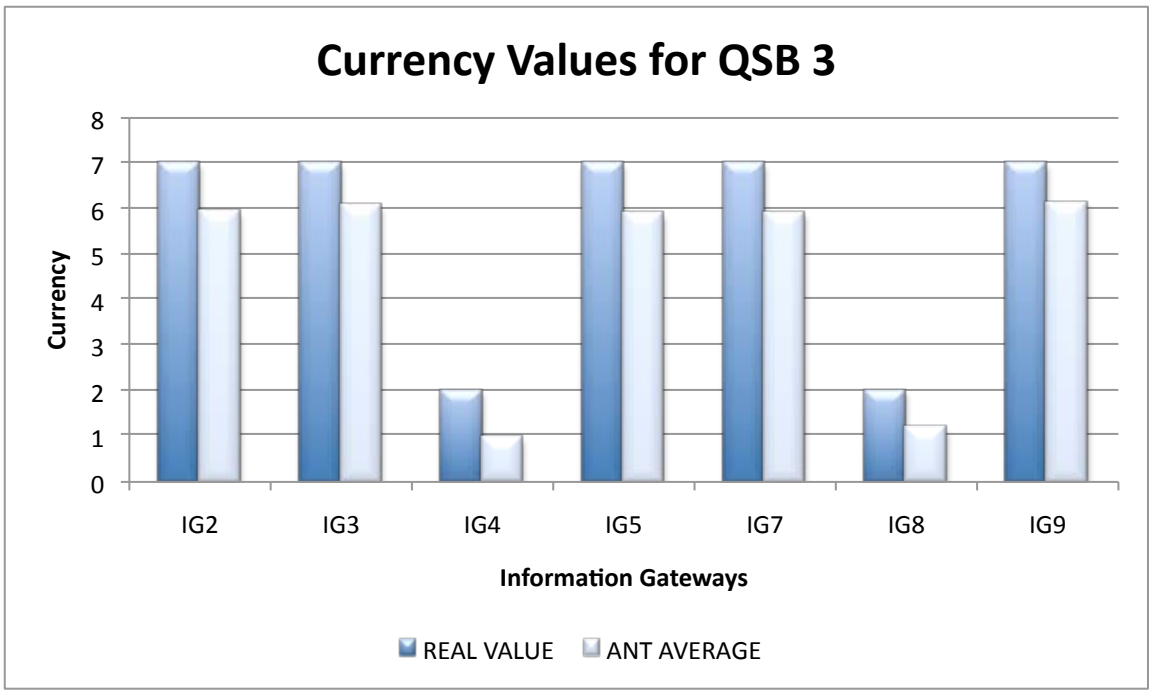


(a) One ant every half-hour.

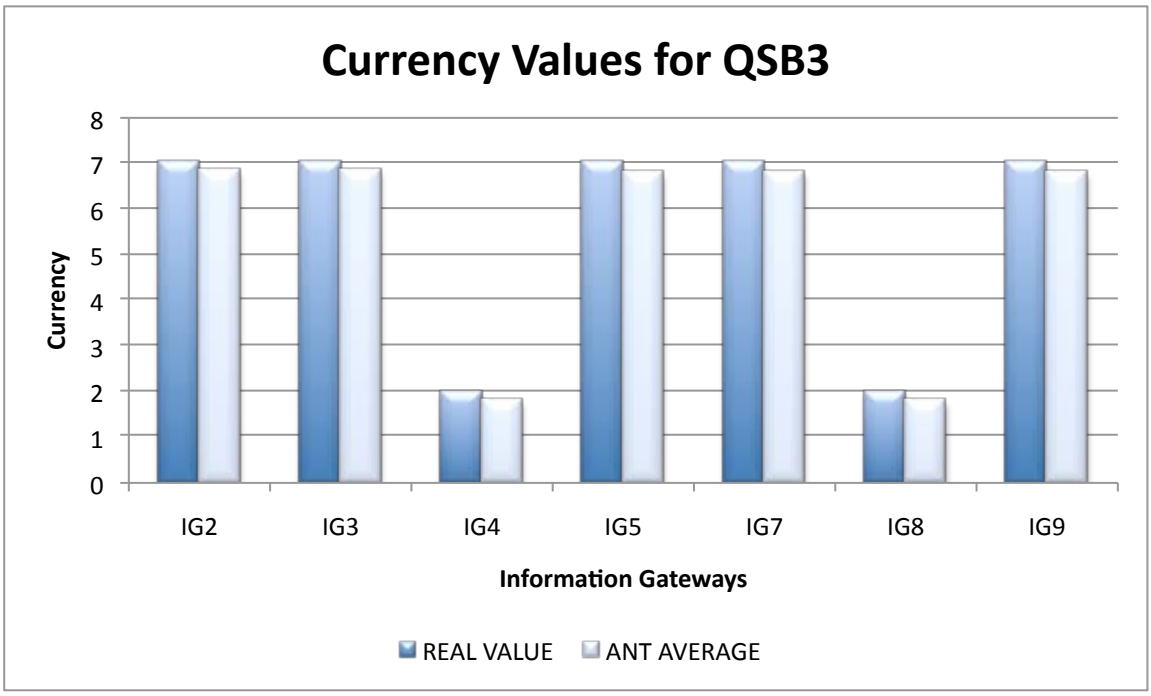


(b) One ant every two hours.

Figure B.3. Obsolescence Metric behavior seen by QSB_1.

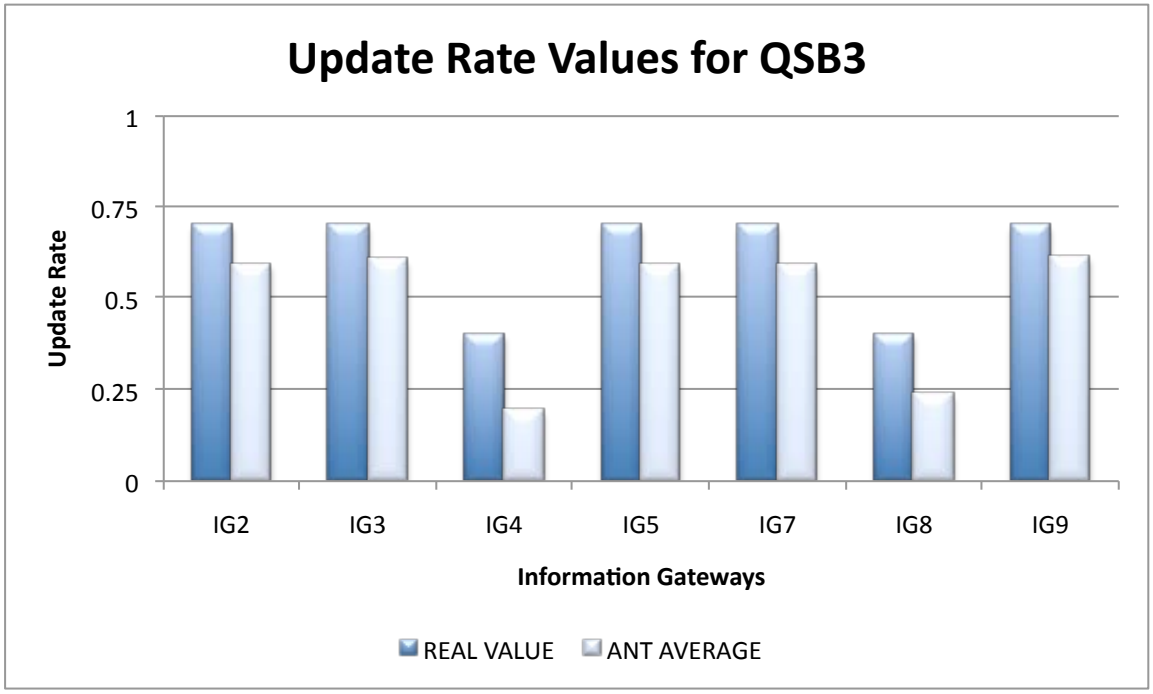


(a) One ant every half-hour.

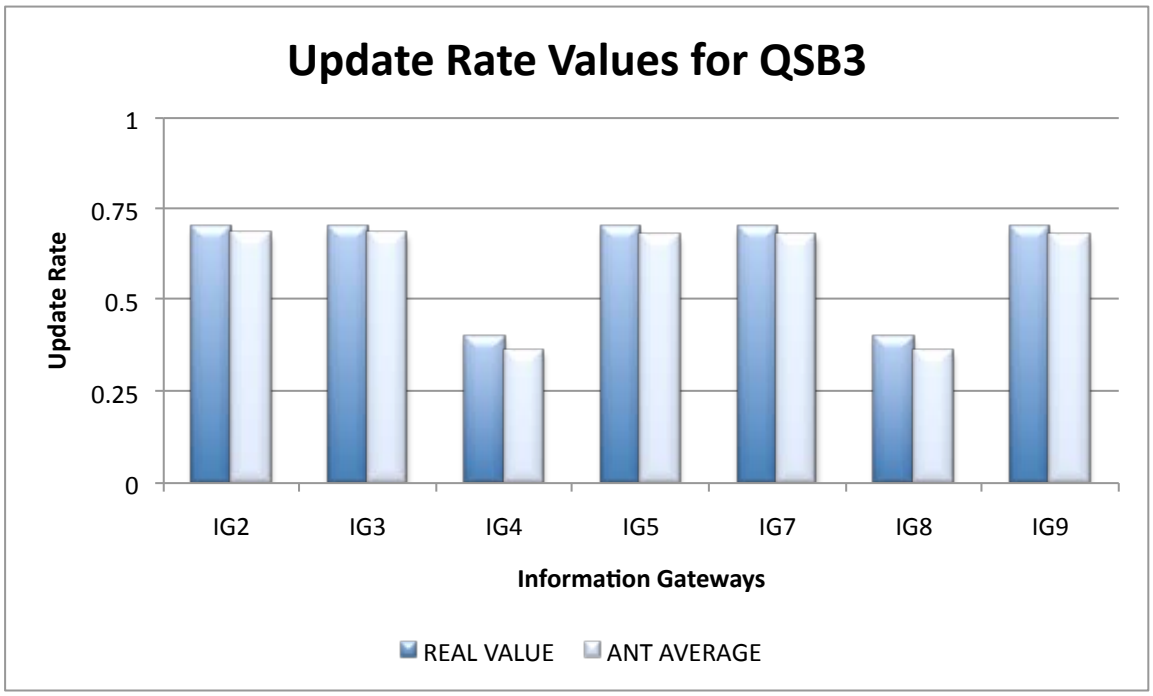


(b) One ant every two hours.

Figure B.4. Currency Metric behavior seen by QSB_3.

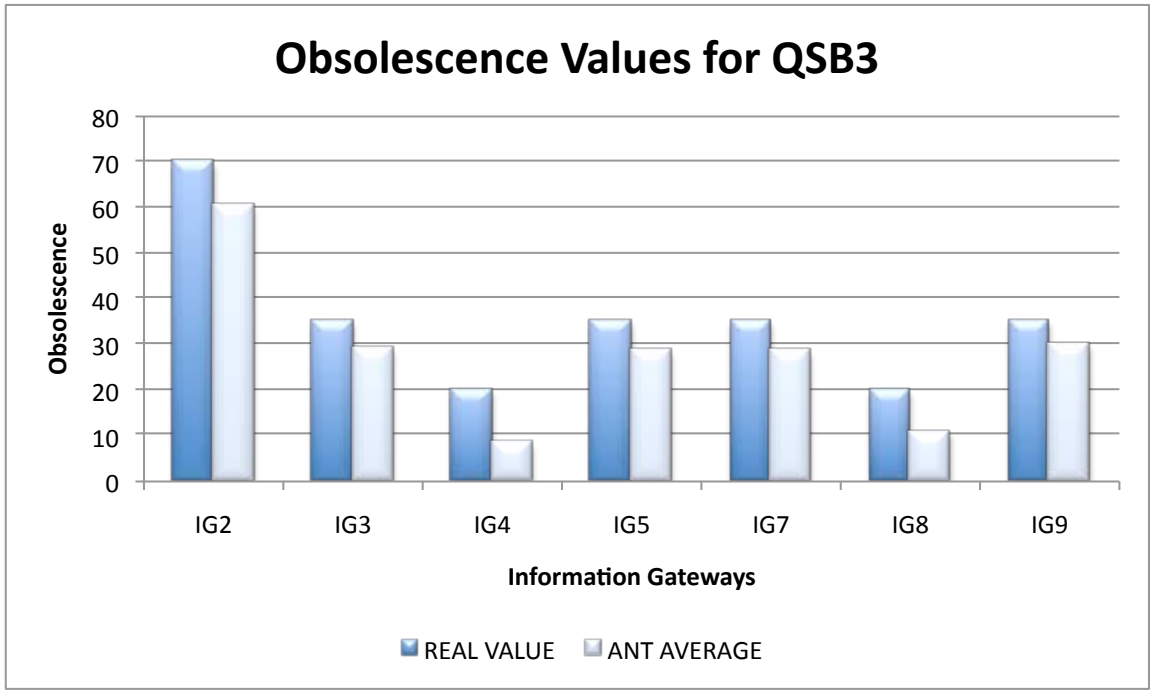


(a) One ant every half-hour.

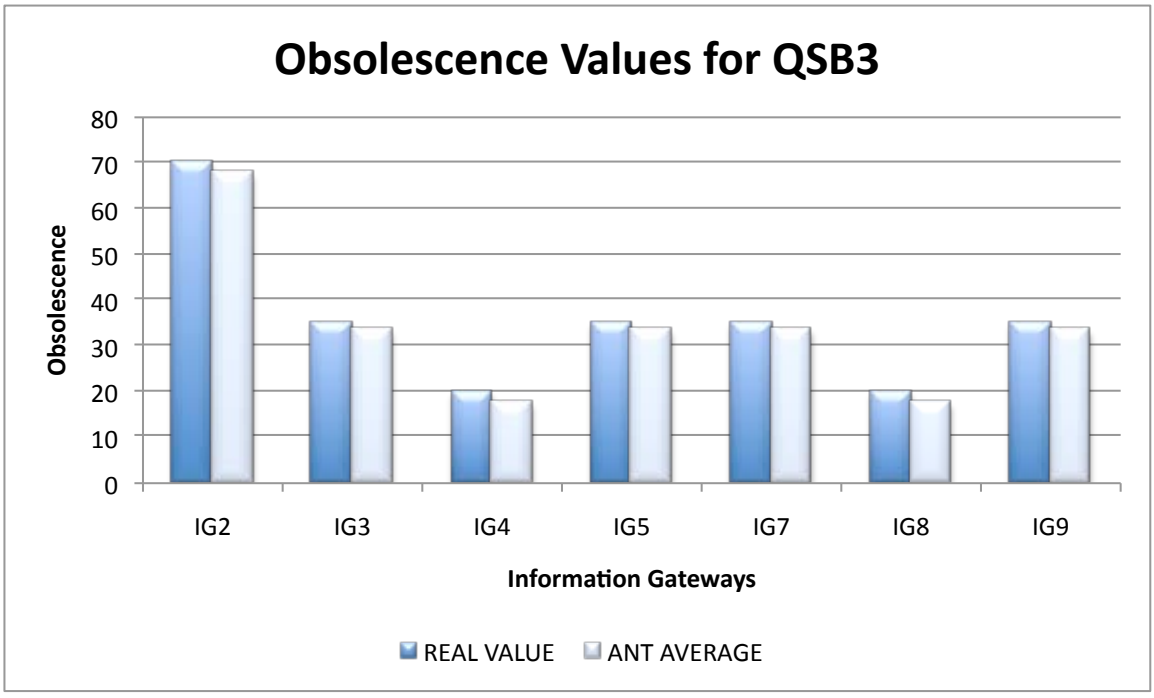


(b) One ant every two hours.

Figure B.5. Rate Metric behavior seen by QSB_3.

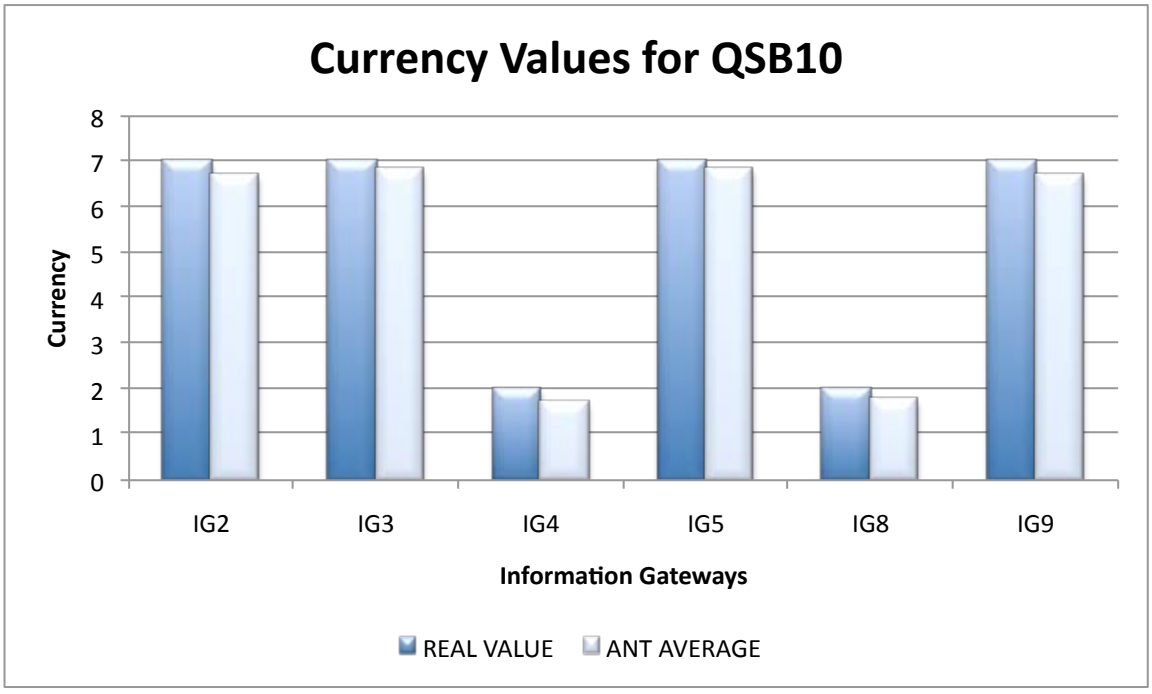


(a) One ant every half-hour.

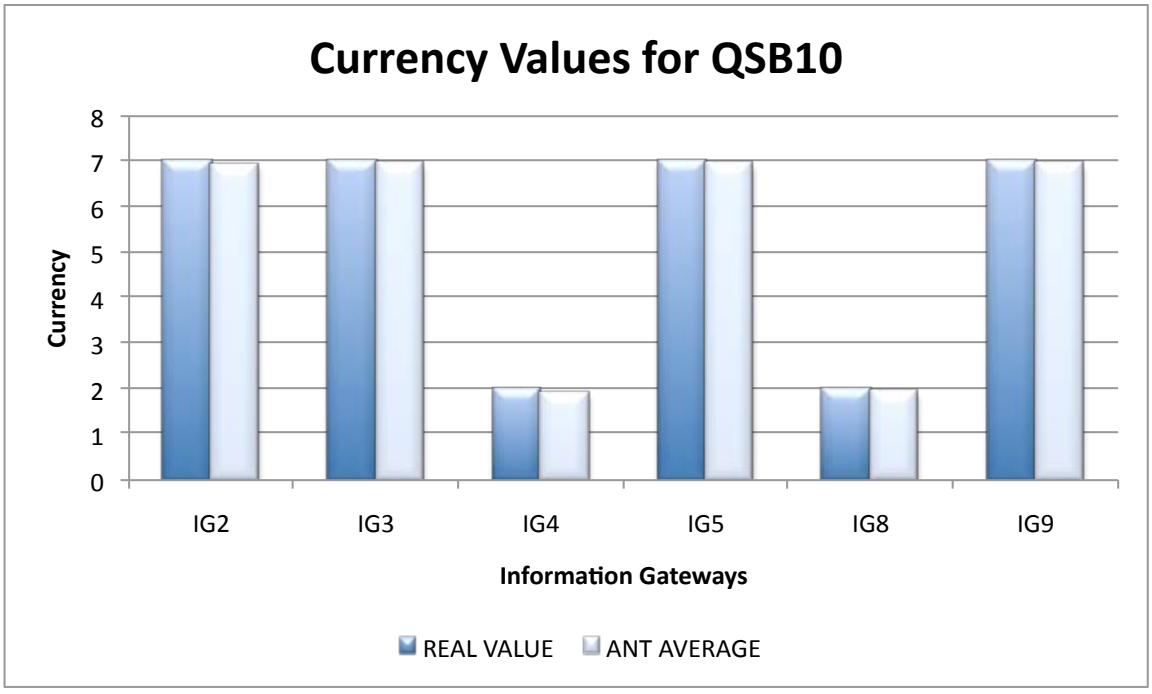


(b) One ant every two hours.

Figure B.6. Obsolescence Metric behavior seen by QSB_3.

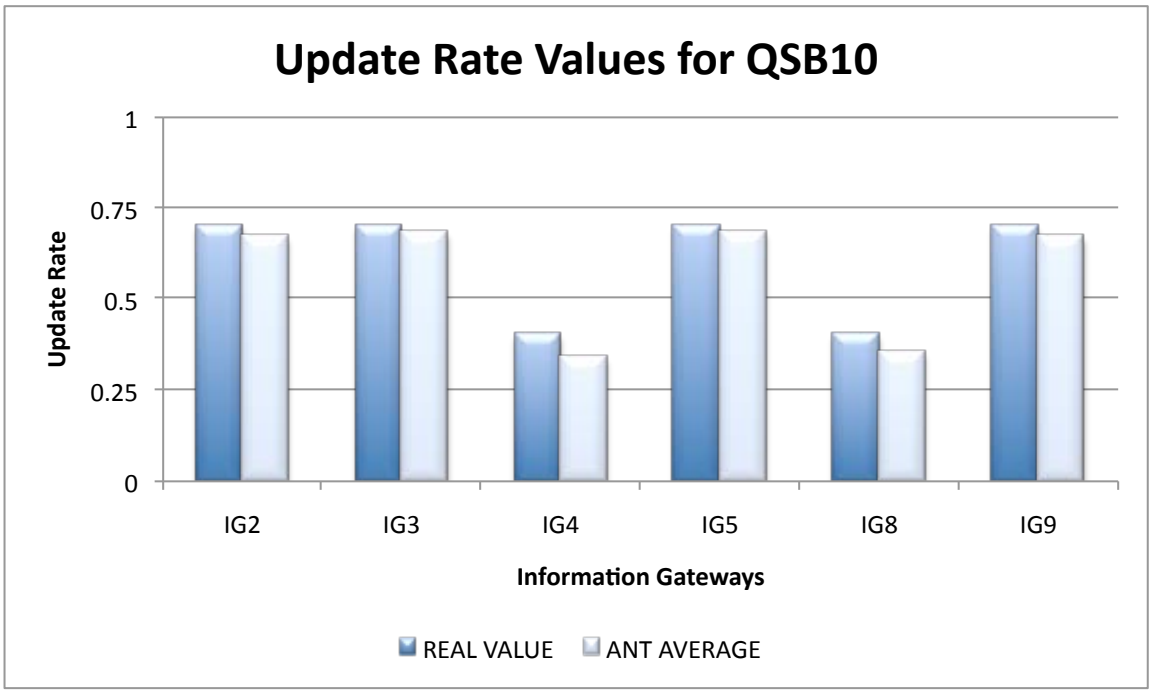


(a) One ant every half-hour.

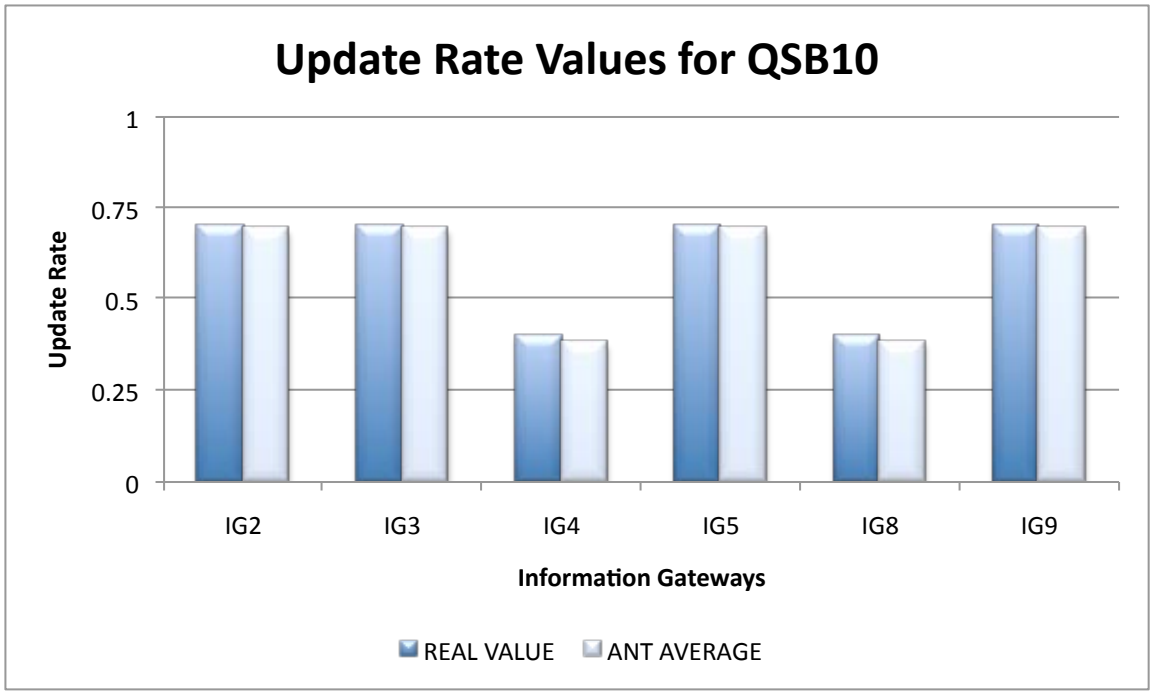


(b) One ant every two hours.

Figure B.7. Currency Metric behavior seen by QSB_{10}.

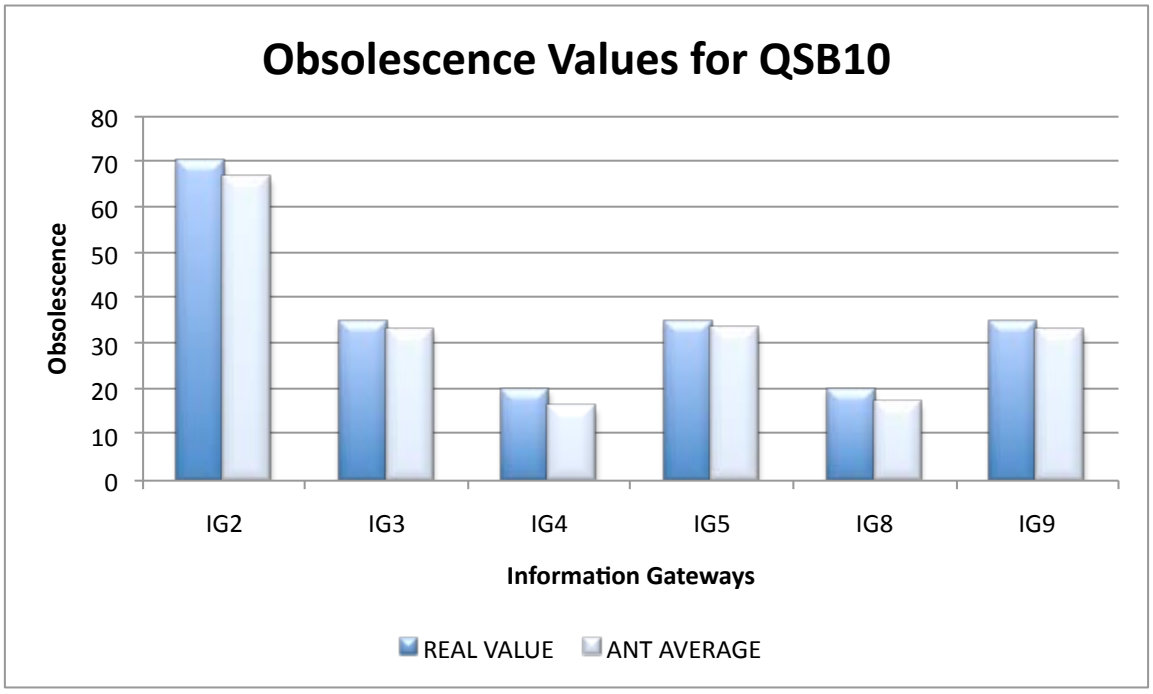


(a) One ant every half-hour.

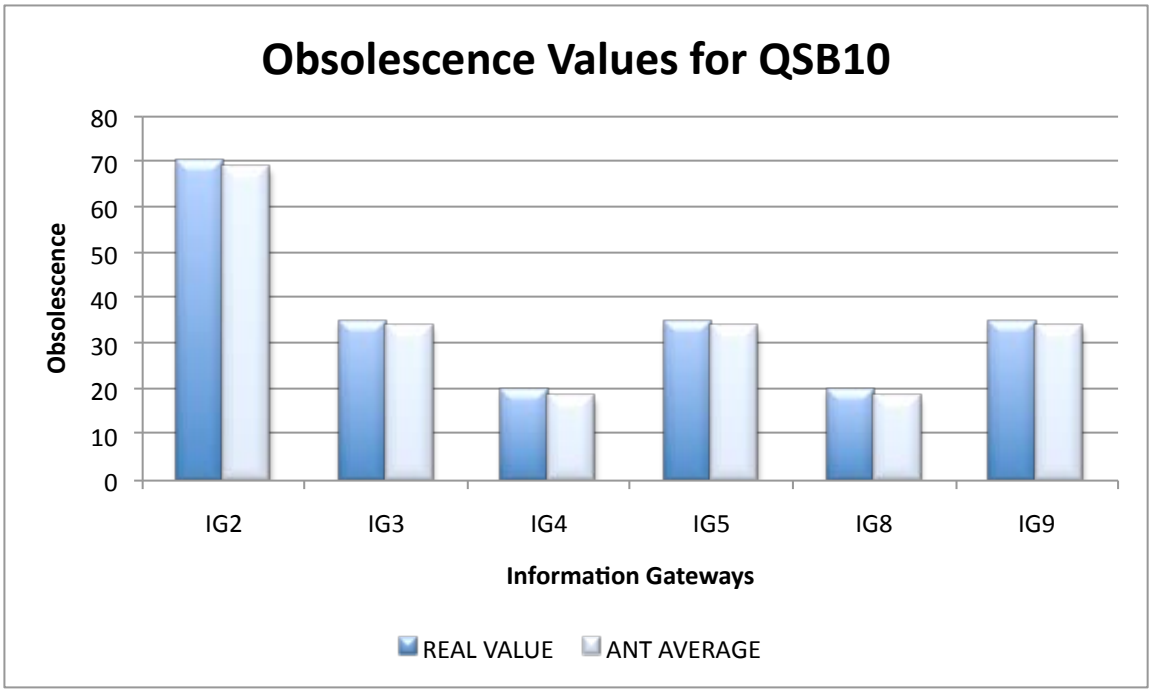


(b) One ant every two hours.

Figure B.8. Rate Metric behavior seen by QSB_{10}.

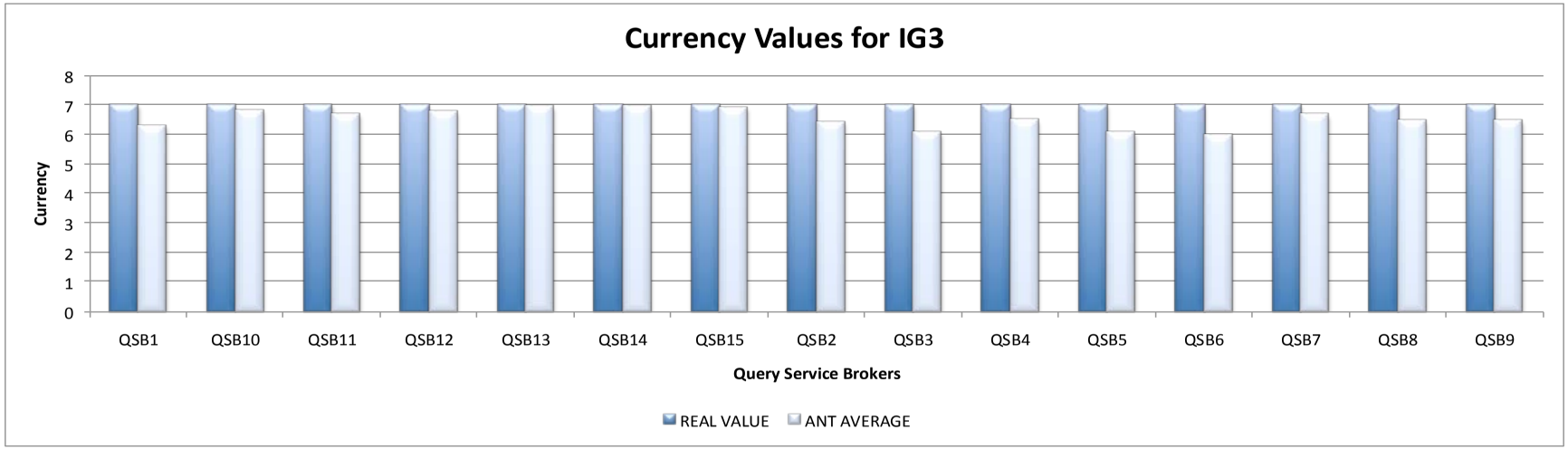


(a) One ant every half-hour.

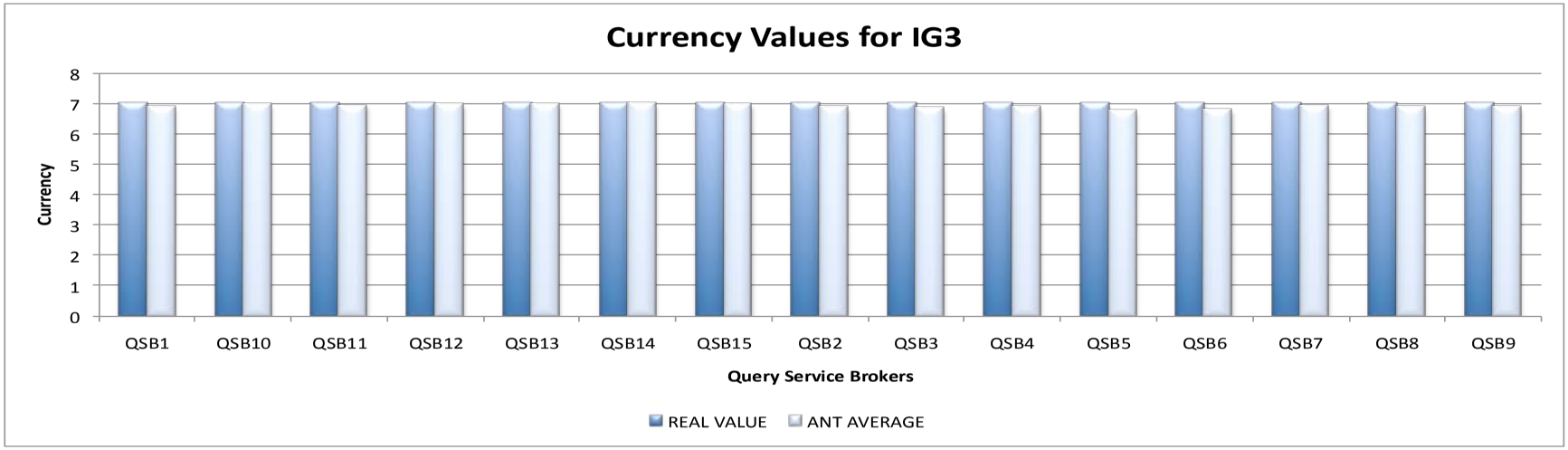


(b) One ant every two hours.

Figure B.9. Obsolescence Metric behavior seen by QSB_{10}.

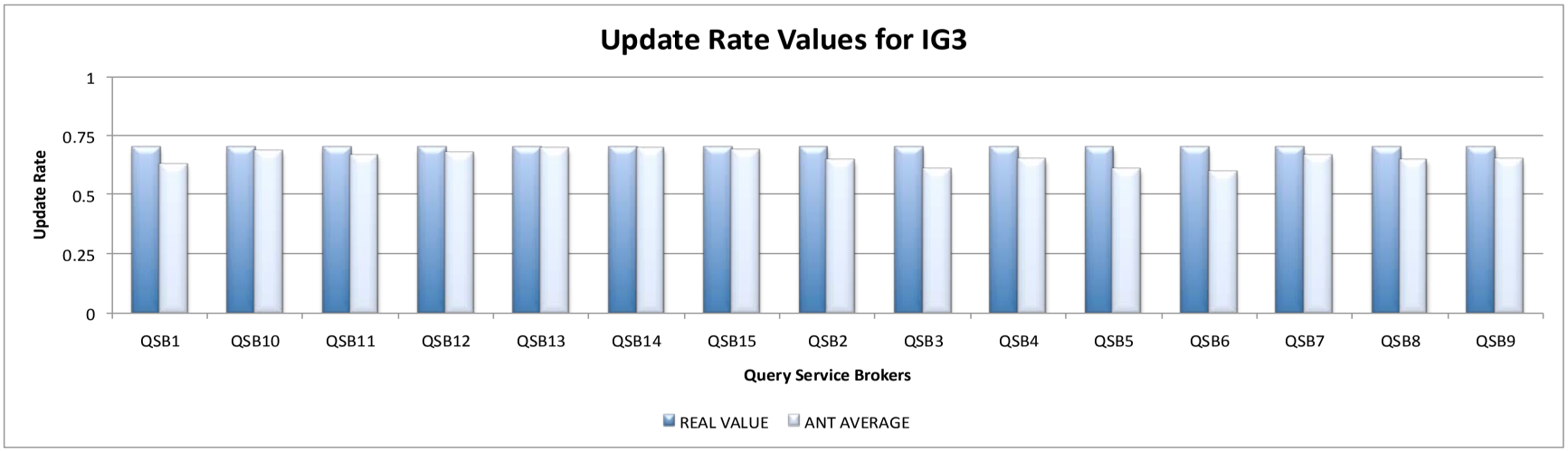


(a) One ant every half-hour.

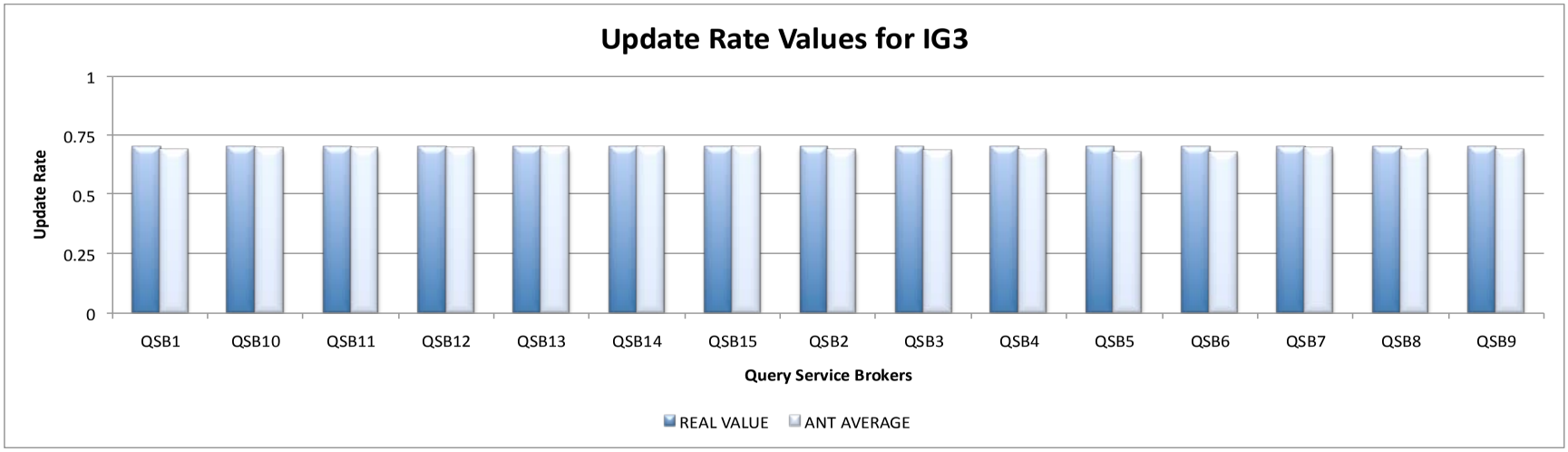


(b) One ant every two hours.

Figure B.10. Currency Metric for IG_3 as seen by the system.

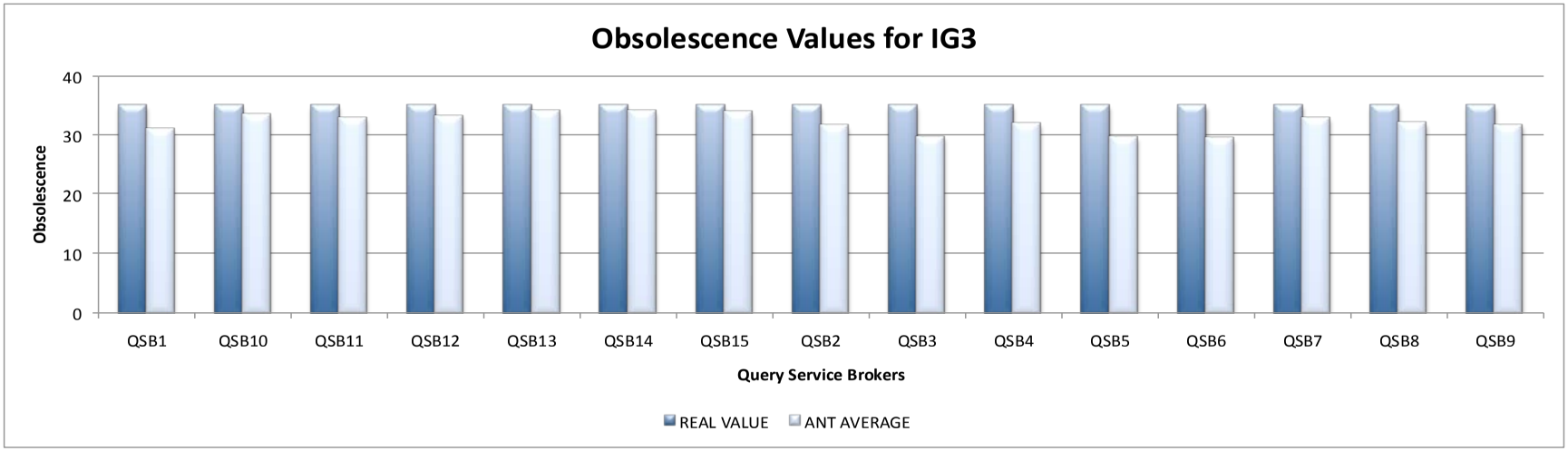


(a) One ant every half-hour.

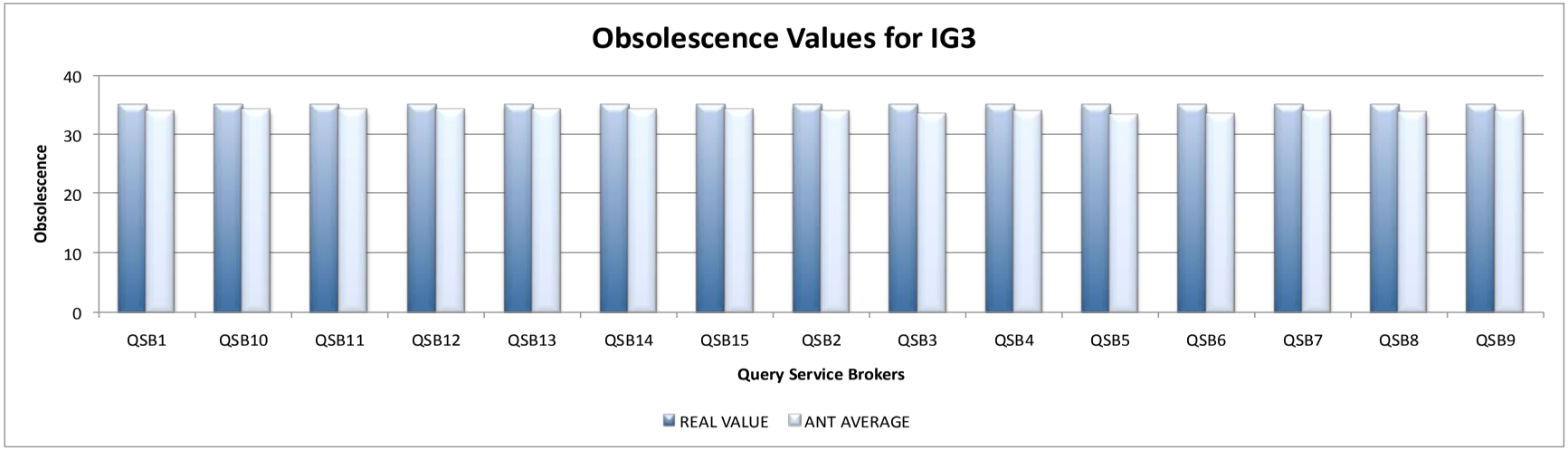


(b) One ant every two hours.

Figure B.11. Rate for IG_3 as seen by the system.

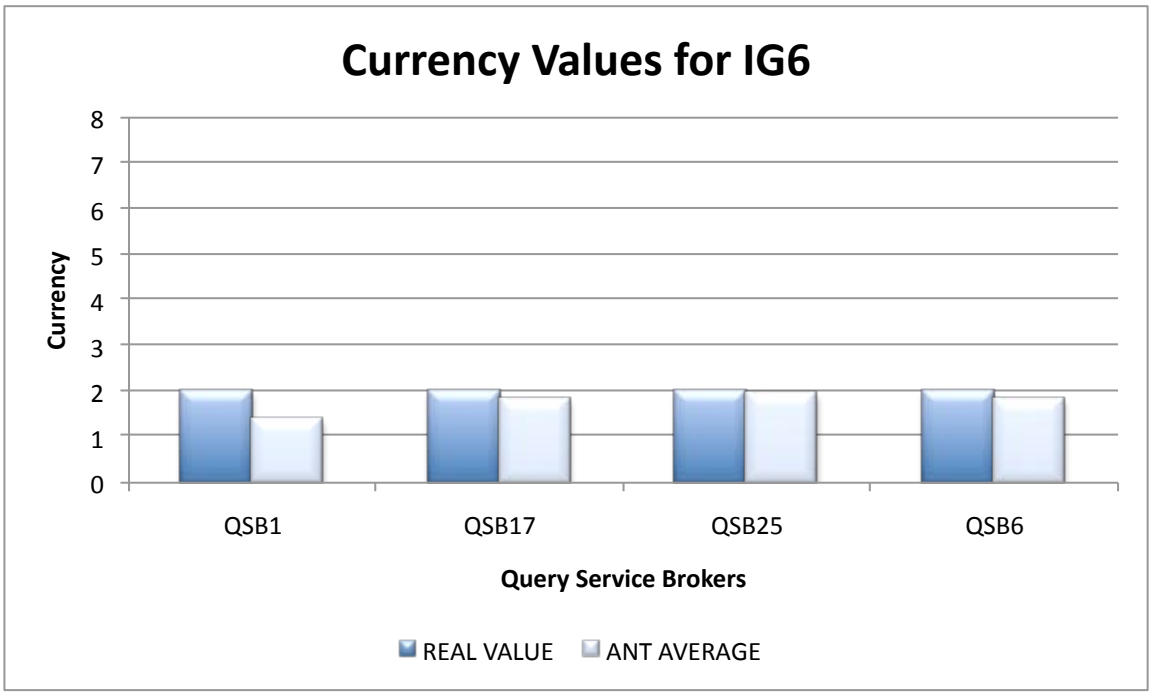


(a) One ant every half-hour.

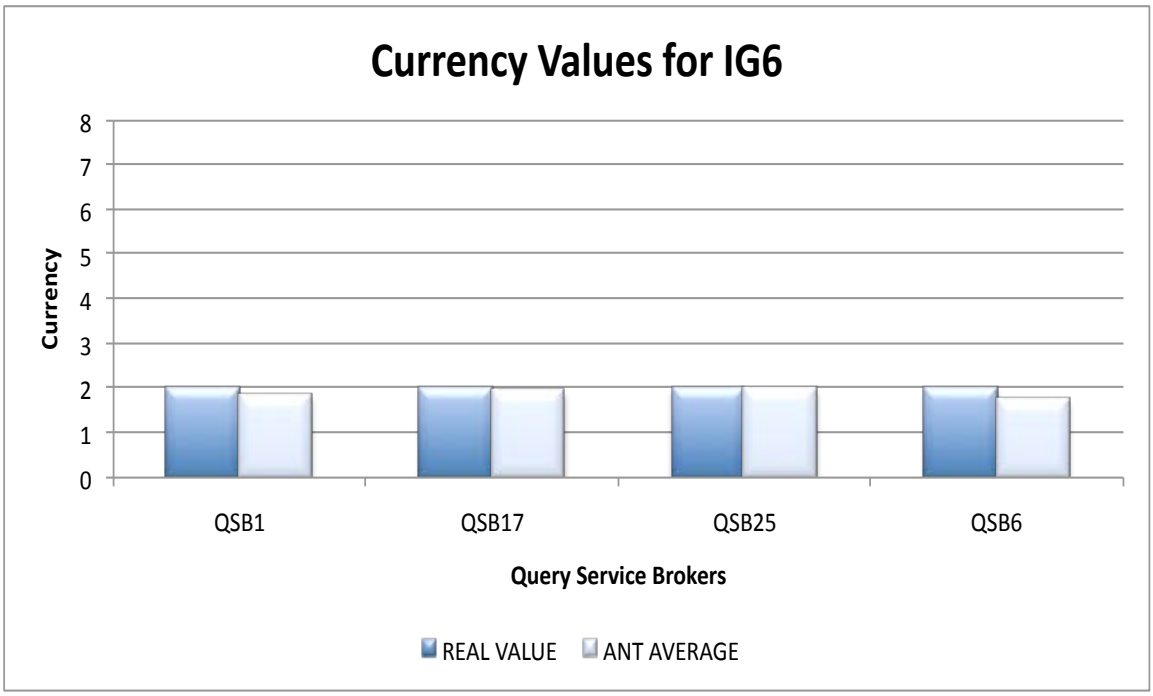


(b) One ant every two hours.

Figure B.12. Data Freshness for IG_3 as seen by the system.

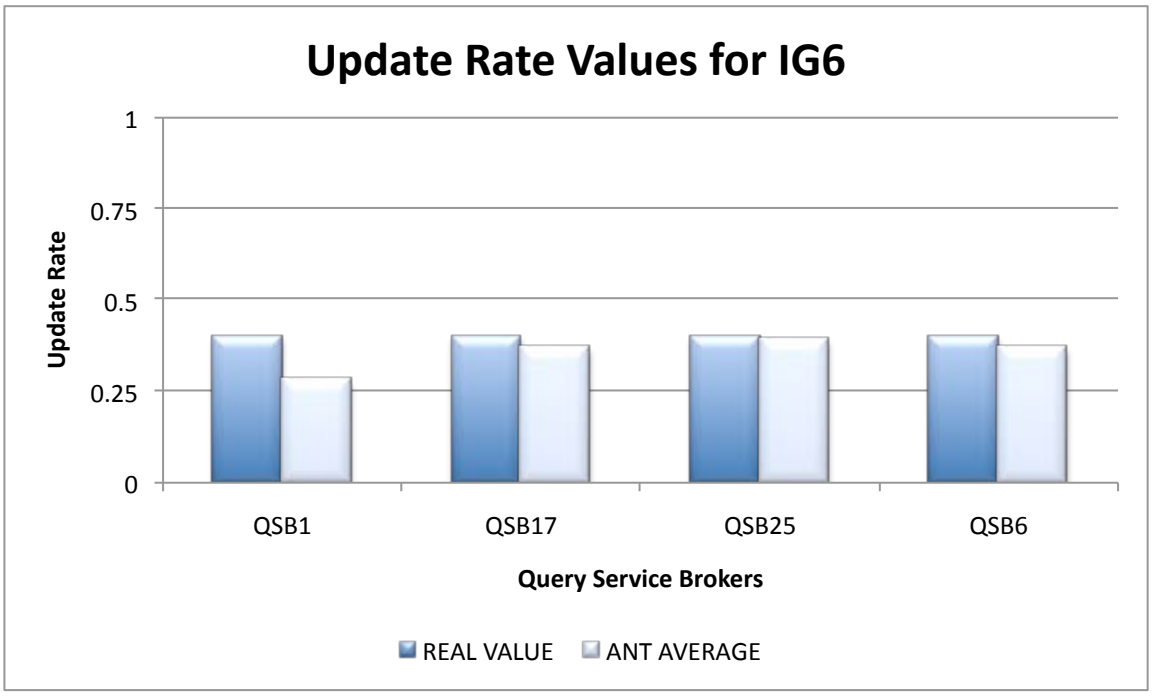


(a) One ant every half-hour.

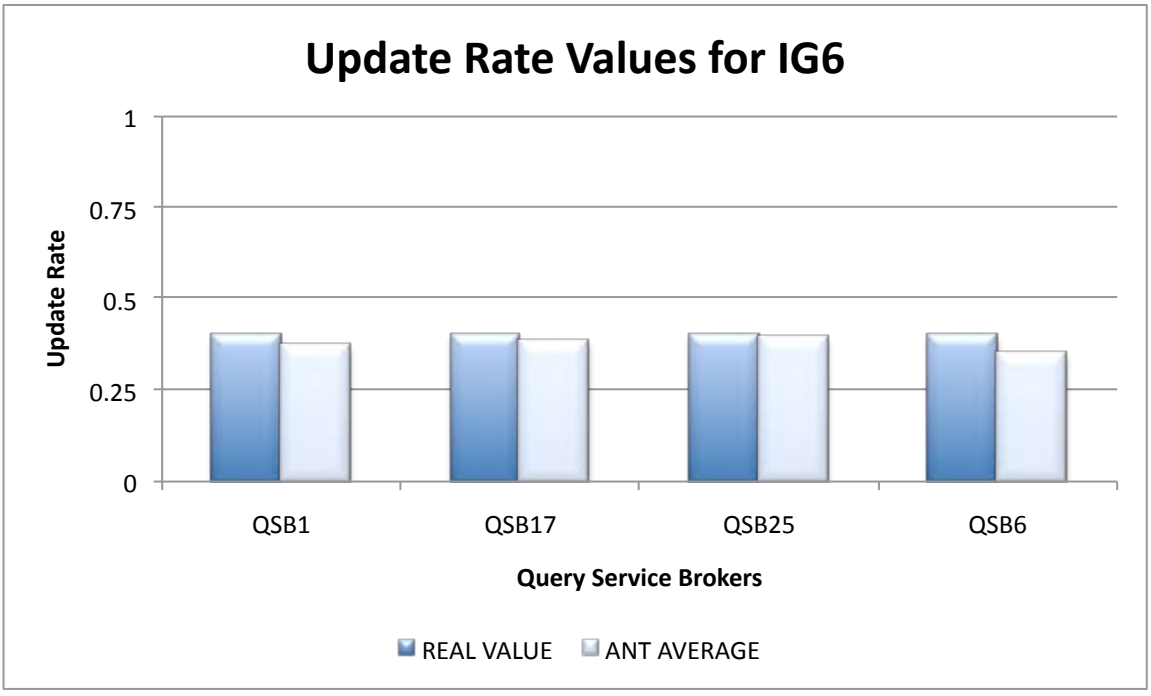


(b) One ant every two hours.

Figure B.13. Currency Metric for IG_6 as seen by the system.

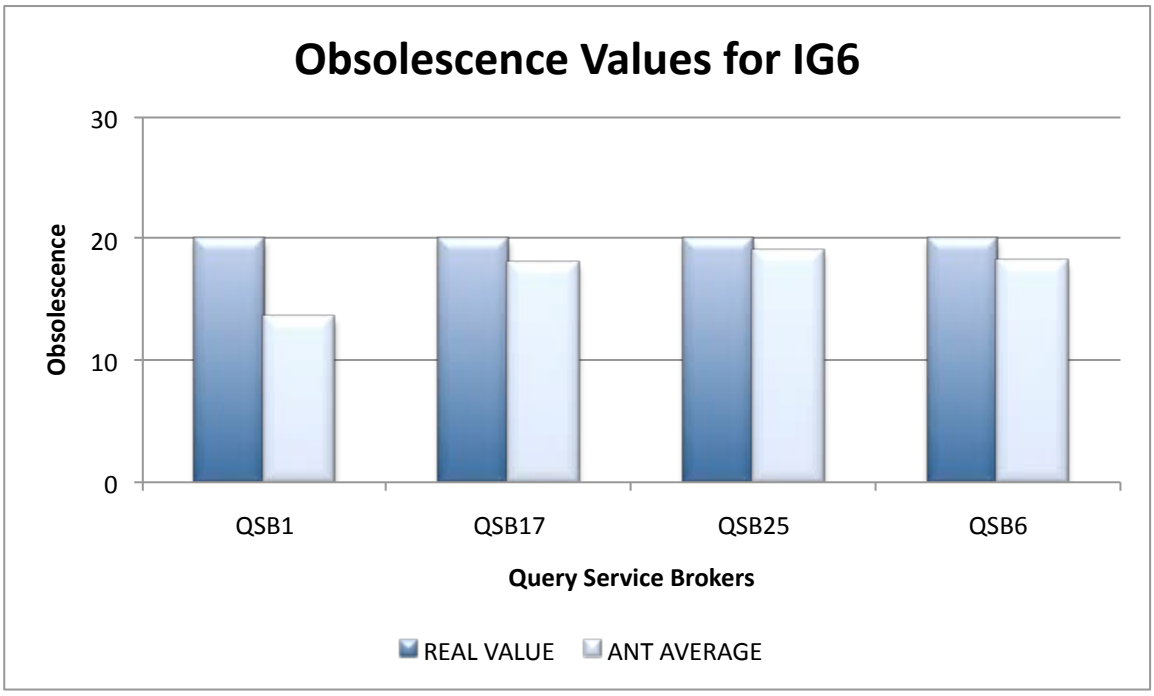


(a) One ant every half-hour.

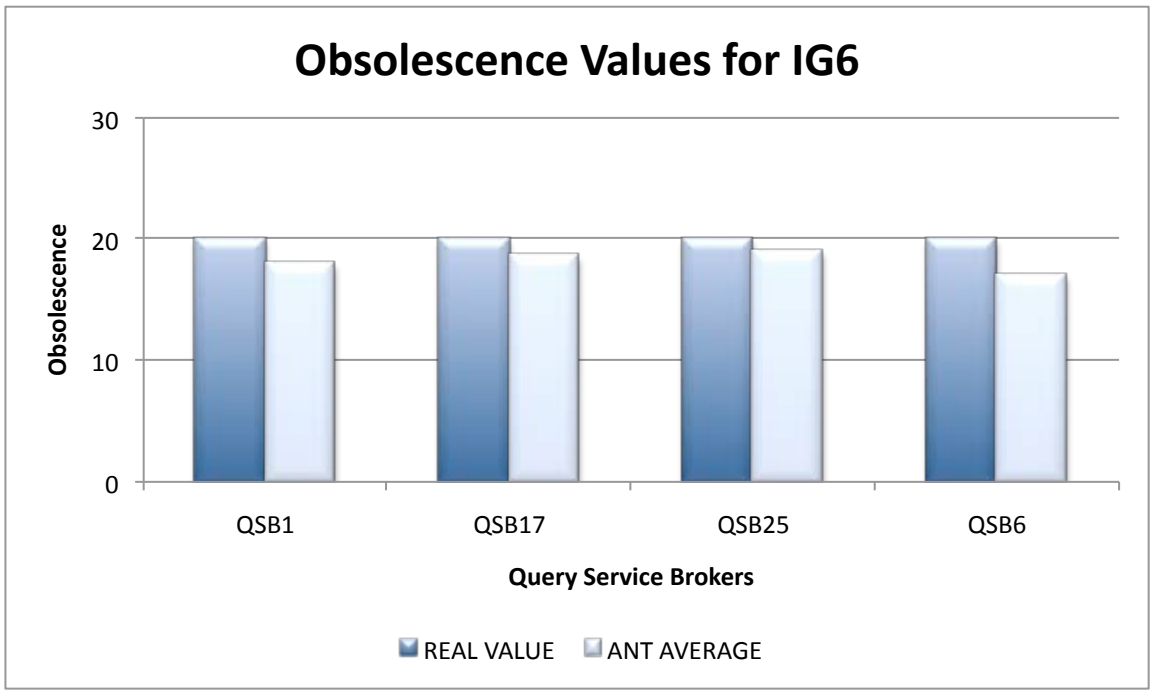


(b) One ant every two hours.

Figure B.14. Rate for IG_6 as seen by the system.

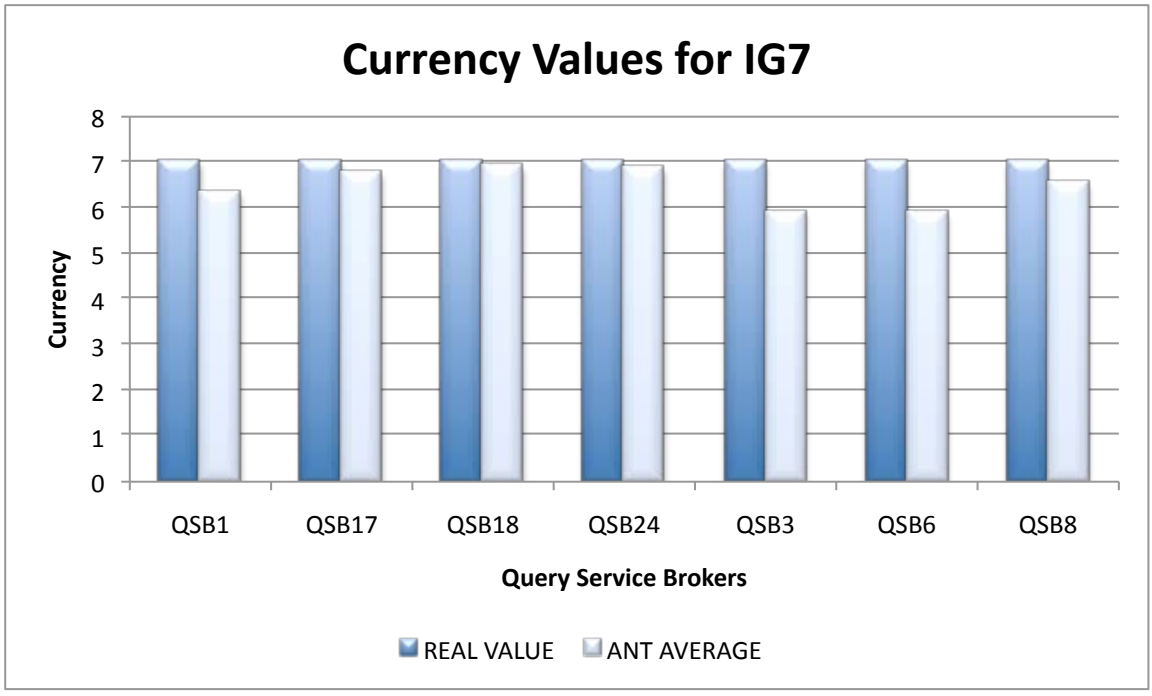


(a) One ant every half-hour.

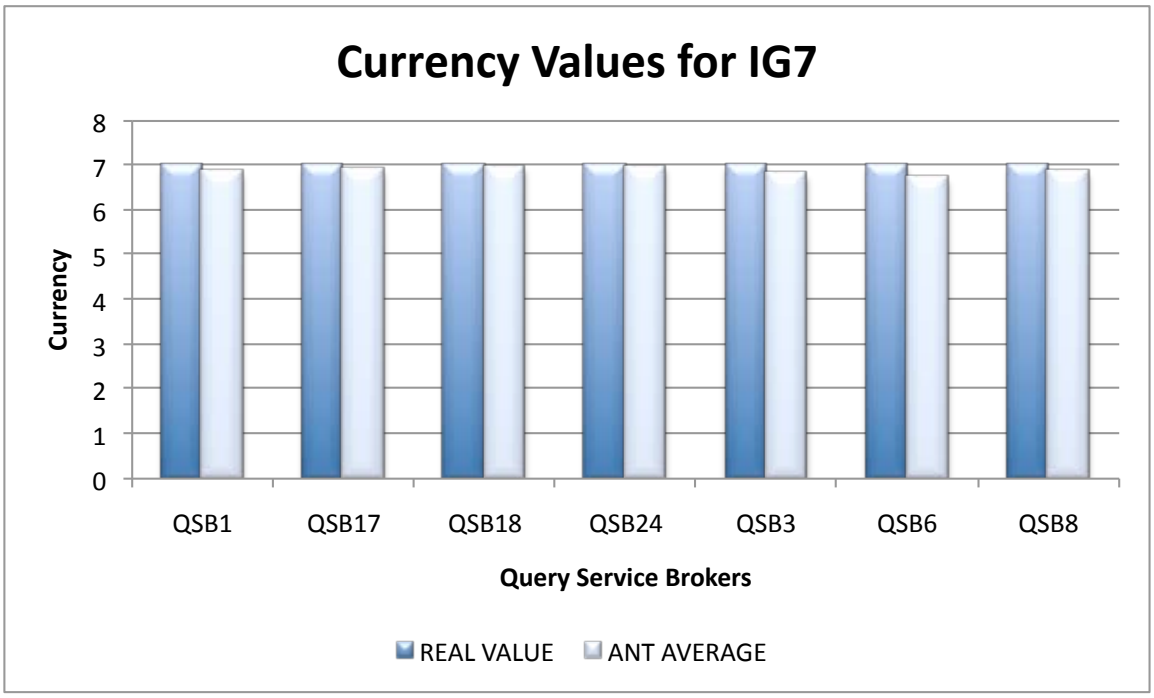


(b) One ant every two hours.

Figure B.15. Data Freshness for IG_6 as seen by the system.

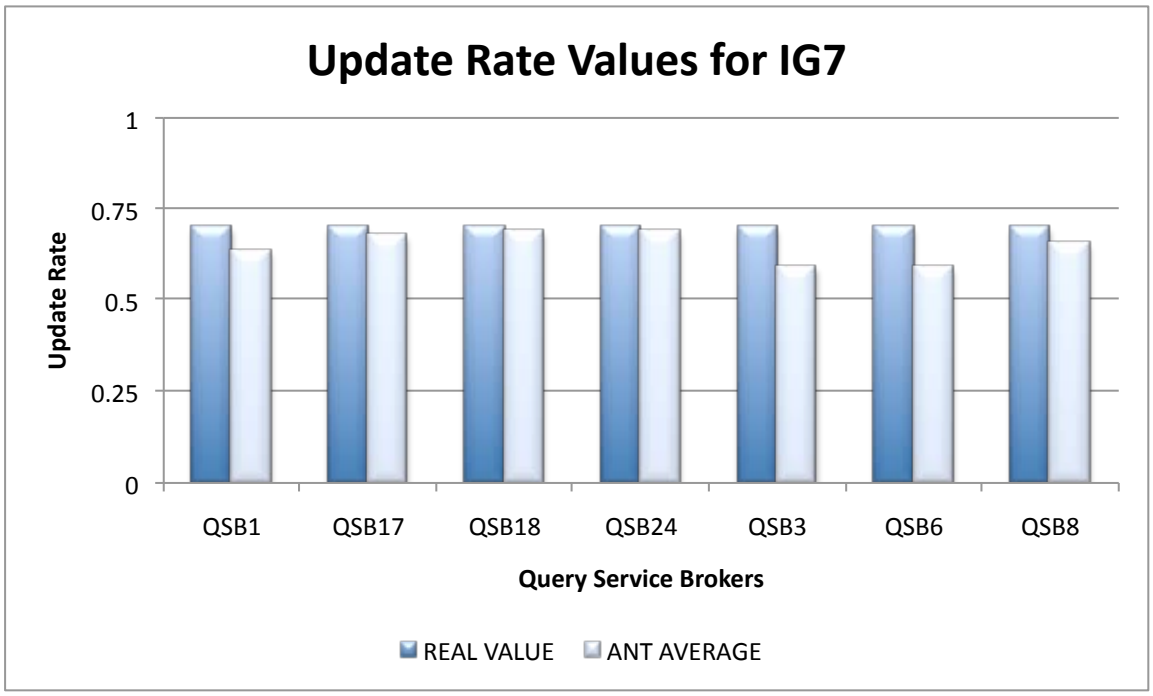


(a) One ant every half-hour.

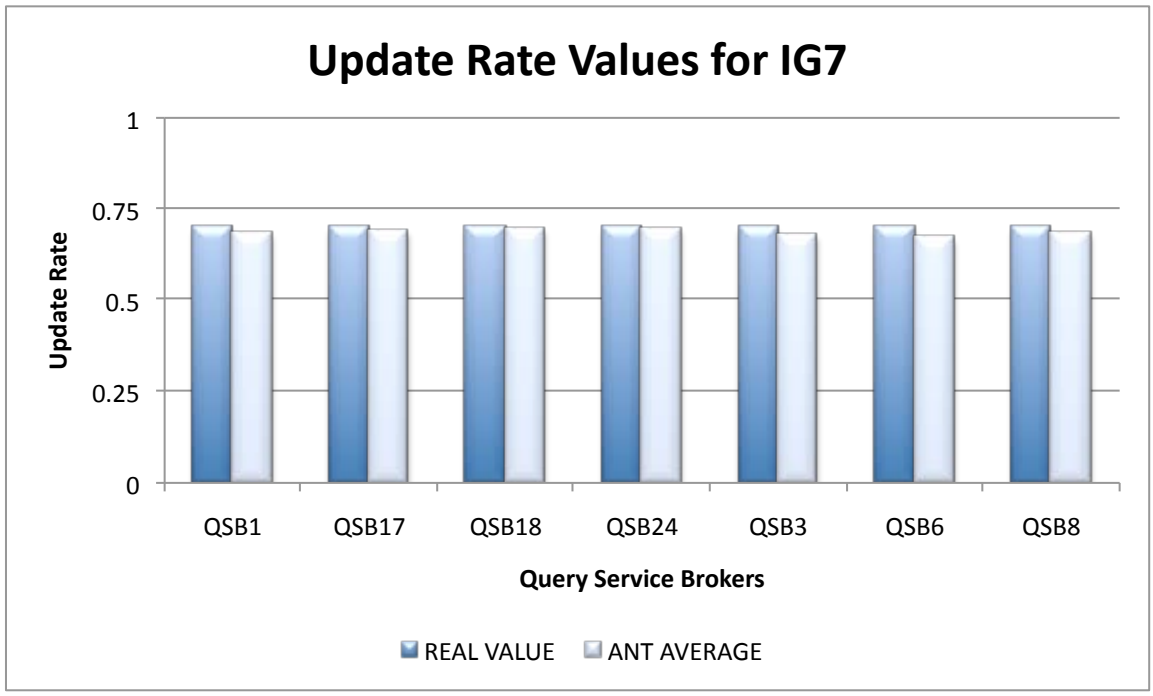


(b) One ant every two hours.

Figure B.16. Currency Metric for IG_7 as seen by the system.

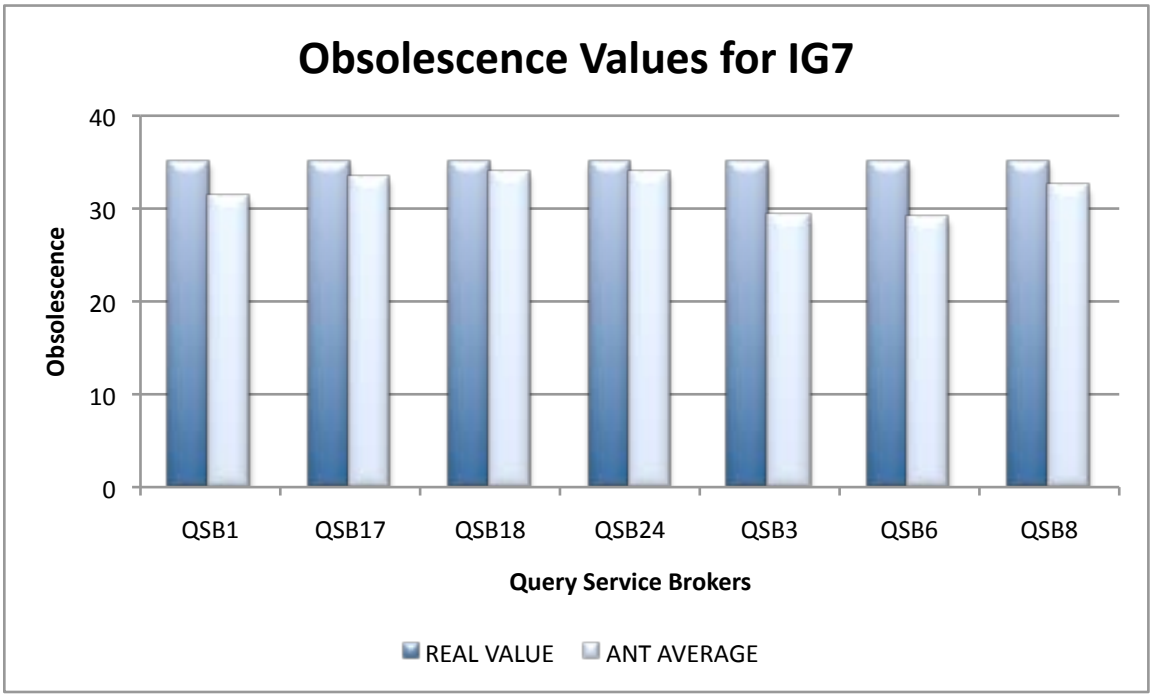


(a) One ant every half-hour.

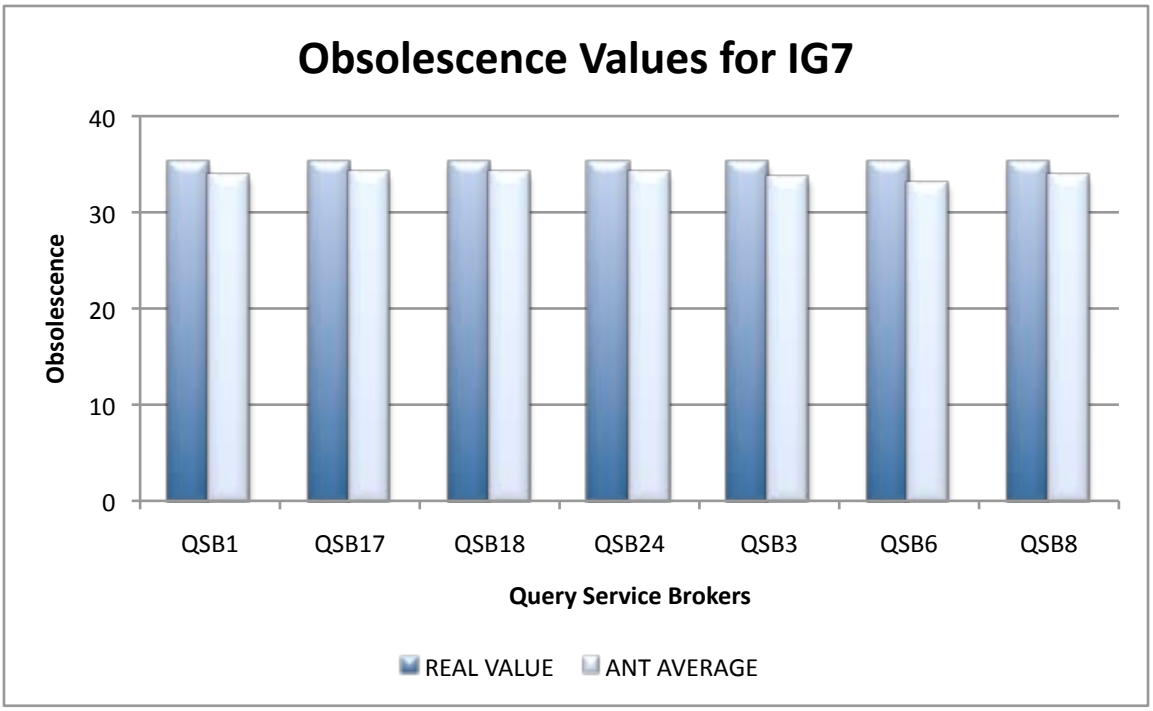


(b) One ant every two hours.

Figure B.17. Rate for IG_7 as seen by the system.



(a) One ant every half-hour.



(b) One ant every two hours.

Figure B.18. Data Freshness for IG_7 as seen by the system.

APPENDIX C

One Ant per Round Strategy : Anova Complete Analysis for Round Frequency

```
Analysis of Variance for Pheromon
Source     DF        SS        MS        F        P
Frecuenc    6  0.060703  0.010117    24.77    0.000
Error      63  0.025736  0.000409
Total      69  0.086439

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ---+---------+---------+---------+---
 0.25      10   0.98617   0.00718                              (----*---)
 0.50      10   0.97660   0.01488                          (----*---)
 1.00      10   0.97660   0.01208                          (----*---)
 3.00      10   0.96809   0.01504                      (----*---)
 6.00      10   0.94681   0.02243                (----*---)
12.00      10   0.91809   0.03755       (---*---)
24.00      10   0.90426   0.01737  (---*----)
                                    ---+---------+---------+---------+---
Pooled StDev =  0.02021             0.900     0.930     0.960     0.990
```

Figure C.1. One-way ANOVA: Pheromone Consistency versus Round Frequency.

```
Analysis of Variance for Similar
Source     DF        SS        MS        F        P
Frecuenc    6  0.093911  0.015652    60.76    0.000
Error      63  0.016229  0.000258
Total      69  0.110140

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ----------+---------+---------+------
 0.25      10    0.9957    0.0103                                  (-*--)
 0.50      10    0.9936    0.0090                                (--*--)
 1.00      10    0.9968    0.0051                                  (--*--)
 3.00      10    0.9894    0.0087                               (--*--)
 6.00      10    0.9766    0.0172                           (--*--)
12.00      10    0.9511    0.0225                     (--*--)
24.00      10    0.8883    0.0267   (--*--)
                                    ----------+---------+---------+------
Pooled StDev =   0.0161                 0.910     0.945     0.980
```

Figure C.2. One-way ANOVA: Similar Cost versus Round Frecuency.

```
Analysis of Variance for Ants
Source     DF        SS        MS        F        P
Frecuenc    6 571378444  95229741  2.3E+05    0.000
Error      63     25595       406
Total      69 571404040

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ----------+---------+---------+------
 0.25      10    8439.5      37.8                                      *
 0.50      10    4265.1      32.5                    *
 1.00      10    2137.1      16.0           (*
 3.00      10     719.8       8.0     *
 6.00      10     364.7       4.4    *)
12.00      10     188.0       3.3    *
24.00      10      96.5       3.0  *
                                    ----------+---------+---------+------
Pooled StDev =     20.2                  2500      5000      7500
```

Figure C.3. One-way ANOVA: Ants versus Round Frecuency.

```
Analysis of Variance for Nodes with Queue
Source     DF        SS        MS        F        P
Frecuenc    6  11228.09   1871.35  1.6E+04    0.000
Error      63      7.40      0.12
Total      69  11235.49

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  -+---------+---------+---------+-----
 0.25      10    34.000     0.000                                      *
 0.50      10    22.500     0.707                    (*
 1.00      10    13.100     0.568                *
 3.00      10     1.000     0.000    *
 6.00      10    -0.000     0.000   *
12.00      10    -0.000     0.000   *
24.00      10    -0.000     0.000   *
                                    -+---------+---------+---------+-----
Pooled StDev =    0.343              0
```

Figure C.4. One-way ANOVA: Nodes with Queue versus Round Frecuency.

APPENDIX D

One Ant per Destination per Round Strategy : Anova Complete Analysis for Round Frequency

```
Analysis of Variance for Pheromon
Source     DF        SS        MS        F        P
Frecuenc    6  0.003298  0.000550     3.86    0.002
Error      63  0.008963  0.000142
Total      69  0.012262

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ----------+---------+---------+------
 0.25      10   0.98830   0.00785                     (------*-------)
 0.50      10   0.98511   0.01249                  (------*-------)
 1.00      10   0.98830   0.01369                     (------*-------)
 3.00      10   0.97660   0.00978           (-------*------)
 6.00      10   0.98404   0.01149                  (------*-------)
12.00      10   0.97766   0.00931            (-------*------)
24.00      10   0.96809   0.01663   (------*-------)
                                    ----------+---------+---------+------
Pooled StDev =  0.01193                  0.970     0.980     0.990
```

Figure D.1. One-way ANOVA: Pheromone Consistency versus Round Frequency.

```
One-way ANOVA: Similar Cost versus Frecuency Round

Analysis of Variance for Similar
Source     DF        SS        MS        F        P
Frecuenc    6 0.0014680 0.0002447     3.55    0.004
Error      63 0.0043459 0.0000690
Total      69 0.0058139

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ---------+---------+---------+-------
 0.25      10   0.99894   0.00336                    (------*-------)
 0.50      10   1.00000   0.00000                     (-------*------)
 1.00      10   0.99787   0.00449                  (-------*------)
 3.00      10   0.99787   0.00449                  (-------*------)
 6.00      10   0.99255   0.00876           (-------*------)
12.00      10   0.99255   0.00876           (-------*------)
24.00      10   0.98617   0.01667  (-------*------)
                                    ---------+---------+---------+-------
Pooled StDev =  0.00831                 0.9870    0.9940    1.0010
```

Figure D.2. One-way ANOVA: Similar Cost versus Round Frecuency.

```
Analysis of Variance for Ants
Source     DF        SS        MS        F        P
Frecuenc    6 1.002E+10 1.670E+09  9.0E+05    0.000
Error      63    117039      1858
Total      69 1.002E+10

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ----------+---------+---------+------
 0.25      10   35361.3      90.5                                      *
 0.50      10   17710.3      49.2                    *
 1.00      10    8878.4      41.6           *
 3.00      10    2982.8      19.0     *
 6.00      10    1508.0      14.6   (*
12.00      10     771.5       7.8   *
24.00      10     390.4       5.3  *
                                    ----------+---------+---------+------
Pooled StDev =     43.1                 10000     20000     30000
```

Figure D.3. One-way ANOVA: Ants versus Round Frecuency.

```
Analysis of Variance for Nodes with Queue
Source     DF        SS        MS        F        P
Frecuenc    6  13865.57   2310.93  8225.34    0.000
Error      63     17.70      0.28
Total      69  13883.27

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  -+---------+---------+---------+-----
 0.25      10    34.000     0.000                                      *
 0.50      10    32.800     0.422                                    (*
 1.00      10    31.000     0.000                                   *
 3.00      10    19.200     1.135                     *)
 6.00      10     7.300     0.483           *)
12.00      10     0.600     0.516   (*
24.00      10    -0.000     0.000   *
                                    -+---------+---------+---------+-----
Pooled StDev =    0.530              0         10        20        30
```

Figure D.4. One-way ANOVA: Nodes with Queue versus Round Frecuency.

APPENDIX E

Multiple Ants per Destination per Round Strategy : Anova Complete Analysis for Round Frequency

```
Analysis of Variance for Pheromon
Source     DF        SS        MS        F        P
Frecuenc    6  0.000960  0.000160     1.25    0.291
Error      63  0.008035  0.000128
Total      69  0.008996

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ---------+---------+---------+-------
 0.25      10   0.98936   0.00709                  (--------*--------)
 0.50      10   0.98511   0.01028            (--------*--------)
 1.00      10   0.98404   0.00561           (--------*--------)
 3.00      10   0.98723   0.00839               (--------*--------)
 6.00      10   0.98404   0.01149           (--------*--------)
12.00      10   0.98298   0.01436          (--------*--------)
24.00      10   0.97660   0.01723  (--------*--------)
                                    ---------+---------+---------+-------
Pooled StDev =  0.01129                 0.9760    0.9840    0.9920
```

Figure E.1. One-way ANOVA: Pheromone Consistency versus Round Frequency.

```
Analysis of Variance for Similar
Source     DF        SS        MS        F        P
Frecuenc    6 0.0006790 0.0001132     3.04    0.011
Error      63 0.0023427 0.0000372
Total      69 0.0030217

                                    Individual 95% CIs For Mean
                                    Based on Pooled StDev
Level       N      Mean     StDev  ---+---------+---------+---------+---
 0.25      10   1.00000   0.00000                (-------*-------)
 0.50      10   1.00000   0.00000                (-------*-------)
 1.00      10   0.99894   0.00336              (-------*-------)
 3.00      10   0.99362   0.00744     (------*-------)
 6.00      10   0.99894   0.00336              (-------*-------)
12.00      10   0.99362   0.00897     (------*-------)
24.00      10   0.99255   0.01009  (-------*-------)
                                    ---+---------+---------+---------+---
Pooled StDev =  0.00610              0.9900    0.9950    1.0000    1.0050
```

Figure E.2. One-way ANOVA: Similar Cost versus Round Frecuency.

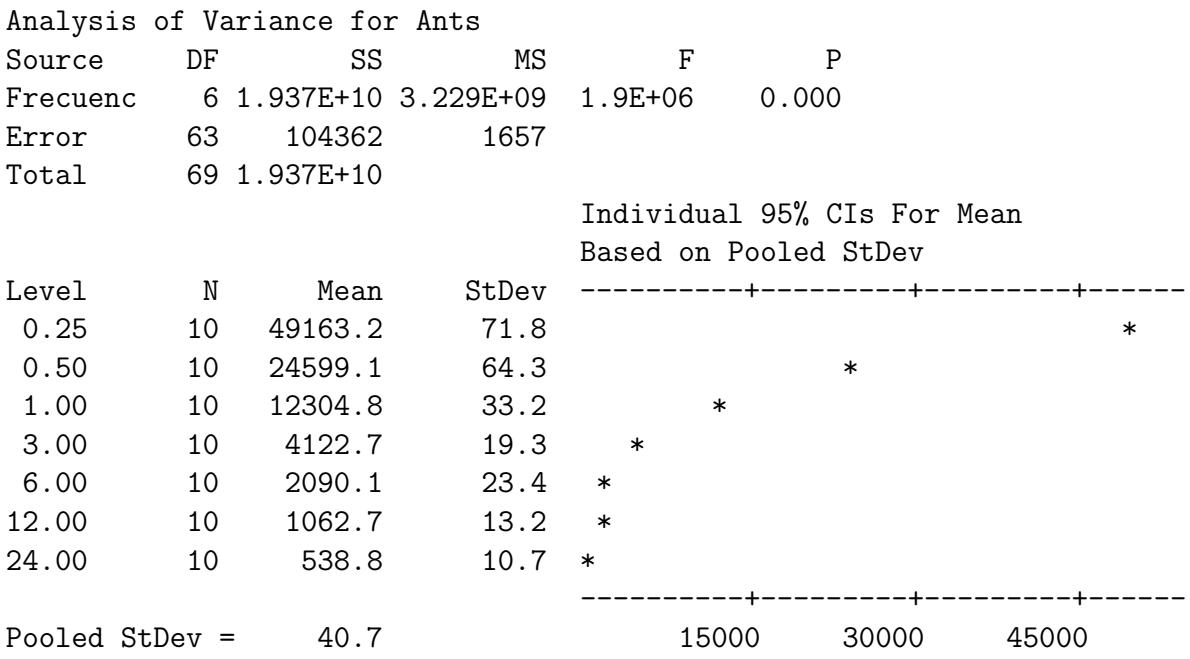


Figure E.3. One-way ANOVA: Ants versus Round Frecuency.

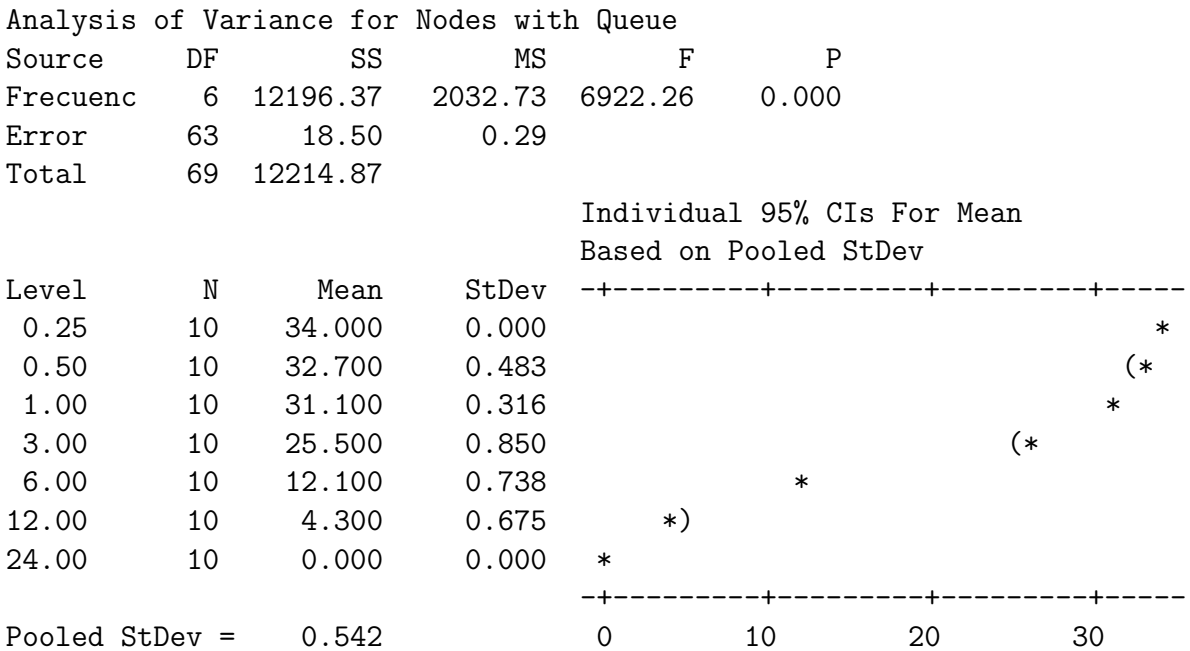


Figure E.4. One-way ANOVA: Nodes with Queue versus Round Frecuency.

APPENDIX F

Comparison Between Ant Launch Algorithms, One Ant per Node: Anova Complete Analysis

```
Analysis of Variance for Pheromon
Source        DF        SS        MS        F        P
Algorith       1  0.000137  0.000137     0.34    0.563
Frecuenc       6  0.140173  0.023362    57.48    0.000
Interaction    6  0.001374  0.000229     0.56    0.759
Error        126  0.051211  0.000406
Total        139  0.192895

                      Individual 95% CI
Algorith        Mean  ----+---------+---------+---------+-------
0             0.9538          (---------------*---------------)
1             0.9518   (---------------*---------------)
                      ----+---------+---------+---------+-------
                    0.9480    0.9510    0.9540    0.9570

                      Individual 95% CI
Frecuenc        Mean  ---+---------+---------+---------+--------
 0.25         0.9872                                 (--*--)
 0.50         0.9803                               (--*--)
 1.00         0.9787                             (--*--)
 3.00         0.9654                         (--*--)
 6.00         0.9415                   (--*--)
12.00         0.9165         (-*--)
24.00         0.9000   (--*--)
                      ---+---------+---------+---------+--------
                   0.9000    0.9300    0.9600    0.9900
```

Figure F.1. One-way ANOVA: Pheromone Consistency versus Round Frequency.

```
Analysis of Variance for Similar
Source        DF        SS        MS        F        P
Algorith       1  0.003208  0.003208    10.41    0.002
Frecuenc       6  0.181462  0.030244    98.14    0.000
Interaction    6  0.001743  0.000290     0.94    0.467
Error        126  0.038830  0.000308
Total        139  0.225243

                      Individual 95% CI
Algorith        Mean  --------+---------+---------+---------+---
0             0.9702                      (-------*--------)
1             0.9606   (-------*--------)
                      --------+---------+---------+---------+---
                         0.9600    0.9650    0.9700    0.9750

                      Individual 95% CI
Frecuenc        Mean  -------+---------+---------+---------+----
 0.25         0.9947                                      (--*-)
 0.50         0.9910                                    (-*--)
 1.00         0.9926                                    (--*-)
 3.00         0.9819                                 (-*--)
 6.00         0.9686                              (--*-)
12.00         0.9410                        (--*-)
24.00         0.8883   (-*--)
                      -------+---------+---------+---------+----
                        0.9000    0.9300    0.9600    0.9900
```

Figure F.2. Two-way ANOVA: Similar Cost versus Algorithm, Round Frecuency.

```
Analysis of Variance for Ants
Source        DF        SS        MS        F        P
Algorith       1  28853801  28853801  1.2E+05    0.000
Frecuenc       6 742368587 123728098  5.0E+05    0.000
Interaction    6  43014311   7169052  2.9E+04    0.000
Error        126     31207       248
Total        139 814267907

                      Individual 95% CI
Algorith        Mean  ----+---------+---------+---------+-------
0             2315.8                                        (*
1             1407.9  *
                      ----+---------+---------+---------+-------
                     1500.0    1750.0    2000.0    2250.0

                      Individual 95% CI
Frecuenc        Mean  ----------+---------+---------+---------+-
 0.25         6802.8                                    *
 0.50         3422.5                      *
 1.00         1716.8             *
 3.00          576.1        *
 6.00          290.6     *
12.00          148.1     *
24.00           75.9   *
                      ----------+---------+---------+---------+-
                           2000.0    4000.0    6000.0    8000.0
```

Figure F.3. Two-way ANOVA: Ants versus Algorithm, Round Frecuency.

```
Analysis of Variance for Nodes wi
Source        DF        SS        MS        F        P
Algorith       1    126.35    126.35   740.47    0.000
Frecuenc       6  21517.67   3586.28  2.1E+04    0.000
Interaction    6    316.90     52.82   309.53    0.000
Error        126     21.50      0.17
Total        139  21982.42

                      Individual 95% CI
Algorith        Mean  ---------+---------+---------+---------+--
0              10.086                                        (-*-)
1               8.186   (-*-)
                      ---------+---------+---------+---------+--
                           8.500     9.000     9.500    10.000

                      Individual 95% CI
Frecuenc        Mean  -+---------+---------+---------+---------+
 0.25         34.000                                    *
 0.50         20.650                         (*
 1.00          8.800              *
 3.00          0.500    *)
 6.00         -0.000    *
12.00         -0.000    *
24.00         -0.000    *
                      -+---------+---------+---------+---------+
                    0.000    10.000    20.000    30.000    40.000
```

Figure F.4. Two-way ANOVA: Nodes with Queue versus Algorithm, Round Frecuency

APPENDIX G

Comparison Between Ant Launch Algorithms, One Ant per Destination: Anova Complete Analysis

```
Analysis of Variance for Pheromon
Source        DF        SS        MS        F        P
Algorith       1  0.008576  0.008576    41.50    0.000
Frecuenc       6  0.008957  0.001493     7.22    0.000
Interaction    6  0.000676  0.000113     0.54    0.773
Error        126  0.026041  0.000207
Total        139  0.044250

                        Individual 95% CI
Algorith        Mean    -------+---------+---------+---------+----
0             0.9812                                (----*-----)
1             0.9655     (----*-----)
                        -------+---------+---------+---------+----
                          0.9660    0.9720    0.9780    0.9840

                        Individual 95% CI
Frecuenc        Mean    ----------+---------+---------+---------+-
 0.25          0.9809                           (------*-----)
 0.50          0.9793                          (-----*------)
 1.00          0.9798                          (------*-----)
 3.00          0.9723                 (-----*------)
 6.00          0.9766                     (------*-----)
12.00          0.9676            (------*-----)
24.00          0.9569     (-----*-----)
                        ----------+---------+---------+---------+-
                             0.9600    0.9700    0.9800    0.9900
```

Figure G.1. One-way ANOVA: Pheromone Consistency versus Round Frequency.

```
Analysis of Variance for Similar
Source        DF        SS        MS        F        P
Algorith       1 0.0003565 0.0003565     7.50    0.007
Frecuenc       6 0.0020307 0.0003384     7.12    0.000
Interaction    6 0.0002037 0.0000340     0.71    0.639
Error        126 0.0059869 0.0000475
Total        139 0.0085777

                      Individual 95% CI
Algorith        Mean  -+---------+---------+---------+---------+
0            0.99514  (----------*---------)
1            0.99833                      (----------*---------)
                      -+---------+---------+---------+---------+
                   0.99360   0.99520   0.99680   0.99840   1.00000

                      Individual 95% CI
Frecuenc        Mean  ----------+---------+---------+---------+-
 0.25         0.9995                          (-----*-----)
 0.50         1.0000                           (-----*-----)
 1.00         0.9989                         (-----*-----)
 3.00         0.9989                         (-----*-----)
 6.00         0.9963                   (------*-----)
12.00         0.9952                 (-----*------)
24.00         0.9883   (------*-----)
                      ----------+---------+---------+---------+-
                           0.9900    0.9950    1.0000    1.0050
```

Figure G.2. Two-way ANOVA: Similar Cost versus Algorithm, Round Frecuency.

```
Analysis of Variance for Ants
Source        DF        SS        MS        F        P
Algorith       1 1.395E+09 1.395E+09  2.9E+05    0.000
Frecuenc       6 9.085E+09 1.514E+09  3.2E+05    0.000
Interaction    6 2.138E+09 356378093  7.4E+04    0.000
Error        126    603538      4790
Total        139 1.262E+10

                      Individual 95% CI
Algorith        Mean  ----------+---------+---------+---------+-
0             9657.5                                          *
1             3344.3   *
                      ----------+---------+---------+---------+-
                           4800.0    6400.0    8000.0    9600.0

                      Individual 95% CI
Frecuenc        Mean  ----------+---------+---------+---------+-
 0.25          23806                                          *
 0.50          11914                      *
 1.00           6007             *
 3.00           2001       *
 6.00           1008      *
12.00            512     *
24.00            258    *
                      ----------+---------+---------+---------+-
                             6000      12000     18000     24000
```

Figure G.3. Two-way ANOVA: Ants versus Algorithm, Round Frecuency.

```
Analysis of Variance for Nodes wi
Source        DF        SS        MS        F        P
Algorith       1   1058.75   1058.75  2127.63    0.000
Frecuenc       6  26884.54   4480.76  9004.39    0.000
Interaction    6   1313.80    218.97   440.03    0.000
Error        126     62.70      0.50
Total        139  29319.79

                      Individual 95% CI
Algorith        Mean  ---------+---------+---------+---------+--
0              17.843                                      (*)
1              12.343  (*)
                      ---------+---------+---------+---------+--
                          13.500    15.000    16.500    18.000

                      Individual 95% CI
Frecuenc        Mean  -+---------+---------+---------+---------+
 0.25          34.00                                     *
 0.50          31.05                                 *
 1.00          26.50                             *)
 3.00          10.15                 *
 6.00           3.65            (*
12.00           0.30     *)
24.00          -0.00     *
                      -+---------+---------+---------+---------+
                     0.00      10.00     20.00     30.00     40.00
```

Figure G.4. Two-way ANOVA: Nodes with Queue versus Algorithm, Round Frecuency

APPENDIX H

Comparison Between Ant Launch Algorithms, Multiple Ants per Destination: Anova Complete Analysis

```
Analysis of Variance for Pheromon
Source        DF        SS        MS        F        P
Algorith       1  0.004427  0.004427    35.30    0.000
Frecuenc       6  0.001830  0.000305     2.43    0.029
Interaction    6  0.000123  0.000020     0.16    0.986
Error        126  0.015799  0.000125
Total        139  0.022179

                      Individual 95% CI
Algorith        Mean  -----+---------+---------+---------+------
0             0.9842                                (-----*------)
1             0.9729   (-----*------)
                      -----+---------+---------+---------+------
                      0.9720    0.9760    0.9800    0.9840

                      Individual 95% CI
Frecuenc        Mean  --------+---------+---------+---------+---
 0.25          0.9840                     (-------*-------)
 0.50          0.9803                 (-------*-------)
 1.00          0.9782             (-------*--------)
 3.00          0.9819                   (--------*-------)
 6.00          0.9771           (--------*-------)
12.00          0.9761         (-------*-------)
24.00          0.9723   (--------*-------)
                      --------+---------+---------+---------+---
                         0.9720    0.9780    0.9840    0.9900
```

Figure H.1. One-way ANOVA: Pheromone Consistency versus Round Frequency.

```
Analysis of Variance for Similar
Source        DF        SS        MS        F        P
Algorith       1 0.0001366 0.0001366     5.30    0.023
Frecuenc       6 0.0008537 0.0001423     5.52    0.000
Interaction    6 0.0001746 0.0000291     1.13    0.349
Error        126 0.0032481 0.0000258
Total        139 0.0044130

                      Individual 95% CI
Algorith        Mean  ----+---------+---------+---------+-------
0            0.99681    (---------*---------)
1            0.99878                   (---------*---------)
                      ----+---------+---------+---------+-------
                    0.99600   0.99720   0.99840   0.99960

                      Individual 95% CI
Frecuenc        Mean  --------+---------+---------+---------+---
 0.25         1.0000                          (------*-------)
 0.50         1.0000                          (------*-------)
 1.00         0.9995                        (-------*------)
 3.00         0.9968              (-------*------)
 6.00         0.9995                        (-------*------)
12.00         0.9957            (------*-------)
24.00         0.9931    (------*-------)
                      --------+---------+---------+---------+---
                         0.9930    0.9960    0.9990    1.0020
```

Figure H.2. Two-way ANOVA: Similar Cost versus Algorithm, Round Frecuency.

```
Analysis of Variance for Ants
Source        DF        SS        MS        F        P
Algorith       1 2.843E+09 2.843E+09  2.6E+06    0.000
Frecuenc       6 1.711E+10 2.851E+09  2.6E+06    0.000
Interaction    6 4.361E+09 726836376  6.6E+05    0.000
Error        126    137885      1094
Total        139 2.431E+10

                      Individual 95% CI
Algorith        Mean  ---+---------+---------+---------+--------
0            13411.6                                        *
1             4399.0    *
                      ---+---------+---------+---------+--------
                    5000.0    7500.0   10000.0   12500.0

                      Individual 95% CI
Frecuenc        Mean  ----------+---------+---------+---------+-
 0.25        32664.0                                           *
 0.50        16335.1                        *
 1.00         8171.5              *
 3.00         2733.6       *
 6.00         1381.3      *
12.00          698.6     *
24.00          353.0    *
                      ----------+---------+---------+---------+-
                           8000.0   16000.0   24000.0   32000.0
```

Figure H.3. Two-way ANOVA: Ants versus Algorithm, Round Frecuency.

```
Analysis of Variance for Nodes wi
Source        DF        SS        MS        F        P
Algorith       1   1465.78   1465.78  6176.86    0.000
Frecuenc       6  24542.69   4090.45  1.7E+04    0.000
Interaction    6   1377.77    229.63   967.67    0.000
Error        126     29.90      0.24
Total        139  27416.14

                      Individual 95% CI
Algorith        Mean  -------+---------+---------+---------+----
0              19.957                                          (*
1              13.486   *)
                      -------+---------+---------+---------+----
                        14.400    16.000    17.600    19.200

                      Individual 95% CI
Frecuenc        Mean  -+---------+---------+---------+---------+
 0.25          34.00                                    *
 0.50          31.20                                *
 1.00          27.35                             *)
 3.00          16.30                        *)
 6.00           6.05           *
12.00           2.15       *
24.00           0.00    *
                      -+---------+---------+---------+---------+
                     0.00      10.00     20.00     30.00     40.00
```

Figure H.4. Two-way ANOVA: Nodes with Queue versus Algorithm, Round Frecuency