Δ.: 658 .5

222

ΜΠΛΕ

Μεταπτυχιακή Εργασία Ειδίκευσης

Ουρές Διαδικασιών Εξαγωγής Μετασχηματισμού Φόρτωσης για Ενεργές Αποθήκες Δεδομένων (Extract-Transform-Load Queues for Active Data Warehousing)

Αλέξανδρος Καρακασίδης

Επιβλέπων: Παναγιώτης Βασιλειάδης

ΙΩΑΝΝΙΝΑ, ΙΑΝΟΥΑΡΙΟΣ 2005

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντά μου Καθηγητή κ. Παναγιώτη Βασιλειάδη για τη βοήθεια και την υποστήριξή του, χωρίς την οποία η εργασία αυτή θα ήταν αδύνατο να υλοποιηθεί. Επίσης, θα ήθελα να τον ευχαριστήσω για την υπομονή που επέδειξε μέχρι την ολοκλήρωση της εργασίας αυτής. Τέλος θα ήθελα να ευχαριστήσω τους κ.κ. Καθηγητές Ευαγγελία Πιτουρά και Ευάγγελο Παπαπέτρου για τη βοήθειά τους.

ΕΙΣΑΓΩΓΗ

Παραδοσιακά, η ενημέρωση των Αποθηκών Δεδομένων γίνεται ασύγχρονα. Ο όρος «Ενεργή Αποθήκευση Δεδομένων» αναφέρεται σε μία νέα τάση, κατά την οποία οι Αποθήκες Δεδομένων ενημερώνονται όσο το δυνατόν συχνότερα, λόγω των υψηλών απαιτήσεων των χρηστών για πρόσφατα δεδομένα. Σε αυτή την εργασία, προτείνουμε ένα πλαίσιο για την υλοποίηση μίας Ενεργής Αποθήκης Δεδομένων έχοντας ως στόχους τα εξής: (α) Ελάχιστες τροποποιήσεις στο ήδη υπάρχον λογισμικό, (β) ελάχιστη επιβάρυνση στο πρόγραμμα που παράγει τα δεδομένα, λόγω του ενεργού χαρακτήρα της μεταφοράς των δεδομένων και (γ) τη δυνατότητα ομαλού καθορισμού των ρυθμίσεων του περιβάλλοντος που θα δημιουργήσουμε. Στο σύστημά μας, έχουμε υλοποιήσει διαδικασίες Εξαγωγής, Μετασχηματισμού και Φόρτωσης χρησιμοποιώντας δίκτυα ουρών. Χρησιμοποιήσαμε επίσης στοιχεία απο τη Θεωρία Αναμονής τόσο για να προβλέψουμε την επίδοση του συστήματος, όσο και για να ρυθμίσουμε τη λειτουργία του. Λόγω των επιβαρύνσεων στην επίδοση του συστήματος, οι οποίες προέκυψαν κατά τη δημιουργία του, διερευνούμε διάφορες αρχιτεκτονικές προσεγγίσεις και σχολιάζουμε τα ζητήματα που προκύπτουν από κάθε μία από αυτές.

ABSTRACT

Traditionally, the refreshment of data warehouses has been performed in an off-line fashion. Active Data Warehousing refers to a new trend where data warehouses are updated as frequently as possible, due to the high demands of users for fresh data. In this thesis, we propose a framework for the implementation of active data warehousing, keeping in mind the following goals: (a) minimal changes in the software configuration of the source, (b) minimal overhead for the source due to the "active" nature of data propagation, (c) the possibility of smoothly regulating the overall configuration of the environment in a principled way. In our framework, we have implemented ETL activities over queue networks and employ queue theory for the prediction of the performance and the tuning of the operation of the overall refreshment process. Due to the performance overheads incurred, we explore different architectural choices for this task and discuss the issues that arise for each of them.

Contents

1. **Introduction** 11

2. **Related Work** 21
   - 2.1. Materialized Views 21
   - 2.2. Web Services 23
   - 2.3. Streams 24
   - 2.4. ETL 25
   - 2.5. Comparison of Related Work to our Contribution 27

3. **Framework and Issues Raised** 29
   - 3.1. System Architecture 29
   - 3.2. Issues Raised 31
     - 3.2.1. Choices Concerning the Topology 32
     - 3.2.2. Choices Concerning the Source 35
     - 3.2.3. Choices Concerning the Staging Area 35
     - 3.2.4. Choices Concerning the Warehouse 37

4. **Theoretical Analysis of ETL Workflows for Active Warehousing using Queue Theory** 39
   - 4.1. Definition of a Queuing Model 40
   - 4.2. Statistical Distribution of Interarrival and Processing Times 41
   - 4.3. Markov Process and Markov Chains 42
   - 4.4. Kendall's Notation and Little 's Law 43
   - 4.5. Measures of Effectiveness 44
   - 4.6. The M/M/1 Queuing System 45
   - 4.7. Networks of Queues and the Jackson Theorem 46
   - 4.8. Multi Class Jackson Networks 49
   - 4.9. Applying Queue Theory to ETL Workflows 50
     - 4.9.1. Filters 51
     - 4.9.2. Transformers 52
     - 4.9.3. Binary Operators 53
     - 4.9.4. Generic Model 55
   - 4.10. Methodology for Solving ETL Scenarios 56

5. **Experiments on Architecture without ETL Processing** 59
   - 5.1. Smooth Upgrade 61
   - 5.2. UDP vs. TCP 63
   - 5.3. Overhead at Source 64
   - 5.4. Number of Queues 67
   - 5.5. Data Freshness 68

5.6. Topology and Source Overhead 75

6. Operational Evaluation **79**
6.1. Overhead at Source 80
6.2. Throughput Capability of ETL Operations 81
6.3. Data Freshness 83
6.4. Theoretical vs. Experimental Evaluation 90

7. Conclusions and Future Work **97**

References **99**

1 INTRODUCTION

Database Management Systems are used by organizations to support everyday operations. Such applications incur small changes to data. This type of applications is called Online Transaction Processing (OLTP) Applications and focus on processing efficiently and reliably large number of transactions.

Large organizations however, apart from using DBMS 's for covering runtime operations, also use them as tools for strategic decisions. Current and historic data are analyzed resulting in trends available for taking decisions. Such applications are called decision support systems (DSS 's).

To extract information based on historic data, trends and cumulative results, the DSS 's should be able to use efficiently grouping operators and aggregation functions over data that are highly voluminous. Applications carrying out such tasks are characterized as On Line Analytical Processing (OLAP) applications.

To be able to deal with an environment having such demanding conditions, specialized DBM S's are used called Data Warehouses. The aim of Data Warehouses is twofold:

a) To integrate heterogeneous data sources, which is achieved by gathering all information in a single location, and
b) To avoid conflicts between OLTP and OLAP applications, resulting in high system performance and availability.

Data Warehouses are usually concluded by Data Marts, which are specialized subject subsets, to further enhance OLAP applications. The relation between OLTP, Data Warehouses and OLAP is illustrated in Figure 1.1.

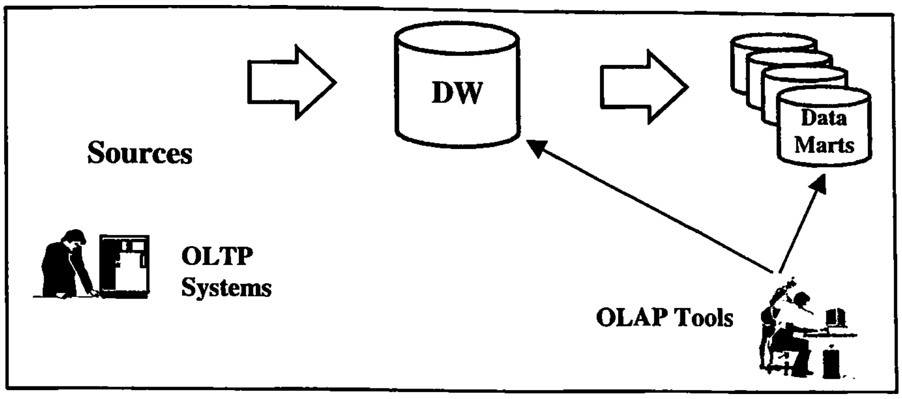


Fig. 1.1. Coarse Architecture Overview

According to [RaGe02], a Data Warehouse is defined as a database that collects and stores data from several databases. In [Inmo02], a Data Warehouse is defined as a subject oriented, integrated, non-volatile, and time variant collection of data in support of management decisions.

The benefits of a data warehouse are coarsely sketched by the following properties: Semantic reconciliation, performance, data quality, and availability. The term "semantic reconciliation" refers to the data warehouse property of modeling the same entities, modelled in different ways at the sources, under a unique database schema. Additionally, the history of the loaded data is kept. Performance is an important issue, since the answers to the posed queries should be available in acceptable time, without affecting the operation of the OLTP application. Moreover, performance is boosted by avoiding normalized schemas for storing data. Data quality is an important issue since data arriving at the warehouse is in most cases inconsistent. Finally, availability is another important factor. The architecture of a Data Warehouse is illustrated in Figure 1.2.

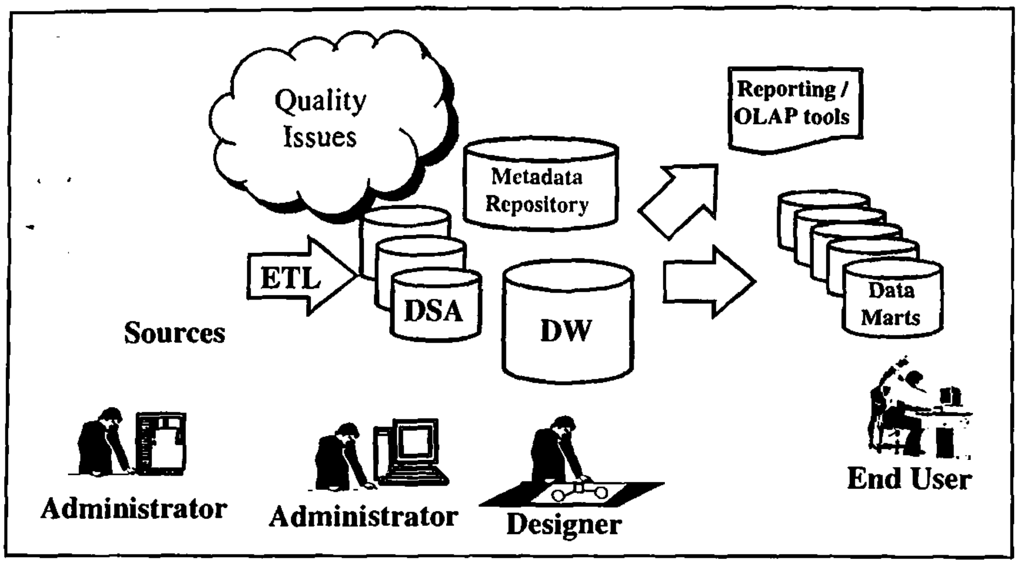


Fig. 1.2. Architecture Overview

The Sources are the actual OLTP applications from which the Data Warehouse retrieves operational data. The Data Staging Area (DSA) is an intermediate database where data are cleaned and transformed before their loading to the Data Warehouse. The Data Warehouse (DW) and the Data Marts (DM) store data provided to the users. The Metadata Repository is the subsystem which stores information concerning the structure and the operation of the system. ETL (Extract - Transformation - Loading) applications extract the data from the sources, transform and clean them before loading them to the Warehouse. Reporting and OLAP tools are reporting applications that perform OLAP, DSS and Data Mining tasks.

ETL, which is an acronym for Extraction-Transformation-Loading, is a category of tools for managing data warehouse operational processes. Their basic tasks, as summarized in [VaSS02] are:

- the identification of relevant information at the source side
- the extraction of this information

- the customization and integration of the information coming from multiple sources into a common format
- the cleaning of the resulting data set, on the basis of database and business rules
- the propagation of the data to the data warehouse and/or data marts.

In this thesis, we are interested in ETL applications implementing the refreshment of DW contents. The transformations useful for our case are filters, transformers and binary operators. These represent common data cleaning tasks used in Data Warehousing environments.

Traditionally, the refreshment of data warehouses has been performed in an off-line fashion. As already mentioned, in a traditional data warehouse setting, data are extracted from the sources, transformed, cleaned and eventually loaded to the warehouse through ETL applications. This set of activities takes place during a 'loading window', usually during the night, in order to avoid overloading the source production systems with the extra workload of this workflow.

Active Data Warehousing refers to a new trend where data warehouses are updated as frequently as possible, due to the high demands of users for fresh data. The term is also encountered as 'real time warehousing' for that reason [Whit02]. To give a concrete example, we mention [AdFi03], where a case study for mobile network traffic data is discussed, involving around 30 data flows, 10 sources, and around 2TB of data, with 3 billion rows. The throughput of the (traditional) population system is 80M rows/hour, 100M rows/day, with a loading window of only 4 hours. The authors report that user requests indicated a need for data with freshness at most 2 hours.

This kind of request is technically challenging for various reasons. First, the source systems cannot be overloaded with the extra task of propagating data towards the warehouse. Second, it is not obvious how the active propagation of data can be implemented, especially in the presence of legacy production systems. The problem becomes worse since it is rather improbable that the software configuration of the source systems can be significantly modified to cope with the new task (both due to the down-time for deployment and testing and the cost to administrate, maintain and monitor the execution of the new environment).

So far, research has dealt with the problem of maintaining the warehouse in its traditional setup [GuMu95, ZGHW05, ZhRu02]. In this case, materialized views are refreshed in the presence of updates, but the general idea is that the refreshment is performed off-line. In a different line of research, data streams [Aba+03, BaWi01, LoGe03] could possibly appear as a potential solution. Nevertheless, at least until now, research in data streaming does not appear to fit naturally within a data warehousing context – on the contrary, it appears to be a competitive paradigm to warehousing. Research in data streams has focused on topics concerning the front-end, such as on-the-fly computation of queries, without a systematic treatment of the issues raised at the back-end of a data warehouse. For example, to our knowledge, there is no work related to how streaming data are produced or extracted from data producers; not to mention the extra problems incurred when the data producers are operational systems.

To this end, in this thesis we attempt to approach the problem from a clean sheet of paper. We investigate the case where the source of the warehouse is a legacy system. The specific problem involves the identification of a software architecture along with appropriate design guidelines for the implementation of active warehousing. We are motivated by the following *requirements* in achieving this goal.

1. *Maximum freshness of data.* We want to implement an active data warehousing environment to obtain as fresh data as possible in the warehouse
2. *Smooth upgrade of the software at the source.* We wish to implement a framework where the modification of the software configuration at the source side is minimal

3. *Minimal overhead of the source system.* It is imperative to impose the minimum additional workload to the source
4. *Stable interface at the warehouse side.* It would be convenient if the warehouse would export a stable interface for its refreshment to all its source sites.

The grand view of our environmental setup is depicted in Figure 1.3. A set of sources comprise source data and possibly source applications that manage them (for the case of legacy sources) or DBMS's for the case of conventional environments. The updates that take place at the sources have to be propagated towards the warehouse. Due to reasons of semantic or structural incompatibilities, an intermediate processing stage has to take place, in order to transform and clean the data. Once ready for loading, the data from the intermediate layer are loaded at the warehouse, through a set of on-line loaders.

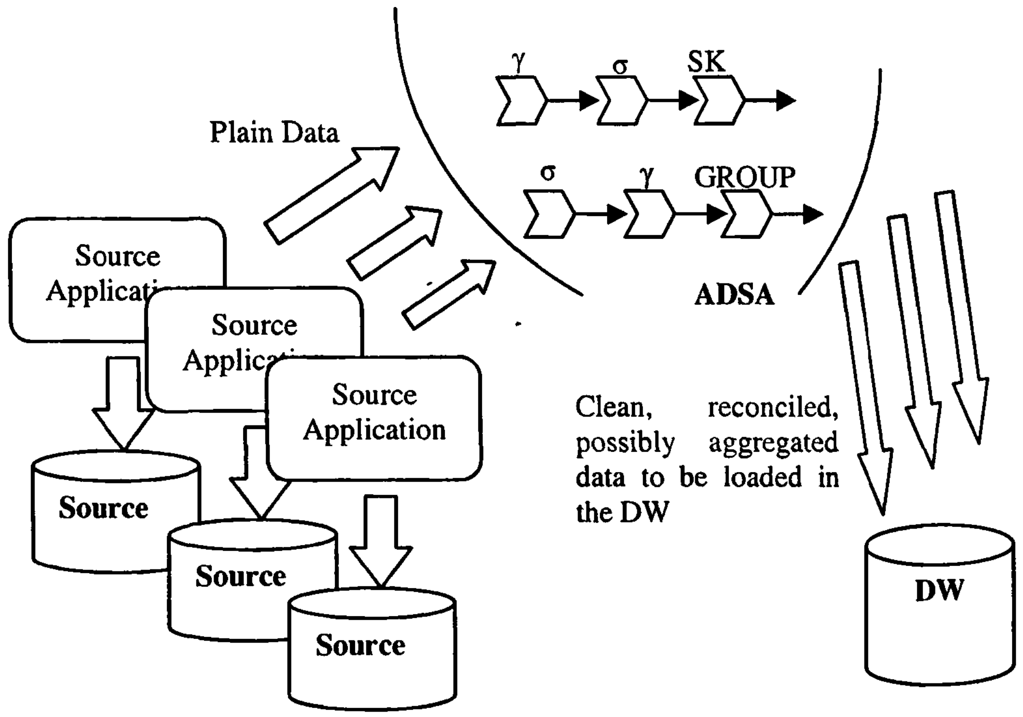


Fig. 1.3. Active Warehouse Architecture Overview (γ represents aggregations and σ selections).

Mapping this grand view to concrete technical choices requires the tuning of several components of the architecture. Following, we briefly summarize our findings that affected our architectural choices.

Starting with the sources, in this thesis, we have focused on legacy systems. Apart from the requirement of minimal changes at the source side, legacy sources pose the interesting problem of having an application (instead of a DBMS) managing the data. We modify a library of routines for the management of data to allow the interception of the calls without affecting the applications. The modification involves (a) inserting no more than 100 lines of code to a library of routines for source management and (b) recompiling the application (which was not affected), over this library. Also, as far as the communication between stages is concerned, we transmit blocks of records for reasons of performance and minimal overhead of the source system.

The internal architecture of the intermediate layer is not obvious, either. For each ETL activity, we employ a queue to store incoming records before they are processed. Each activity processes the incoming data on-line and then passes its output to the next queue for further processing. Again, for reasons of performance, blocks of records are the unit of exchange and not individual records.

We do not assume a fixed set of ETL operators, but rather we provide a taxonomy of such operations, based on their operational semantics. New operators can be added to the taxonomy as they are defined. Specifically, the taxonomy of activities consists of the following categories: (a) Filters, (b) Transformers and (c) Binary Operations.

Filters examine each incoming tuple to determine whether it meets certain criteria. If these criteria are fulfilled, then a tuple is accepted and propagated towards an acceptance output. If not, it is rejected and possibly propagated towards a rejection output.

Considering the case of Transformers, tuples entering a transformer undergo changes to their value and/or their schema. We can distinguish two subclasses of Transformers taking into account the relationship between the number of tuples entering and the number of tuples exiting the transformation. In the first case the two quantities are equal. In the second case, the number of tuples entering the system is different compared to the number of tuples exiting. This occurs because some of the tuples entering service are aggregated or merged.

The third class of ETL activities deals with Binary operators. This is the case where data from multiple sources are combined and a single outgoing stream is produced. Examples of such operations involve variants of the join operation, including the join of data from different tables, as well as difference and update detection operations among different snapshots of the same table.

To predict the performance of the system, we employ queue theory for networks of queues. Our experimental results indicate that the assumption of a M/M/1 queue for each of the ETL activities provides an accurate estimation.

At the same time, to implement the requirement for stable interface at the side of the warehouse, the data are further propagated towards the warehouse through an interface involving Web Services [ACKM03]. The need for web services as the technical solution for populating the warehouse with fresh data is not self-evident and requires justification. In fact, web services are known to be rather heavy middleware in terms of resource consumption [Duqu03], which potentially jeopardizes the requirement of fresh data and minimal overhead. The main advantages of web services compared to other middleware solutions (RPC, ORB's, message queues, etc) are two: (a) interoperability, meaning that they can be deployed in all platforms and configurations and (b) possibility of exporting them outside the intranet of an organization. We emphasize the interoperability property: in a large organization, there is a wide variety of data sources, involving several platforms and configurations. Web services can provide a common, stable interface for the warehouse to all these sources without requiring major design and integration effort. Also, this loose coupling of sources and the warehouse results in minimal impact in the case of changes, either at the source or at the warehouse. Obviously, performance has been a concern too. Still, as we discuss in Section 4, our experiments indicate that the overall delay, incurred by the adaptation of a solution based on web services is rather small, especially if one is willing to trade resource (mainly main memory) for freshness.

In a nutshell, our contributions can be listed as follows:

- We set up the architectural framework and the issues that arise for the case of active data warehousing.
- We set up the theoretical framework for the problem, by employing queue theory for the prediction of the performance of the system.
- We provide technical solutions for the implementation of our reference architecture, achieving (a) minimal source overhead, (b) smooth evolution of the software configuration at the source side and (c) fine-tuning guidelines for the technical issues that appear.
- We substantiate our results through extensive experimentation.

The rest of this thesis is organized as follows. In Chapter 2, we present work related to our approach. In Chapter 3, we detail the different architectural choices and the technical challenges each of them incurs. Chapter 4 contains elements of queue theory and the model used to describe our architecture. Chapter 5 and 6 contain our experimental evaluation for defining our architectural setup and measuring our system performance respectively. Finally, in Chapter 7 we sum up the lessons learned and present some thoughts for future work.

2 RELATED WORK

In this Chapter, we present work related to our approach. We structure related work as follows: first, we discuss the area of materialized views. Next, we make a reference to Web Services. The third section of our related work presents the area of streams. Finally, we conclude with work in the area of ETL.

Work in materialized views refreshment is quite related to our setting. This is because a materialized instance of the relations stored at the Source side resides at the warehouse. For the non expert reader, we make a quick reference to Web Services. The Web Services API is an important part of our architecture, since they are used to transfer data to the warehouse. Work concerning streams is related to our system, too. Both streams and our system focus on managing continuous flows of data. However, while in the case of streams, data losses are acceptable, the same does not happen in our case. Finally, we present work concerning ETL transformations. Our architecture uses ETL transformations in the Staging Area, thus, work in this field is related to our approach.

2.1 Materialized Views

In [ZhRu02], the authors propose the Schema change and Data update Concurrency Control system for checking the concurrency of schema changes and data updates performed by distributed Information Sources.

[GuMu95] describes materialized views and proposed techniques for their maintenance. A taxonomy is also presented over four different dimensions. A materialized view is like a cache: a copy of the data that can be accessed quickly. The difference between the materialized and non materialized views is that the tuples are stored in the database.

Incremental maintenance of a materialized view means that only changes in the database are used to compute changes in the materialized view.

There is no algorithm to solve the view maintenance problem for deletions using only the materialized view. The counting algorithm which is described, works by storing the number of alternative derivations of each tuple in the materialized view, in order to handle deletions. For recursive views the DRed algorithm is mainly outlined. This algorithm deletes from the view an overestimation of affected tuples, calculates then the alternative derivations and inserts the new and those that exist again into the view. Three variations of this algorithm are also presented. Also, altered variations of counting algorithms to handle recursive views are presented. Next, algorithms using partial information for view maintenance are presented. These focus on checking whether the view can be maintained using the available information, and then how to maintain the view. Some algorithms only test whether a view remains unaffected by an update. If this test fails then another view update algorithm is used. Self-maintainable views are those that can be maintained using only the materialized view and key constraints.

In [ChCR02] the usage of a transaction model for data warehouse maintenance is proposed. This model assumes autonomous data sources, which means sources that can alter their data autonomously without accepting external locks. It is also assumed that concurrency control, which is achieved by means of versions, is sequential, i.e. only one transaction can be processed. Two types of transactions are distinguished: the first is source update transactions, which trigger the second type, DW maintenance transactions. A wrapper is responsible for managing the versioning system. Initially, it creates versions whenever data are updated.

In [ZGHW95] the authors propose the eager compensation algorithm (ECA) and some variations of it for dealing with view maintenance anomalies in a warehousing environment. The model assumed is the following: a legacy source is assumed, incapable of handling views. Every update that occurs is sent to the DW which maintains a materialized view. When the DW receives the update (insertion or deletion) issues a query towards the source. The source calculates the result which is then stored in the materialized view.

With this setup, anomalies occur when more than one updates occur, one after the other and the warehouse sends queries to the source without waiting the answer from the source to the previous query. To deal with issue the authors propose the Eager Compensating Algorithm. When the DW detects an inconsistent state, i.e. a new update received, without an answer to a previous query has been received, a compensating query is issued to the Source resulting to a consistent result in the Warehouse. Moreover two improvements of the algorithm are presented: the first, the ECA - key algorithm, which reduces the communication load between the Source and the DW when the view includes a key. The second, the ECA – local algorithm determines which update can be handled locally at the warehouse.

2.2 Web Services

Web services appear to be the latest development in the field of middleware, crafted towards enabling the integration of software at an Internet scale. Web services evangelize universal interoperability by exploiting Internet technologies, XML messaging, widely accepted standards, and loose coupling of applications. An excellent reference book for the field of Web services is [ACKM03]. Web services assume a software stack ranging from the low HTTP transfer protocol, to the execution part (SOAP), the service description part (WSDL) that exports the public interface of services [WSDL03] and service composition [WSFL01]. In the context of this thesis, the main protocol of interest is the Simple Object Access Protocol (SOAP) [SOAP01]. SOAP specifies a message format for the communication of Web services along with the bindings to HTTP and SMTP protocols for the delivery of messages. The messages are XML documents, or *envelopes* comprising a *header*, with meta-information for the processing of the message and a *body* with the actual contents of the message.

2.3 Streams

Aurora [ACCC+03] is a data flow system designed to support monitoring applications. Its basic job is to process incoming streams. Aurora has an extended query model supporting real time processing, views, materialized views and ad hoc queries. All these operations can be combined with each other and form a network. This network also includes caches, called connection points, which allow applications joining the system to have access to data of the recent past. A connection point can also be materialized through a DBMS.

Moreover an optimization method for the operations' network at run-time is presented. According to Aurora's approach, the user provides the system with 2d graphs designating critical areas for Quality of Service. If some of these variables is not fulfilled during operation, Aurora sheds some of its load. The data operations communicate with each other with LIFO queues. In order to store them, Aurora uses predefined blocks of space, which either doubles or reduces by half. Finally, a scheduler brings in memory the queues with higher priority.

In [BBDM+02] fundamental models and issues are considered for the development of a general purpose Data Stream Management System Model. Differences are outlined between the data stream model and the conventional relational model. For instance, concerning the type of queries each family of systems can answer. The authors also outline the bad behavior of triggers if used in such systems and the need of high performance techniques in order to answer queries in such data intensive systems.

The authors also examine in detail issues concerning queries over streams, such as unbounded memory requirements, approximate query answering, sliding windows and other. There also is an extensive reference to approaches of many streaming systems and the system they have developed called STREAM. Finally algorithmic issues concerning streams are discussed such as histograms, sampling, etc.

The most relative work to our approach is the one presented in [JiCh03]. In this paper, the authors model single SELECT operations as M/D/1 or M/M/1 queuing systems, depending on the type of condition used and PROJECT operations as M/D/1 queuing systems. Moreover they develop a formula using queue theory, in order to model the hash join of two incoming streams as an M/(D1, D2)/1 queuing system.

2.4 ETL

Potter's wheel [RaHe01] is an interactive data cleaning system. Users gradually build transformations by composing and debugging transforms on a spreadsheet like interface. Discrepancy detection is done in the background on the latest transformed view of the data. The desired results can be specified as example values. The main components of Potter's wheel are the following: a Data Source (ODBC data source or text file), which provides the data to be cleaned. A Transformation Engine where transformations can be ordered via examples or patterns. The Online Reorderer, where the desired transformations are declared. The Automatic Discrepancy Detector runs in the background data the cleaning algorithms. Moreover, an accumulated state is maintained, in order to detect multi-row anomalies where a set of values is individually correct, but together violate some integrity constraint.

The Evaluation of pattern suitability is made upon three characteristics: Recall meaning that the structure should match as many column values as possible, Precision where the structure should match as less other values as possible and finally the structure should have minimal length in order to be as generic as possible. The minimum description length principle offers a way to make a trade-off between overfitting and underfitting, minimizing the total length required to encode data using a structure. Better structures result in smaller Description Lengths. The Description Lengths are computed using specific formulas and taking into account the three aforementioned characteristics.

Ajax [GFSS00] is a data cleaning tool. It is based on user interaction and on automatic procedures defined in AJAX scripting language, which as an SQL extension. It aims at cleaning data with quality problems of the following classes: object identity problems, errors and inconsistencies. Ajax distinguishes four types of transformations: mapping, which standardizes data format, matching, which finds data referring possibly to the same object, clustering which groups similar objects, and finally merging which eliminates duplicates. These operations can be performed combining the following alternatives: using stored system procedures, using additional procedures defined by a human expert, or interacting directly with the human expert.

In [VaSS02] A general framework is presented for modeling the internal structure of ETL activities. ETL activities and their consistent parts are initially modeled and reduced to a graph called the "Architecture Graph", in order to treat the ETL scenario as a skeleton of the overall environment. In this graph, data and functions are represented as nodes, while the edges of the graph depict relationships between data. The authors also provide two zooming algorithms for transforming the graphs. The first, the "In and Out Zooming" algorithm aims at the elimination of the information overflow produced by the modeling of the ETL activity. The second, the "Major Flow" algorithm focuses on following the data flow from sources to targets.

In [VSGT02] the authors describe a framework for specifying ETL scenarios aiming mainly to achieve genericity and customization. The main focus is on the data-centric part of the ETL activities. A generic metamodel is presented, which covers all the types of entities that comprise such an activity. These are generic classes in which the ETL entities belong. Moreover, a specialized form of the metamodel is introduced which contains models of frequently used ETL tasks. In specific, this specialization, called template layer, consists of subclasses of the more generic metamodel layer. The materialization of these subclasses are the actual ETL entities.

Genericity and customization is achieved by introducing new templates, which will be specializations of the metamodel layer, and will also be abstractions of the entities used in the ETL scenario. The authors also present ARKTOS II a graphical tool, for the design of ETL scenarios based on their approach.

2.5 Comparison of Related Work to our Contribution

As an overall evaluation of the related work and a comparison to our contributions, we can mention the following. First, work on materialized views has specifically focused on the issue of relational views. Transformations that lie outside the realm of relational algebra have not been taken into consideration by the related work. At the same time, ETL workflows frequently comprise of transformations that employ external functions to compute tuples and values in ways more or less far from the expressive power of relational algebra. As far as the existing work on ETL is concerned, this has mainly to do (a) with classes of transformations executed either interactively, or off-line, and (b) with the design aspects of ETL workflows. Our work covers a topic that has not been tackled so far by related work in ETL. Research efforts on the management streams constitute the most relevant area to our work. Still, to our knowledge, stream workflows have not been studied in a principled manner, whereas in our work we employ queue theory for that purpose. Moreover, typically, the problem studied in the area of streams concerns continuous relational queries rather than the propagation of data from one data store to another. Again, queries (standing for transformations in our case) outside relational algebra have not been studied yet.

3 FRAMEWORK AND ISSUES RAISED

There are several issues concerning the implementation of a framework for active data warehouse. Therefore, in this Chapter we will start by presenting the general architecture of such a system. In section 3.1, we present the grand view for active warehousing and its specific instantiation that we have investigated. Then, in section 3.2, we proceed to a detailed presentation of the issues raised within this framework.

3.1 System Architecture

In our architecture we assume we have a single source of data. We consider this limitation in order to evaluate architectural alternatives which will offer the best behavior to our framework. Hence, our architecture consists of the following elements: a Data Source generating data, an intermediate data staging area that will be referred to as the Active Data Staging Area (ADSA) where the processing of data takes place and the Data Warehouse. The architecture is illustrated in Figure 3.1.

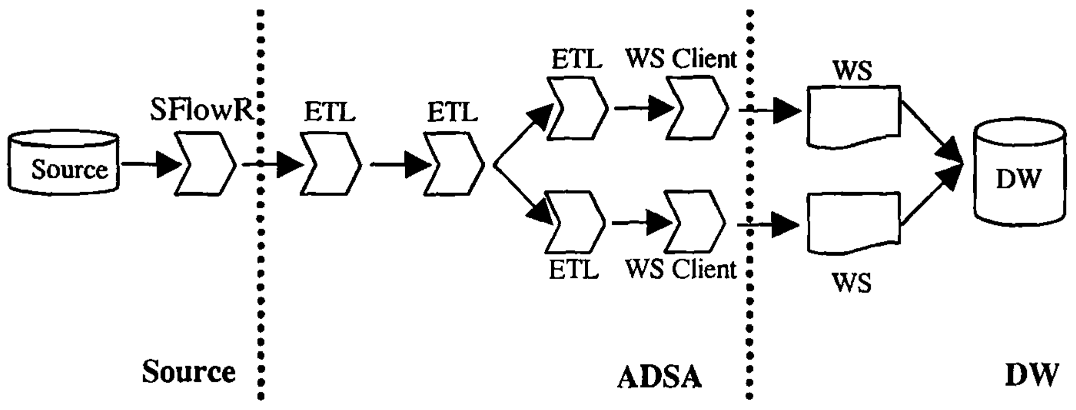


Fig. 3.1. Architecture Overview

The Source comprises of a data store (legacy or conventional) and an operational data management system (e.g., a DBMS or an application, respectively). Changes that take place at the source side have to be propagated towards the warehouse, which typically resides in a different host computer. The communication between hosts employs a network protocol (e.g., TCP or UDP). To avoid the extra overhead of overloading the network with half-full packets and, as our experiments indicate, to avoid overloading the source with the extra task of performing this task, we employ a Source Flow Regulator (SFlowR) module that compiles changes in blocks and propagates them towards the warehouse.

Once record blocks leave the source, an ETL workflow receives them at the intermediate staging area. The role of the ETL workflow is to cleanse and transform the data in the format of the data warehouse. The ETL workflow comprises a set of ETL activities, also called ETL queues, each pipelining blocks of tuples to its subsequent activities, once its filtering or transformation processing is completed. To perform this task, each ETL activity checks its queue (e.g., in a periodic fashion) to see whether data are waiting to be processed. Then, it picks a specified number of records, performs the processing and forwards them to the next stage. If less than the specified records exist in the queue, then they are all retrieved. If the queue is empty, then the invocation is postponed, until there exist data to be processed.

The role of the active data staging area is versatile: (a) it performs all the necessary cleansings and transformations, (b) it relieves the Source from having to perform these tasks, (c) it can act as a regulator for the data warehouse, too (in case, the warehouse cannot handle the online traffic generated by the source) and (d) it can perform various tasks such as checkpointing, summary preparation, and quality of service management.

Once all ETL processing is over, data are ready to be loaded to the warehouse. As already explained, we chose to perform this task through a heavy but reliable (syntactically and operationally) middleware, Web Services. For each target table or materialized view at the warehouse, we define a receiving web service. To be able to invoke the web service, a client needs to be constructed. The client, in order to regulate the traffic between the staging area and the warehouse, compiles the data in blocks, too. The web service at the warehouse side then populates the target table it serves. Load-balancing mechanisms at the warehouse side and physical warehouse maintenance (e.g., index maintenance) can also be part of this architecture. Still, for the moment, we do not consider these possibilities.

In the particular implementation that we have used in our experiments, we have studied the problem as it appears over legacy sources. In our configuration, the Source includes two software modules: (a) an ISAM file and (b) an application used to modify data in the legacy data source. For manipulating ISAM files, there is a library of ISAM routines that are invoked from the application at the source side. We have modified these library routines in order to replicate the data manipulation commands and send updates towards the staging area. Several ETL queues reside at the staging area performing cleanings transformations and aggregations. Each ETL activity retrieves data from its queue with a constant rate, retrieving a given number of elements in constant timeouts. ETL activities communicate both with each other and with the Web service clients via Java thread safe queues. The transfer from the staging area towards the Data Warehouse is done over HTTP (implying TCP as the underlying network protocol). For our experiments, we have assumed that the warehouse simply stores the data performing no other task.

3.2 Issues Raised

To fulfill all the goals mentioned in Section 1, using the architectural elements described above, there are some issues raised which mainly concern the tuning and configuration of the system. The key issues that affect system performance and need to be resolved are discussed in this section and classified with respect to their locality at the source or the staging area, as well as the overall setup of the environment. All the technical choices and their alternatives are summarized in Table 3.1.

3.2.1 Choices concerning the Topology

Having described our architectural elements, the next step is to decide their topology. Our architecture offers the ability of selecting different number of tiers. Several choices exist:

One-tier architecture. Using the one-tier architecture is the simplest solution, overloading however the single host. In any case, data warehouses were introduced exactly for the purpose of separating the source production systems from decision support applications for performance reasons (practically due to transactional deadlocks and system overload). Under these considerations, a single tier approach is not recommended. This solution is illustrated in Figure 3.2.

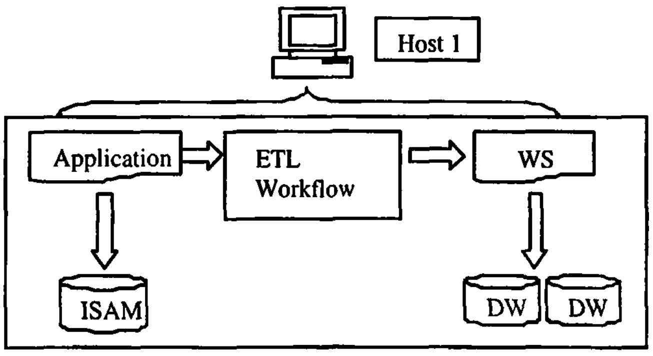


Fig. 3.2. One tier topology: The Source, the Staging Area and the Data Warehouse reside on the same host

As the single-tier alternative is not the most realistic case, we proceed to a two-tier architecture, where the source and the Warehouse are found on different machines.

Two-tier architecture. The source and the Warehouse are found on different machines. Regarding the two-tier architecture, the main issue that arises is related to the placement of the staging area. There are two alternatives concerning this choice: the first is to place the staging area together with the source, putting the data warehouse on a separate machine

(Figure 3.3). The second alternative is to place the staging area at the host where the data warehouse resides (Figure 3.4).

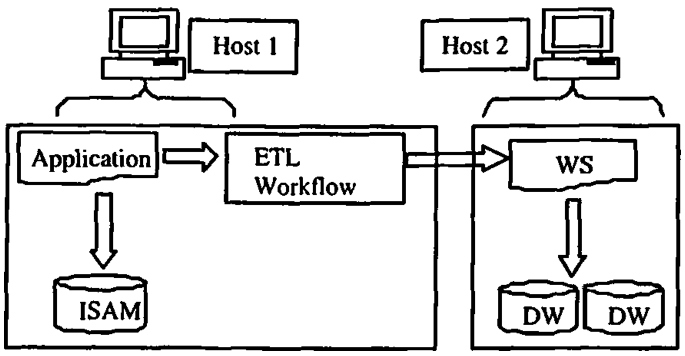


Fig. 3.3. Two-tier topology: The Source and the Staging Area reside on the same host, while the Data Warehouse resides on another machine.

In the case of the staging area placed at the Source, data warehousing operations do not burden the Source, but still the resources used by the web services API to perform the invocation remain considerable. A way for dealing with this is to move the staging area to the warehouse host (Figure 3.4), which can be expected to be more powerful from the source host.

This way, the source is completely detached from the Active Data Warehousing process. Naturally, if the warehouse server is too loaded or its configuration too complex for the extra software setup of a web service server, a three-tier architecture can also be employed.

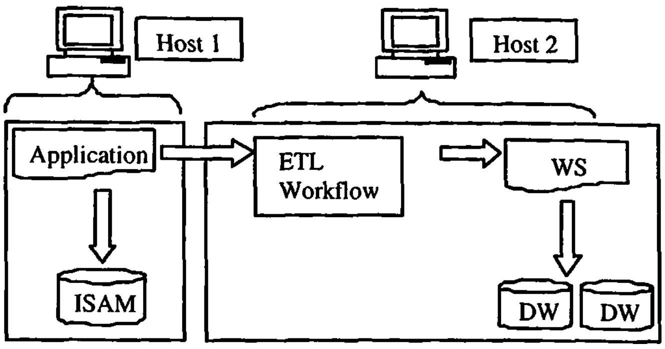


Fig. 3.4. Two tier topology: The Data Warehouse and the Staging Area reside on the same host, while the Source resides on a separate machine.

Three-tier architecture. Finally, there is always the alternative to use a separate dedicated machine for the staging area, leading to a three-tier topology. Using the three-tier architecture (Figure 3.5) solves all the abovementioned problems, but increases the setup and maintenance cost, since an extra server, apart from the one used from the warehouse, has to be engaged and administered.

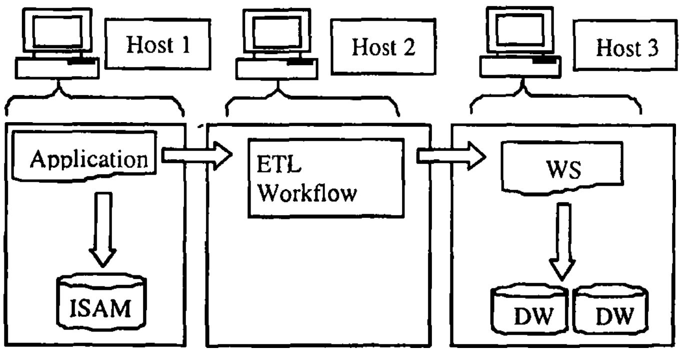


Fig. 3.5. Three tier topology: The Source, the Data Warehouse and the Staging Area reside on three separate machines.

Having discussed the architectural alternatives for our topology, we proceed to discuss the technical issues raised for each of the main components and their overall setup.

3.2.2 Choices concerning the Source

Concerning the source side, the first consideration that arises has to do with the interconnection type between the Source and the staging area. Since our goals are to impose as little impact as possible to the Source and to make only minor changes, we have chosen the solution of sockets both due to its footprint characteristics and the easiness of programming such a solution.

The next choice is between TCP and UDP protocols for the transmission of data between the source and the staging area. On the one hand, TCP offers reliability. On the other hand, UDP offers speed through non blocking calls followed by a concern on the server side for the socket buffer size, in case of extended datagram bursts and no reliability.

A third architectural choice that has to be made concerns the way changes to the source file are written to the socket, i.e., whether data are organized in blocks before being further propagated to the staging area. There are two ways to deal with this issue: either to write each modification to the socket, or to write bulks of modification commands. In the first case, whenever a data manipulation command is issued, it is immediately written to the socket along with the respective data. In the second case, nothing is written, until a number of records is completed. Then, all records together are sent to the staging area. Sending one record at a time, while being a straightforward solution burdens the system with additional communication cost. On the other hand, using a block has the drawback of determining the block size, but reduces communication cost significantly.

3.2.3 Choices concerning the Staging Area

The internal structure of the data staging area and the tuning of its operation are the major issues concerning the performance of our architecture. The staging area is a multithreaded environment with shared components, thus having to be set up properly to avoid race conditions and ensure consistency. Each transformation is implemented as an independent thread. The part of the Staging Area listening for connections from the Source and the Web Service Client are implemented as Threads as well. All of these components communicate with each other with shared queues, making locking necessary to ensure consistency and avoid race conditions.

Issue	Alternatives
General Architecture	
Topology	- 2-tier, Staging Area at the source side - 2-tier, Staging Area at the DW side - 3 tier
Source	
Connection Type	- UDP - TCP
Propagation Type	- One at a time - Block-based
Active Data Staging Area	
Interface between the two APIs	- None - Synchronized Queue
Web Service invocation type	- Blocking - Non Blocking
Propagation Type	- One at a time - Block-based
Data Warehouse	
Session management	- Single WS - Instance per session - Instance per request

Table 3.1. Architectural choices

The problem of locking raises the issue of the queue emptying rate. Assuming that the input to the staging area is determined by the workload of the source (i.e., it cannot be constrained by the warehouse administrator), a proper emptying rate for the ETL queues has to be determined. A high arrival rate compared to the configured service rate will result in instability and queue length explosion. On the contrary, a very high service rate potentially results in too many locks of the queue (resulting again in delay, contrary to what would normally be expected). It is obvious that the service rate should be close to the arrival rate in order to have both efficient service times, and as less locks as possible.

Another dilemma is related to the interconnection type between the Staging Area and the Data Warehouse. As already mentioned, the Staging Area invokes a Web Service residing at the warehouse side. There are two different alternatives for invoking the Web Service, namely (a) blocking and (b) non-blocking. Blocking invocation involves an acknowledgment message to be sent from the web service, before its client can continue. In our case, this means that a response from the warehouse is required, delaying however the queue emptying rate. Non-blocking invocation does not delay the queue emptying process of the web Service client, since no response is returned from the invocation.

Finally, the issue of sending data as tuple-at-a-time or blocks is raised again for the communication between the Staging Area and the warehouse. In this case, apart from the network overhead, the cost of parsing the incoming web service messages at the warehouse plays a role for this choice.

3.2.4 Choices concerning the Warehouse

The data warehouse side is characterized by a Web Service per target table, receiving the cleansed data from the Staging Area. The web services API offers three ways of handling the remote invocations of the client that resides in the data Staging Area. The first way is to create a single web service instance that handles all incoming requests. This is a good solution for configurations where a small number of clients creates a lot of invocation requests, but is not recommended in cases where a large number of clients wish to invoke the same Web Service, since it will result in high latency times. In this case, the second way for handling remote invocations is recommended. That is to create an instance for each invocation request. However, in cases of high frequency invocation requests, this solution behaves poorly in terms of performance, since it creates a new instance of an invocation object. The third alternative is a solution combining the functionality of the previous two cases: an instance is created for every session. The only issue that has to be resolved in this case is the duration of the session time.

In our configurations, we use the first of these alternatives, namely a single web service instance that handles all incoming requests. The reason is that in our experiments, we have employed one client for the service, which stops its operation after inserting a specific amount of records into the ISAM file. This makes the case of using an instance per session the same as using a single instance. Using an object per request is prohibitive, since we assume high frequency invocations.

4 THEORETICAL ANALYSIS OF ETL WORKFLOWS FOR ACTIVE WAREHOUSING USING QUEUE THEORY

In our architecture, data flows from the source to the Staging Area, where data sustain various types of processing: filtering, transformation and binary operations. To establish a cost model for our system and to calculate interesting performance measures such as the delay of extracting results to the warehouse, we use queue theory as the means of acquiring estimations for each case of operation.

The roadmap of this chapter is summarized as follows: in Section 4.1 the definition of the generic queuing model is presented. In Section 4.2 lays the proof of the memoryless property of the exponential distribution. Section 4.3 provides a brief introduction to Markov chains. Section 4.4 displays Kendall's Notation and Little's Law. Kendall's notation is a standard way to describe a queuing system in terms of its input, output and internal architecture. Little's law is the fundamental law governing all queuing systems and relates the input and the output rate of the system with the average queue size. Section 4.5 presents measures of effectiveness valid for all types of queuing systems. Section 4.6 describes the simplest queuing system of all, M/M/1. Section 4.7 presents the $M/M^{[K]}/1$ system which is an extension of the M/M/1 system to model customers serviced in batches. This will be used to describe the function of the source's flow regulator. Section 4.8 presents the case of constant service times, i.e., the M/D/1 system. In Section 4.9, we present methods for dealing with networks of queues and in Section 4.10 we describe how this method is extended for the case of multiple classes of customers. In Section 4.11, we distinguish data operations occurring in the Staging Area in categories and argue about the type of queuing model that can be used to describe each of them.

4.1 Definition of a Queuing Model

The simplest queuing model is depicted in Figure 4.1. It can be used to model machines or operators processing orders or communication equipment processing information. According to this model, a sequence of customers arrives at a server. If a customer arriving at the server finds the server occupied, it waits in the queue until its turn comes to be served. After the customer is served, it leaves the system [Magl].

If λ customers arrive at the system per time unit, then the mean inter-arrival time is equal to $1/\lambda$. Similarly, if μ customers are served at the system per time unit, then the mean service time is equal to $1/\mu$. Based on these parameters, we also define $\rho=\lambda/\mu$ as the traffic intensity which denotes the server utilization. We require that $\rho<1$ or the queue length will explode. In figure 4.1, a basic queuing model is depicted.

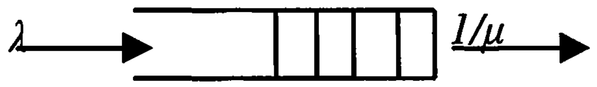


Fig. 4.1. Basic queuing model

Among others, a queuing model is characterized by [AdRe01]:

- *The arrival process of customers.* Usually, we assume that the interarrival times are independent and have a common distribution. In many practical situations, customers arrive according to a Poisson stream (i.e. exponential interarrival times). Customers may arrive one by one, or in batches. An example of batch arrivals is the customs office at the border where travel documents of bus passengers have to be checked.

- *The service times.* Usually we assume that the service times are independent and identically distributed, and that they are independent of the interarrival times. For example, the service times can be deterministic or exponentially distributed. It can also occur that service times are dependent on the queue length. For example, the processing rates of the machines in a production system can be increased once the number of jobs waiting to be processed becomes too large.

- *The service discipline.* Customers can be served one by one or in batches. There are many alternatives for the order in which they enter service. A common discipline that will be used henceforth is the First Come First Served discipline (FCFS), i.e. customers are served in order of arrival.

- *The service capacity.* There may be a single server or a group of servers helping the customers.

- *The waiting room.* There can be limitations with respect to the number of customers in the system. For example, in a data communication network, only finitely many cells can be buffered in a switch. The determination of good buffer sizes is an important issue in the design of these networks.

4.2 Statistical Distribution of Interarrival and Processing Times

In this section the proof of the memoryless property of the exponential distribution is presented. This property is very important because it denotes that *the arrival time of an event in a system is not dependant on the arrivals of previous events and does not affect future arrivals.*

A poisson random variable X has the following distribution:

$$P(X = n) = \frac{\mu^n}{n!} e^{-\mu}, n = 0,1,....$$

The time between successive arrivals following a Poisson distribution follows an exponential distribution. The formula for the distribution function is:

$$F(t) = 1 - e^{-\mu t}, t \geq 0$$

The density of an exponential distribution with parameter μ is given by

$$f(t) = \mu e^{-\mu t}, t > 0$$

The exponential distribution is the only distribution which has the memoryless property [Will04, p. 66-68]. A real-valued non-negative random variable X is called *memoryless* if for all $s, t \in R_0^+$:

$$P[X > s + t \mid X > s] = P[X > t]$$

Intuitively, this means that the time remaining for the next future event is independent of the time the last event occurred [Magl]. It is easy to prove that the exponential distribution has the memoryless property:

$$P[X > s + t \mid X > s] = \frac{P[X > s + t, X > s]}{P[X > s]} = \frac{P[X > s + t]}{P[X > s]} =$$

$$\frac{e^{-\mu(s+t)}}{e^{-st}} = \frac{e^{-\mu s} e^{-\mu t}}{e^{-st}} = e^{-\mu t} = P[X > t]$$

4.3 Markov Process and Markov Chains

A Markov chain is a discrete-time process for which the future behavior, given the past and the present, depends only on the present and not on the past. Figure 4.2 shows a Markov chain. A Markov process is the continuous-time version of a Markov chain.

A Markov chain, studied at the discrete time points 0,1,2,... is characterized by a set of states S and the transition probabilities p_{ij} between the states. Here, p_{ij} is the probability that the Markov chain is in state j at the next time point, given that it is at state i at the present time point. The matrix P with elements p_{ij} is called the *transition probability matrix* of the Markov chain. The row sums of P are equal to 1.

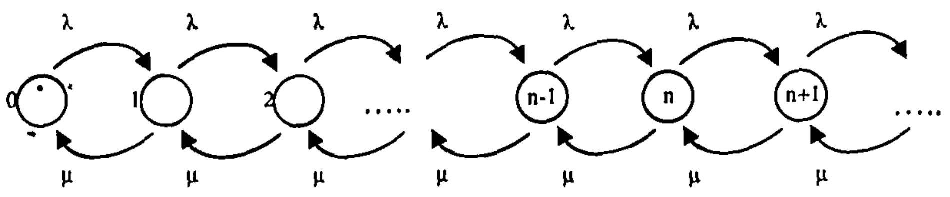


Fig. 4.2. A Markov chain

In a Markov process there is also a discrete set of states S. In each state there are a number of possible events that can cause a transition. The event that causes a transition from state i to j, where $j \neq i$, takes place after an exponential amount of time, with probability q_{ij}. As a result, in this model, transitions take place at random points in time.

A special case of a Markov process is a *birth-death process.* In a birth-death process the only allowed transitions are between neighboring states. The transition from state n to $n+1$ is identified with a *birth* event, and the transition from n to $n-1$ is a *death* event. Many queuing systems can be described as birth-death processes. The Markov process of Figure 2 is a birth-death process, since there are only transitions between neighboring states.

4.4 Kendall's Notation and Little's Law

To characterize different queuing models Kendall introduced a shorthand notation [Magl] using a code of the form *A/B/m/K/M*. Each of these symbols has the following meaning:

- A: Distribution of the inter-arrival times. The following symbols are used to indicate some common distributions: M (exponential / memoryless distribution), G (general distribution), D (Deterministic), E_k (Erlangian distribution with k stages).

- *B*: Distribution of the service times. The aforementioned symbols are also used in this case.

- *m*: Number of servers.

- *K*: size of the queue's waiting room (only used in the case of finite waiting room)

- *M*: Size of population to be served. If not used, infinite population is implied.

Little's Law gives a very important relation between the mean number of customers in the system N, the customer mean arrival rate in the system λ, and the mean time a customer remains in the system T. This relation is formulated as:

$$N = \lambda T$$

The importance of Little 's Law resides in the fact that this equation holds for every type of queuing system irrespectively of the arrival and service distributions.

4.5 Measures of Effectiveness

Measures of effectiveness are measures valid for all types of queuing systems, no matter of the arrival or service rates. In brief, these measures are the following:

- Mean number of customers in the system in steady state:

$$L = E[n] = \sum_{n=0}^{\infty} np_n$$

- Mean number of customers in the queue:

$$L_q = \sum_{n=1}^{\infty} (n-1) p_n$$

In these formulas L_q represents the mean number of customers waiting in the queue to be processed, while L represents the total customers in the system: the customers currently being processed and the customers waiting in the queue.

Using Little's formula it is easy to obtain the expected waiting times for the system, and the waiting queue.

4.6 The M/M/1 Queuing System

M/M/1 is the simplest queuing system and it can be described as follows: FIFO service, single server, infinite waiting line, the customer inter-arrival times are independent, identically and exponentially distributed with some parameter λ. The customer service times are also independent, identically and exponentially distributed with some parameter μ. The assumption of independent and identically distributed variables means that each random variable has the same distribution with the others and they are mutually independent. We can describe this type of queuing system through the following equations [Will04]:

Mean number of customers in the system:

$$L = E[n] = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

Mean number of customers in the queue, or the queue length:

$$L_q = L - (1 - p_0) = \frac{\rho^2}{1-\rho} = \frac{\lambda^2}{\mu(\mu - \lambda)}$$

Mean delay time for a customer in the system:

$$T = \frac{E[n]}{\lambda}$$

Mean delay time for a customer in the queue:

$$T = \frac{L_q}{\lambda}$$

Probability of n customers in the system:

$$p_n = (1-p)p^n$$

A method for obtaining steady state flow equations for a birth - death process is the stochastic balance procedure [GrHa]. This type of analysis looks at a given state and requires that the flow into a state equals to the flow out of the state.

Consider a state *n* (*n≥1*) in an M/M/1 queue. As shown in figure 2, the system goes from state *n* (meaning *n* customers in the system) to state *n-1* when a service completes, or to state *n+1* when an arrival occurs. Similarly, the system goes from state *n-1* to state *n* when a new arrival occurs, and from *n+1* to *n* when a service is completed. The mean flow into state *n* equals to $\mu p_{n+1} + \lambda p_{n-1}$ and the flow out of state *n* is $\mu p_n + \lambda p_n$. Equating incoming flow to outgoing flow results in:

$$\mu p_{n+1} + \lambda p_{n-1} = (\lambda + \mu) p_n, \quad (n \geq 1)$$

In the case that *n=0*, since *n* is non negative the following equation is obtained:

$$\lambda p_0 = \mu p_1$$

4.7 Networks of Queues and the Jackson Theorem

Many queuing systems consist of a network of queues. In a *queuing network* (QN), a customer finishing service in a service facility is either immediately proceeding to another service facility or leaves the system. For our purposes we assume that each node of this network consists of a single server with exponential arrival and exponential service times.

One basic classification of queuing networks is the distinction between *open* and *closed* queuing networks. In an open network new customers may arrive from outside (coming from a conceptually infinite population) and later on leave the system. In a closed queuing network the number of customers is fixed and no customer enters or leaves the system. In our case we are exclusively interested in open networks. We will consider only the case of a single class network where all customers belong to the same class, for example share the same service times.

In the following analysis we will use the following notation:

- N: number of nodes (single service centers)
- k_i: the number of jobs at the i-th node. The nodes are numbered from 1 to N. The k_i are grouped into the vector $(k_1, \ldots, k_N)$.
- m_i: the number of parallel servers at node i. All servers have the same service rate.
- μ_i: service rate of all the servers at node i. The overall service rate of this node is $m_i \cdot \mu_i$. The mean service time of a single customer is $1/\mu_i$.
- $p_{i,j}$: the routing probability that a customer leaving node i proceeds to node j. These probabilities remain fixed over time. Clearly, when there is no direct path from i to j we have $p_{ij} = 0$. In our case we assume that the occurring transitions follow a birth – death process, i.e, they occur only between neighboring states.
- $p_{0,j}$: the probability that a new job entering the system from outside enters the system at node j. In an open network the following holds:

$$\sum_{j=1}^{N} p_{0,j} = 1$$

- $p_{i,0}$: the probability that a job leaves the system immediately after getting service at node i.
- $\lambda_{0,i}$: the arrival rate of jobs from outside to node i.
- $\lambda_{0,i}$: the total arrival rate to node i. This includes arrivals both outside the system and from other nodes (including feedback).

- λ: the total arrival rate to all nodes in the network from outside.

$$\lambda = \sum_{i=1}^{N} \lambda_{0,i}$$

- $e_i = \lambda_i/\lambda$ *is the visit ratio* of node *i*, i.e. how often the node is visited by a single job.

If an open queuing network is in steady state then for each node, its arrival rate λ_i equals its departure rate. The arrival rate λ_i to node *i* is clearly the sum of all arrivals from the outside to *i* and from all nodes to *i* (including *i* itself), thus we have:

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{N} p_{j,i}\lambda_j$$

or in vector – matrix form:

$$\lambda = \lambda_0 + \lambda R$$

These equations are called traffic equations and they can be transformed into a set of N simultaneous linear equations. If we divide these equations by λ we arrive at:

$$e_i = p_{0,i} + \sum_{j=1}^{N} p_{j,i}e_j$$

In order to calculate the performance measures in queuing networks the steady state probabilities have to be found:

$$\pi\,(k_1,\ldots,\,k_N) = Pr\,[k_1 \text{ customers in queue } 1,\,\ldots,\,k_N \text{ customers in queue } N]$$

The term $\pi(k_1,\ldots,\,k_N)$ denotes the probability of k_1 customers in queue 1, k_2 customers in queue 2 and so on. The overall throughput of an open queuing network is the rate by which jobs leave the network. If the network is in the steady state, then this rate equals the arrival rate λ from outside to the network.

Jackson's Theorem specifies the conditions, under which a product form solution in open queuing networks exist:

- The number of customers in the network is not limited.
- Every node in the network has Poisson arrivals from outside the network.
- A customer can leave the system from any node (or a subset of them).
- All service times are exponentially distributed.
- In every node the service discipline is FIFO.
- The *i*-th service facility consists of mi identical servers, each with service rate μ_i (as a generalization the service rate μ_i may depend on the number of customers in system *i*).

Jackson's Theorem: If in an open network the condition $\lambda_i < \mu_i \cdot m_i$ holds for every $i \in \{1, ..,N\}$ then the steady state probability of the network can be expressed as the product of the state probabilities of the individual nodes:

$$\pi (k_1,..., k_N) = \pi_1(k_1)\pi_2(k_2)... \pi_N(k_N)$$

Therefore, we can solve this class of networks in three steps:

- Solve the traffic equations to find λ_i for each queuing system *i*.
- For each queuing system *i*, determine separately its steady-state probabilities $\pi_i(k_i)$.
- Determine the global steady-state probabilities $\pi (k_1,..., k_N)$ from the above formula.
- Derive the desired global performance measures.

Jackson 's theorem offers a very important result since it allows us to calculate, in a straightforward way, the steady state probabilities of the whole network by calculating the probabilities by treating each node separately.

4.8 Multi Class Jackson Networks

A generalization to Jackson networks is a Jackson network with different classes of customers [GrHa]. In specific, customers of different classes have different routing probabilities, depending on the class they belong. To solve such networks, we assume a separate routing matrix $R^{(t)}$ for each customer class, where the superscript (t) represents the class of the customers. The routing equations are solved separately for each class of customers. The formula describing the arrival rate in a node for class t in vector – matrix form:

$$\lambda^{(t)} = \lambda_0^{(t)} + \lambda^{(t)} R^{(t)}$$

The overall input rate λ in the network equals to the sum of the input rate of the network for each class of customers. This can be formulated as:

$$\lambda = \Sigma \lambda^{(t)}$$

To obtain individually the average number of customers of class t in node i, denoted as $L_i^{(t)}$, the following equation holds:

$$L_i^{(t)} = \frac{\lambda_i^{(t)}}{\lambda_i^{(1)} + \lambda_i^{(2)} + \ldots + \lambda_i^{(n)}} L_i$$

Again L_i*'s* are computed by the M/M/1 formulas, since we can treat each node separately.

4.9 Applying Queue Theory to ETL Workflows

Each ETL queue can direct customers to more than one subsequent queue, depending on the type of operation it performs. The composition of queues in queue theory is treated by queue networks and the computation of the interesting properties of such networks depends on the nature of the involved individual queues. The question that arises is what kind of individual queues do the ETL activities produce. One possible way to answer this question is to define an extension of the relational algebra, specifically tailored for ETL purposes, containing for instance, operators for answering continuous queries and study the properties of each operator from the viewpoint of queue theory. Since this would probably produce quite complex queues, we adopt a different, black-box, approach and define a taxonomy of

ETL transformations, based on the relationship of their input and output. This way, we practically categorize ETL tasks in families without delving in the particularities of their internal functionality. Specifically, the taxonomy of activities consists of the following categories: (a) Filters, (b) Transformers and (c) Binary Operations.

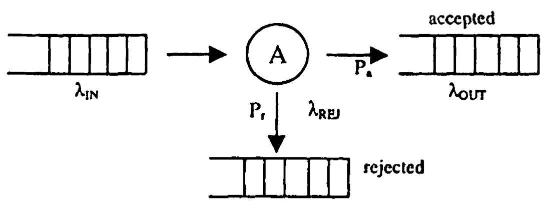


Fig. 4.3. Queuing model for Multi-output activities

4.9.1 Filters

Filters examine each incoming tuple to determine whether it meets certain criteria. If these criteria are fulfilled, then a tuple is accepted and propagated towards an acceptance output. If not, it is rejected and possibly propagated towards a rejection output. We assume that tuple arrivals occur due to a Poisson process and service times follow an exponential distribution. We define the probability that some tuple *i* is accepted as P_a and the probability that some tuple *i* is rejected by the system as P_r. This is illustrated in Figure 4.3. It is obvious that $P_a + P_r = 1$.

The filtering operations do not impose a change in the overall number of tuples making the following equation valid:

$$|\,\textit{tuples entering service}\,| = |\,\textit{tuples accepted}\,| + |\,\textit{tuples rejected}\,|$$

Also, these operations do not incur changes to the schema of the tuples entering the service facility compared to the schema of the tuples exiting. Typical operations of this category are not null, domain and foreign key checks, selections and in general, any type of operation, operating locally on a tuple and determining whether it will be further propagated or not. Due to their multiple outputs, filters can also act as routers for tuples whose destination depends on their value.

4.9.2 Transformers

Considering the case of Transformers, tuples entering a transformer undergo changes to their value and/or their schema. We can distinguish two subclasses of Transformers taking into account the relationship between the number of tuples entering and the number of tuples exiting the transformation. In the first case, the two quantities are equal which means:

| *tuples entering service* | = | *tuples accepted* |

We assume that tuple arrivals occur due to a Poisson process and service times follow an exponential distribution, in other words, we have the same case with filters transformations. Again, we define the probability that some tuple *i* is accepted as P_a and the probability that some tuple *i* is rejected by the system as P_r. Since all tuples are accepted, we have: $P_a = 1$ and $P_r = 0$ (Figure 4.4). An example of such a transformation is the surrogate key transformation, the usage of functions for the derivation of new values and, in general, any transformation that derives an output tuple solely on the basis of the value of a single input tuple.

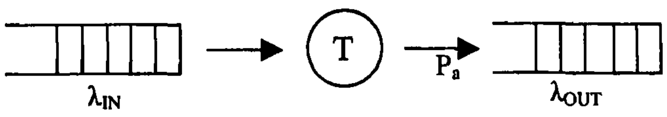


Fig. 4.4. Queuing model for Single-output activities

In the second case, the number of tuples entering the system is different compared to the number of tuples exiting and in specific:

| *tuples entering service* | > | *tuples accepted* |

This occurs because some of the tuples entering service are aggregated or merged. We assume that tuple arrivals occur due to a Poisson process and service times follow an exponential distribution. The problem with this kind of transformations is that practically queue customers disappear and new customers are produced by each transformation. To model this property in terms of queue theory, we make the assumption that depending on the aggregation or merging factor, some of the incoming customers continue and some exit the system. In other words, we assume that some of the tuples after being transformed continue through the system as accepted. The number of these tuples equals the number of tuples produced as a result by the transformation. The rest of the tuples are assumed to be rejected by the system after their service and exit the system. The following obvious equation holds:

$$|\,\textit{tuples rejected}\,| = |\,\textit{tuples entering service}\,| - |\,\textit{tuples accepted}\,|$$

Again, we define as P_a the probability that some tuple i is accepted and P_r the probability that some tuple i is rejected by the system: $P_a + P_r = 1$. Given the aggregation factor of an incoming set of data, we can easily compute the output and rejection rate as well as the respective routing probabilities on the basis of the number of tuples we wish to aggregate each time, or more generally, to impose a transformation of this category. Thus, the routing probabilities are:

$$P_a = \frac{|result\_tuples|}{|input\_tuples|} \text{ and } P_r = \frac{|input\_tuples| - |result\_tuples|}{|input\_tuples|}$$

4.9.3 Binary Operators

The third class of ETL activities deals with Binary Operators. This is the case where data from multiple sources are combined and a single outgoing stream is produced. Examples of such operations involve variants of the join operation, including the join of data from different tables, as well as difference and update detection operations among different snapshots of the same table. [JiCh03] describes a window-based hash join algorithm for continuous streams. In the context of ETL, we make the following assumptions and observations:

- One of the two inputs is consider as the *primary input flow*. Tuples of this flow are checked over filters or transformed according to the values of some other relation and ultimately, either propagated towards the Warehouse or rejected.

- The second input of the operator is acting as a *regulator of the primary flow*. In other words, its values are only needed in order to determine the processing and routing of the tuples of the primary flow. For all practical purposes, where active ETL functionality is needed (update detection, difference, facts joined with dimension values), a static snapshot of the regulator flow can even be assumed.

- Adopting the model of [JiCh03], both inputs arrive at the same queue – they simply undergo processing with different distributions of processing times.

In principle, a binary operator has to be dealt with as a multi-class queuing system, with one class for each flow (input or output) – see Figure 4.5. We refer the interested reader to [JiCh03] for such a treatment. Still, based on the aforementioned assumptions, we can avoid modeling the system as a multi-class queue and deal only with the primary flow of the operator. In the rest of the thesis, we will consider single-class queues, the tuples of which (a) either continue in the system or (b) are ultimately rejected. An interesting observation here is that no matter how many different categories of tuples enter the node for service, the output tuples can be assumed to belong in one of the two aforementioned categories.

We consider Poisson arrivals and exponential service times. As stated earlier the two routing classes are accepted with probability P_a and rejected with probability P_r and as before $P_a + P_r = 1$. This type of operations does not impose a change in the overall number of tuples existing making the following equation valid (Figure 4.5):

$$| \textit{tuples entering service} | = | \textit{tuples accepted} | + | \textit{tuples rejected} |$$

However, differently from Filters, the schema of the tuples possibly changes.

4.9.4 Generic Model

It is easy to observe, that a Generic Model can generalize the three aforementioned classes, in a single case where a node consisting of a single server serves possibly more than one class of customers. All customers arrive according to a Poisson process and are serviced with exponential service times. The general case is depicted in Figure 4.5 and involves a M/M/1 queuing node.

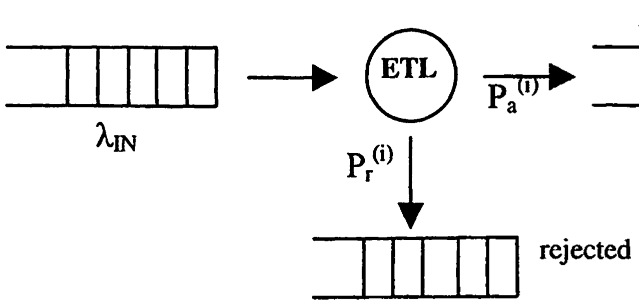


Fig. 4.5. Generic Model for ETL Queues

In the general case, we can assume that tuples of customers, after their ETL transformation at the node, leave the system with probability $P_r^{(i)}$ and continue in the queue network with probability $P_a^{(i)}$. Concerning the number of tuples in the system the following equation is still valid:

$$| \textit{tuples entering service} | = | \textit{tuples accepted} | + | \textit{tuples rejected} |$$

Concerning the schema of the tuples before and after service, we observe that the schema changes in the general case, apart from the case of filters.

In the rest of this thesis, we will follow the assumption of a primary input flow. This obviously results in forming an M/M/1 queuing node as the constructing element of our ETL queue network.

4.10 Methodology for Solving ETL Scenarios

In this section we illustrate the solution of a queue network of ETL operations. We use Queue Theory to predict a performance metric of our system based on the values of the other metrics. Moreover, we provide an example to illustrate the usage of Queue Theory in order to predict the number of packets in the system, knowing the arrival and service rates.

Specifically, given a network of ETL queues, there are three inter-related metrics for each queue: *arrival rate, service rate* and the *number of packets in the queue.* The overall number of packets in the system can easily be determined as the sum of the individual queues. The problem formulation is simple: given the two of the three metric, determine the third one. The methodology for solving such a problem is quite simple. We work on open queuing networks, since tuples arrive from outside (coming from a conceptually infinite population) and later on leave the system. Each node of the network is assumed to be a M/M/1 system. In order to solve this network·we follow these steps:

1. We determine the routing probabilities for each node.
2. We solve the traffic equations system to calculate the arrival rate for each node.
3. We solve for each node separately the M/M/1 equations to calculate performance metrics

To demonstrate the usage of this technique, which comes from queue theory, we assume the following scenario, as illustrated in figure 4.6. In brief, the scenario is as follows:

a. Filter 10% of incoming data.
b. A surrogate key operation to the first column of the filtered data.
c. Group by sum.
d. Data are then fed to the warehouse.

We assign the following identifiers, for reasons of ease, to each node of the network, thus:

F	Filter 10%
SK	Surrogate Key
GB	Group By Sum
WS	Web Service Client
O	Outside the network

Table 4.1. Assigned identifiers

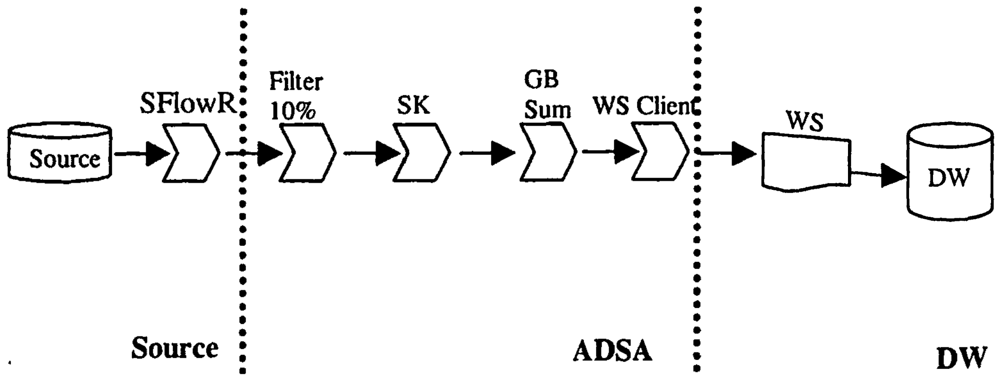


Fig. 4.6. The scenario we solve

The routing probabilities for each node of the Staging Area are the following:

$P_{F,O} = 0.1$	$P_{F,F} = 0$	$P_{F,SK} = 0.9$	$P_{F,GB} = 0$	$P_{F,WS} = 0$
$P_{SK,O} = 0$	$P_{SK,F} = 0$	$P_{SK,SK} = 0$	$P_{SK,GB} = 1$	$P_{SK,WS} = 0$
$P_{GB,O} = 0$	$P_{GB,F} = 0$	$P_{GB,SK} = 0$	$P_{GB,GB} = 0$	$P_{GB,WS} = 1$
$P_{WS,O} = 1$	$P_{WS,F} = 0$	$P_{WS,SK} = 0$	$P_{WS,GB} = 0$	$P_{WS,WS} = 0$

Next we solve the traffic equations, which are the following:

$$\lambda_F = \lambda_{O,F} + \lambda_F P_{F,F} + \lambda_{SK} P_{SK,F} + \lambda_{GB} P_{GB,F} + \lambda_{WS} P_{WS,F}$$

$$\lambda_{SK} = \lambda_{O,SK} + \lambda_F P_{F,SK} + \lambda_{SK} P_{SK,SK} + \lambda_{GB} P_{GB,SK} + \lambda_{WS} P_{WS,SK}$$

$$\lambda_{GB} = \lambda_{O,GB} + \lambda_F P_{F,GB} + \lambda_{SK} P_{SK,GB} + \lambda_{GB} P_{GB,GB} + \lambda_{WS} P_{WS,GB}$$

$$\lambda_{WS} = \lambda_{O,WS} + \lambda_F P_{F,WS} + \lambda_{SK} P_{SK,WS} + \lambda_{GB} P_{GB,WS} + \lambda_{WS} P_{WS,WS}$$

The solution results in the following equations:

$$\lambda_F = \lambda_{O,F}$$

$$\lambda_{SK} = \lambda_F P_{F,SK}$$
$$\lambda_{GB} = \lambda_{SK} P_{SK,GB}$$
$$\lambda_{WS} = \lambda_{GB} P_{GB,WS}$$

The measured arrival rate from the Source is:

$$\lambda_O = \lambda_{O,F} = 22.252 \text{ packets / sec}$$

Substituting we get:

$$\lambda_F = 22.252 \text{ packets / sec}$$
$$\lambda_{SK} = 22.252 \text{ x } 0.9 = 20.0268 \text{ packets / sec}$$
$$\lambda_{GB} = \lambda_{SK} P_{SK,GB} = 20.0268 \text{ packets / sec}$$
$$\lambda_{WS} = \lambda_{GB} P_{GB,WS} = 20.0268 \text{ packets / sec}$$

Now we can solve for each node separately the M/M/1 equations to obtain the performance metrics. To do this we also need the service rates for each node. We get the following values from our experiments:

$$\mu_F = 26.403$$
$$\mu_{SK} = 25.648$$
$$\mu_{GB} = 27.194$$
$$\mu_{WS} = 28.595$$

Thus, for each case, the average number of packets in the system is the following:

$$L_F = 5.360$$
$$L_{SK} = 3.960$$
$$L_{GB} = 4.502$$
$$L_{WS} = 2.60$$

Similarly, we can easily calculate the rest of the performance rates using the equations that hold for the M/M/1 queuing system.

5 EXPERIMENTS ON ARCHITECTURE WITHOUT ETL PROCESSING

In this Chapter, we present the experiments we conducted to determine the best configuration for our architecture.

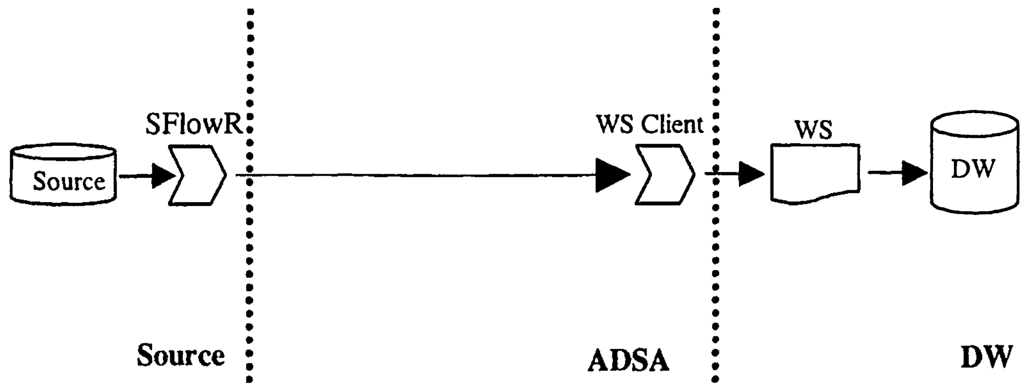


Fig. 5.1. Experimental setup for set (a)

We present two sets of experiments: (a) and (b). Set (a) presented in this chapter, deals with the general behavior of the system. The purpose of this set of experiments is to figure out the behavior of each system component separately, and to establish guidelines for building the system. In this case, data are just transferred to the warehouse and no ETL operations are involved. In set (b), presented in Chapter 6, we evaluate the behavior of a system in a realistic setup, based on the conclusions derived from the first set. Naturally, in this case, we also transform out data using ETL operations.

Our experimental setup is as follows: In our configuration, the Source includes two software modules: (a) an ISAM file and (b) an application used to modify data in the legacy data source. To manipulate ISAM files, there is a library of ISAM routines that are invoked from the application at the source side. We have modified these library routines in order to replicate the data manipulation commands and send updates towards the Staging Area. The ISAM library that we altered is the PBL/ISAM suite [PBL04] available under GPL license. We have used a sample program distributed within the suite as the legacy application. We use two different data sets for our purposes. The first consists of 100,000 records and the second of 1,000,000 records. The ETL queues of the Staging Area have been implemented using the Sun JDK 1.4, whose runtime engine has also been used. As a Web Services platform we have used Apache Axis 1.1 [AXIS04] with Xerces XML parser running over Apache Tomcat 1.3.29. Our Data Warehouse is implemented as a MySQL 4.1 database.

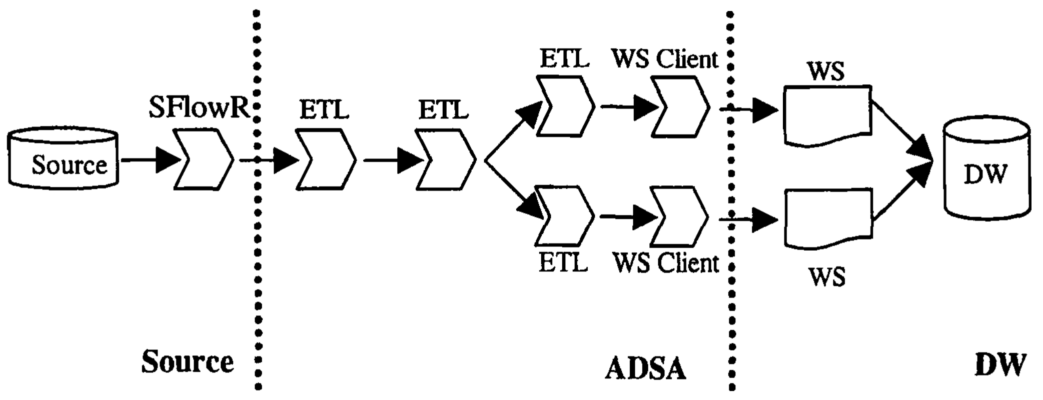


Fig. 5.1. Experimental setup for set (b)

The host we used for the Source was a PIII·700MHz with 256MB of physical memory running SuSE Linux 8.1. The host used as the data warehouse a Pentium 4 2.8GHz with 1GB of physical memory running Mandrake Linux. This server also hosted the Staging Area. The hosts are interconnected via the switched Fast Ethernet LAN of our department. Our data were created from the TPC-H data generation tool. For set (a) each row of data has fixed size equal to 20 bytes. For set (b), where we evaluate the system behavior under real data warehousing conditions, we used data of variable size. In this case, each row has an average size of 140 bytes.

In our experiments, we evaluate the cost in marginal conditions. Thus in order to evaluate the worst case, the Source generates data at its peak capability. Moreover, since our warehouse host is a much faster computer than the source host, we would not be able to make safe conclusions if we always let it operate at its full capability. Thus, we simulate slower server performance by employing time outs between operations. This will be explained in more detail later.

5.1 Smooth Upgrade

One of the goals of our architecture is to pose minimal modifications to code of the source system. In our approach, we do not alter the Legacy Application itself, but the library which manipulates the ISAM files by adding a few lines of code to the routines that are of interest to the purpose of active warehousing. These routines are: the file opening routine, the record insertion routine and the file closing routine. The alterations are located only in 4 points of the library's source code:

1. The first modification is to include our library which contains the socket's client and the Source Flow Regulator.
2. The second modification is to add a call to the routine of our library that opens a socket to the Staging Area at the ISAM file opening routine. This call is performed only if the opening of the ISAM file is successful.
3. The third modification is to also add the call to our library's function that writes to the socket at the routine of the ISAM file library that writes the record to the file. This routine stores the specific record to the Source Flow Regulator's buffer and when the defined number of records is completed, it delivers them to the Staging Area. Again, this routine is called only after a successful insertion.
4. The fourth modification is to add a call to the routine of our library that closes the opened socket to the Staging Area, at the ISAM file closing routine. This call is performed only if the closing of the ISAM file is successful.

Table 3 shows the alterations that we have performed to the library in pseudo-code. The overall length of code that had to be written for this part of the implementation, including the additions at the ISAM library, is approximately 100 lines.

Table 3. Code alterations at the routine that opens the ISAM file.

Original Routine	Altered Routine
	Open_isam_File()
Open_isam_File()	{
{	...
...	opening_isam_file_commands
opening_isam_file_commands	...
...	if(open==success)
}	ADSA_socket_open()
	}
	Write_record_to_File()
Write_record_to_File()	{
{	...
...	insert_record_commands
insert_record_commands	...
...	if(write==success)
}	write_to_Source Flow RegulatorR()
	}
	Close_isam_File()
Close_isam_File()	{
{	...
...	closing_isam_file_commands
closing_isam_file_commands	...
...	if(close==success)
}	ADSA_socket_close()
	}

The routine that opens the socket to the Staging Area reads configuration information from a plain text file, before the opening of the socket. This file contains the following three pieces of information:

1. The number of records the Source Flow Regulator will gather
2. The address of the Staging Area
3. The port of the Staging Area

As an overall assessment of the impact of our changes, we can say that (a) minimal code had to be written to achieve the replication of incoming updates to the warehouse in an active fashion, (b) simple configuration parameters are required, (c) no changes were required to the code, rather than a simple recompilation under the new library.

5.2 UDP vs. TCP

The first parameter that needed to be tested involved the network protocol between the source and the Staging Area. The goal of our first experiment is to determine the system's behavior using UDP and specifically if there are any datagram losses. In the graph shown in Figure 5.3, the results of sending 100,000 records from the Source to the Staging Area using UDP are shown. The Staging Area uses a queue and performs asynchronous invocation.

The results show a 35% packet loss of data, most probably due to the overflowing of data. Such losses are prohibitive for normal operation of an on-line environment. Therefore, for the rest of the thesis, we have fixed TCP as the interconnection protocol between the Source and the Staging Area.

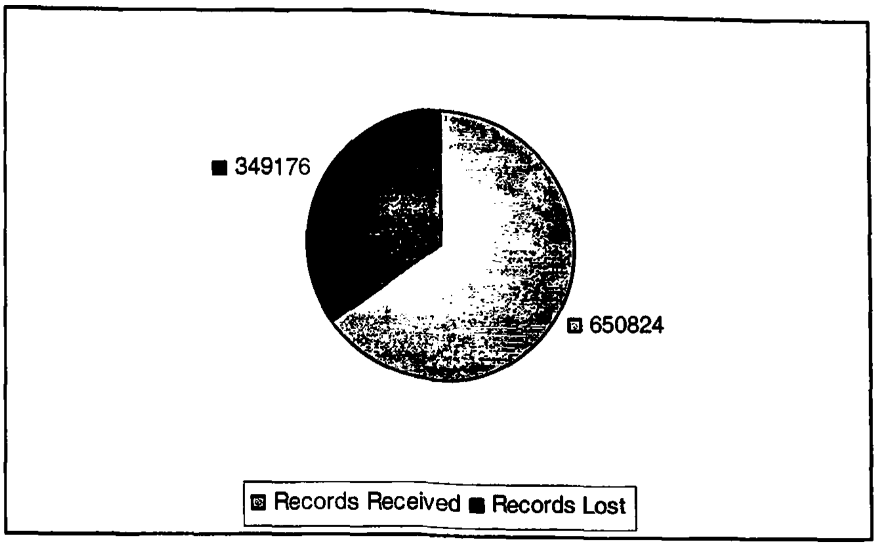


Fig. 5.3. Data loss using UDP as the interconnection protocol between the Source and the Staging Area

5.3 Overhead at Source

The main requirement for the architecture at the source side involves minimal overhead during regular operation. Therefore, the goal of the next experiment is to measure the overhead that our configuration incurs at the Source side. We measure the time to complete the insertion of (a) 100 thousand records and (b) 1 million records to the ISAM file. We experiment on the impact of the following two parameters:

- First, we measure the effect of using the Source Flow Regulator. We try three values: 1, 100, and 1000 records for each packet that the Source Flow Regulator sends to the Staging Area (see the x-axis for Figures 5.4 and 5.5). When using one record at a package, we have in fact the case of not using a Source Flow Regulator.

- Second, we experiment with the behavior of the source in terms of completion time. In experiment set (a) that we call "plain", the source performs its regular operation during normal time. In this case, no records are propagated to the Staging Area.

Another issue worth investigating is the potential impact that the tuning of the Staging Area has over the source. Therefore, except for the two parameters that we have already described (both of which concern choices at the source side), we employ two modes for the operation of the Staging Area, for assessing its impact. Each test case is examined with blocking and non-blocking invocation for the communication between the Staging Area and the Web Service. We assume that the Staging Area uses a synchronized queue. The input rate at the queue is equal to the output rate of the Legacy Application. The queue's output rate is fixed to one thousand records per second.

The y-axis of the diagrams measures the throughput of inserting the records to the ISAM file. Figure 5.2 depicts the results of the experiment for 100,000 records, while Figure 5.3 the results for 1,000,000 records.

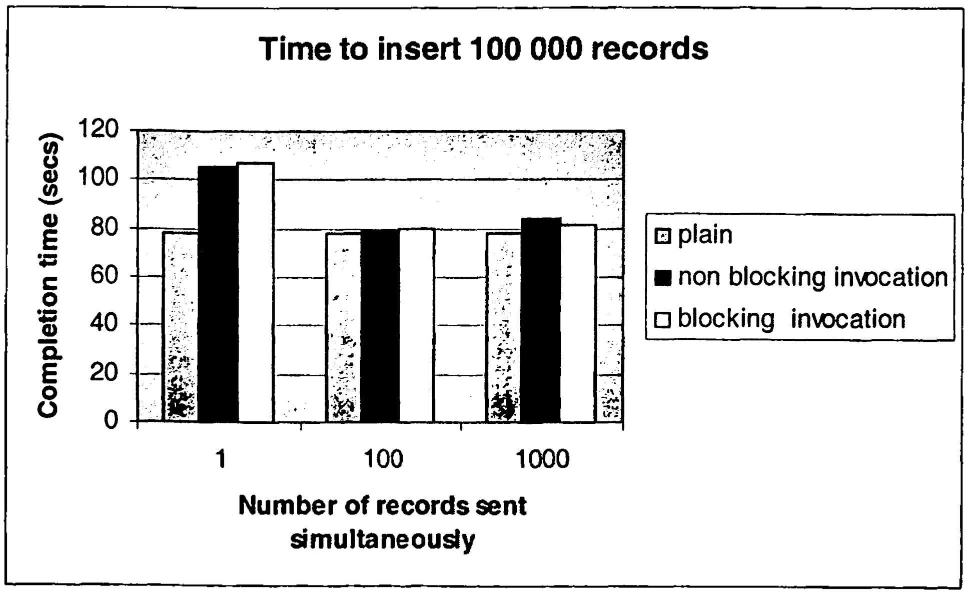


Fig. 5.4. Time to insert 100,000 records using a two-tier topology

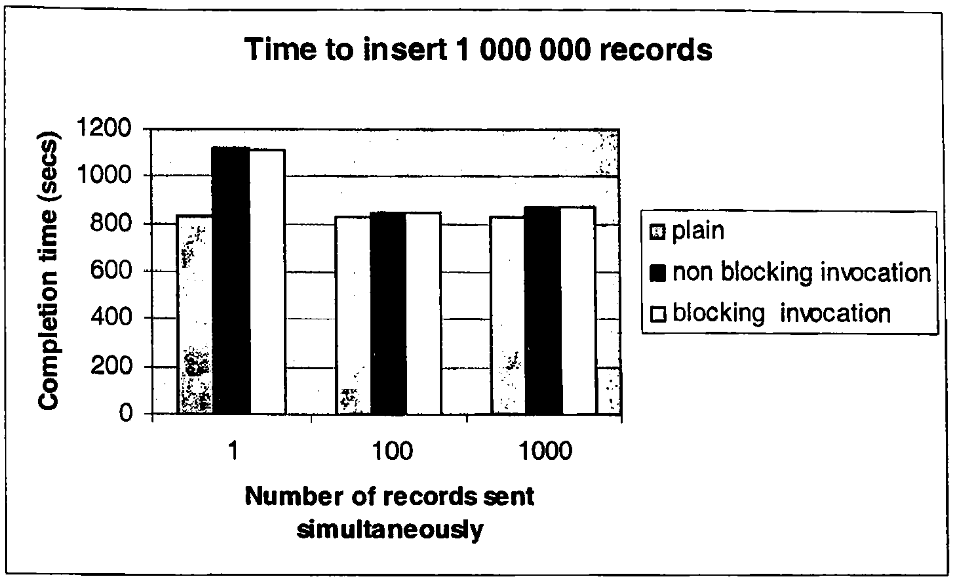


Fig. 5.5. Time to insert 1,000,000 records using a two-tier topology

Based on our experimental results the following observations are made:

(a) The Source Flow Regulator plays a very important role, since without it, the throughput deteriorates by 34 %, while using a Source Flow Regulator incurs an impact of 1.7%.

(b) The way that the Staging Area is tuned does not affect the source. Regardless of using blocking or non-blocking Web Service invocation at the Staging Area, the Source's throughput is the same in both cases. This is a key observation for our architecture, since it proves that the operations of each tier are independent. Thus, we can examine each tier separately.

(c) Sending smaller packets of records performs slightly better, since in the case of 1000 records, network propagation time decreases throughput. Moreover, choosing a packet size of 100 instead of 1000 records saves buffer size at the Source Flow Regulator.

5.4 Number of Queues

In this experiment, we examine how the number of queues impacts the Source's performance. Our measurements concern the time to complete the insertion of 100,000 records to the ISAM file. We experiment to determine the necessary number of queues both at the Source and at the Staging Area, as indicated on the x-axis of Figure 5.6. We have conducted experiments with the following four settings.

1. Initially we have used no queues at all, which implies the absence of the Source Flow Regulator at the source and the absence of the queue at the Staging Area.
2. In the second setting we have used only the Source Flow Regulator at the Source without the queue at the Staging Area.
3. Our third experiment's setting involves the absence of the Source Flow Regulator and the presence of the Staging Area's queue.
4. Finally, in our fourth experiment, we have used our architecture in full deployment: The Source Flow Regulator at the queue and the queue at the Staging Area.

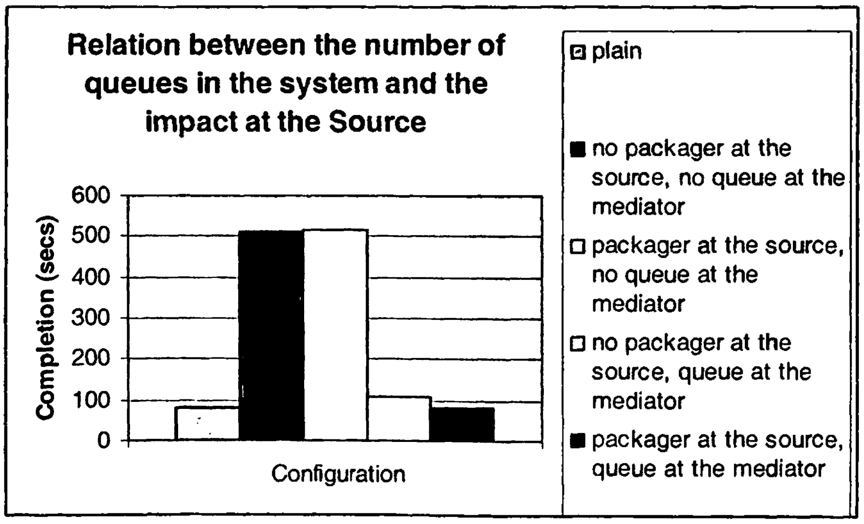


Fig. 5.6. Time to insert 100 000 records using two-tier topology

Figure 5.4 depicts the results of the experiment for 100 000 records. The y-axis measures the throughput of inserting the records to the ISAM file. Plain depicts the case where no propagation towards the warehouse was performed (standing again for the stand-alone behavior of the source).

Based on our experimental results the following observations are made:

(a) Not using a queue in the Staging Area poses the greatest impact in the system, no matter whether Source Flow Regulator is used or not. This impact is 553.40% of the time the Legacy Application requires to insert 100 000 records, in the case of not using any kind of queue in the system and 553.76% using only a Source Flow Regulator.

(b) Using only a queue at the Staging Area increases system performance with respect to the previous configurations, but the overhead at the Source is still considerable and measured at 36.19%.

(c) Using both a Source Flow Regulator at the Source and a queue at the Staging Area provides the best system performance adding the smallest overhead to the Source. This overhead is measured at 1.78% of the plain Source time.

5.5 Data Freshness

A major requirement in our setting is to achieve the maximum data freshness possible, through our framework. With a 1.78% delay at the source, as derived from the experiments of Section 5.4, the focus of interest is isolated at the side of the Staging Area. The goal of the next set of experiments is to measure the data freshness time provided by our application with respect to the queue emptying rate and the block retrieved from the queue. We consider as *data freshness time* the time required for a record that was inserted in the ISAM file to be transferred to the warehouse.

Specifically, we measure the overall throughput, i.e., the time needed to empty the Staging Area's queue after the first record is sent to the warehouse. The freshness is then measured as the time needed to empty the queue, which practically stands for the response time for the last record. To perform these measurements, we assume that the Legacy Application sends 100,000 records to the Staging Area in packs of 100 records over TCP. Also, we measure the queue length as an indicator of resource consumption at the Staging Area site.

A major parameter affecting the overall performance of our environment is the impact of the block size of records we deliver to the warehouse. Thus we present three sets of experiments emptying the Staging Area 's queue with three different ways using the aforementioned rates:

(a) We empty the queue as soon as possible and then propagate the records to the web service.
(b) We empty the queue retrieving the records from the queue using timeouts of 0.1 seconds retrieving 50, 100, 150, 200, 250 and 300 records each time and then invoking the Web Service.
(c) We empty the queue retrieving the records from the queue using timeouts of 1 second retrieving 500, 1000, 1500, 2000 and 2500 records each time and then invoking the Web Service.

Two other parameters play a major role. The first parameter, as indicated on the x-axis of Figure 5.7, is the time required to empty the queue. The second parameter as shown on y-axis of Figure 5.7, is the number of elements in the Staging Area 's queue. The queue's input rate is equal to the Source's output rate, i.e. 1250 records per second approximately. We experimented using the following queue emptying rates: 500, 1000, 1500, 2000, 2500 and 3000 records per second. These are the maximal emptying rates, which means that, if the queue contains fewer records, then all the records from the queue are retrieved.

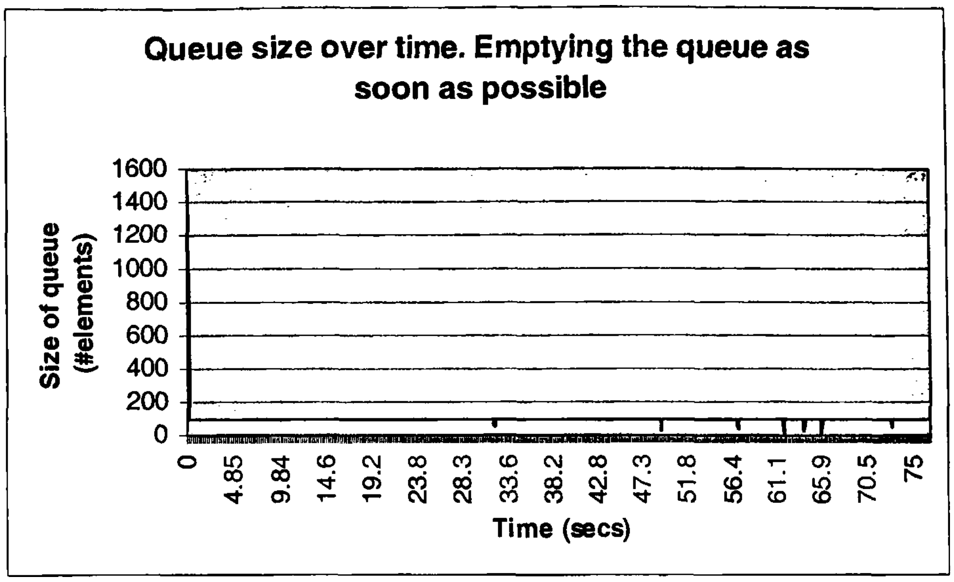


Fig 5.7 Queue size at the Staging Area emptying the queue as soon as possible

The results of emptying the queue immediately are depicted in Figure 5.7. Figures 5.8 and 5.9 show the queue sizes using emptying strategy (b) and (c) respectively. Figures 5.10 and 5.11 show the overall queue emptying time which represents the time required for all the records inserted in the ISAM file to be stored in the Data Warehouse.

In Figure 5.7 x-axis depicts the time elapsed since the first record reached the Staging Area. Y-axis measures the number of records in the queue of the Staging Area at each time point. We observe that practically no queue is ever formed. The mean queue size is 100 records, which is the rate of the Source Flow Regulator. In other words, the Staging Area is one step later than the Source in terms of performance.

In order to further examine the behavior of our architecture, since the host of the Staging Area is a faster computer, compared to the machine where the Source is hosted, we will simulate slower service rates for the Staging Area. In specific, we will simulate slower service rates by using timeouts between successive services, and by adjusting the number of records served each time.

In Figure 5.8, we can see six diagrams. In all six diagrams we introduce a frequency rate of 1 sec to the service of the Staging Area. X-axis represents the time in seconds. The y-axis stands for the number of records in the queue. The difference lies at the queue emptying rates, starting with an emptying rate resembling 500 records per second at the top left diagram, up to 3,000 records per second at the lower right diagram.

We observe that in all but a small number of occasions (practically the ones where the processing rate is slightly higher than the input rate) the queue size is growing. The mountain size shape is easy to explain: the peak is reached when the 100,000 records have been inserted, no other records are produced and consequently the queue size drops. Figure 5.8 shows the impact of emptying rate clearly: higher emptying rates lock the queue too often and the overall performance drops. Small emptying rates are obviously insufficient, since they empty the queue too slowly.

Figure 5.9 also contains six diagrams. This time, in all six diagrams we introduce a frequency rate of 0.1 sec to the service of the Staging Area. X-axis represents the time in seconds. The y-axis stands for the number of records in the queue. The difference lies at the queue emptying rates, starting with an emptying rate resembling 500 records per second at the top left diagram, up to 3000 records per second at the lower right diagram.

This time we have a quite better picture, being in the middle between immediate emptying and rather slow emptying. As the number of removed records increases each time, the situation starts to approach the behavior of immediate emptying. An interesting lesson here is that it pays off to pay the price of frequent dequeuing rather than remove big chunks of data from the queue. Immediate dequeuing appears to provide the best performance among all alternatives.

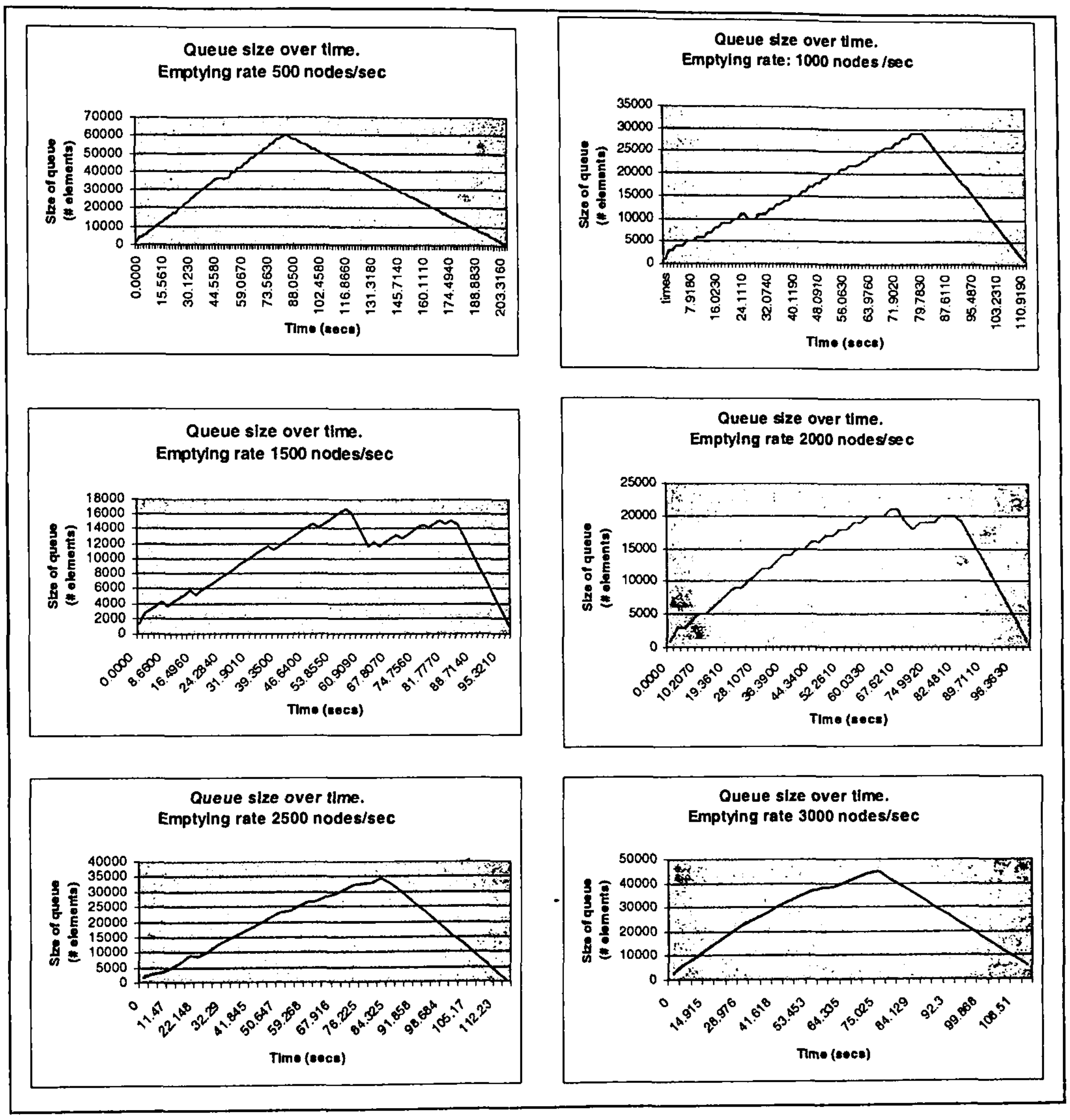


Fig. 5.8. Queue size at the Staging Area using time outs of 1 sec

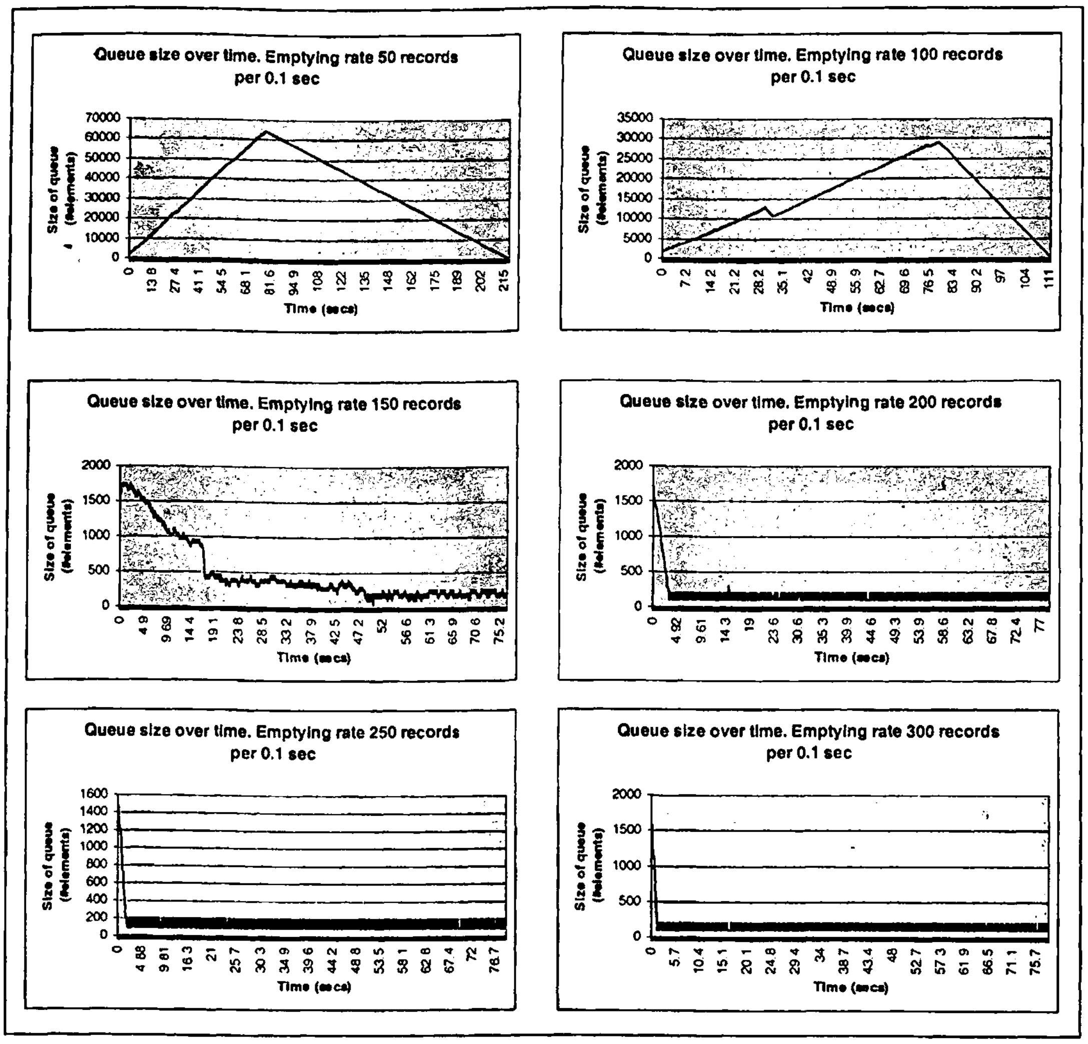


Fig. 5.9. Queue size at the Staging Area using time outs of 0.1 sec

In Figures 5.10 and 5.11 we show the time required to complete the transfer of our entire dataset from the Staging Area to the Warehouse. In both Figures, the x-axis represents the various service rates of the Staging Area and the y-axis the time tin seconds required to complete the transfer. The difference between the two figures is that in Figure 5.10 we employ timeout of 1 between successive services and in Figure 5.11 we use timeouts of 0.1 seconds.

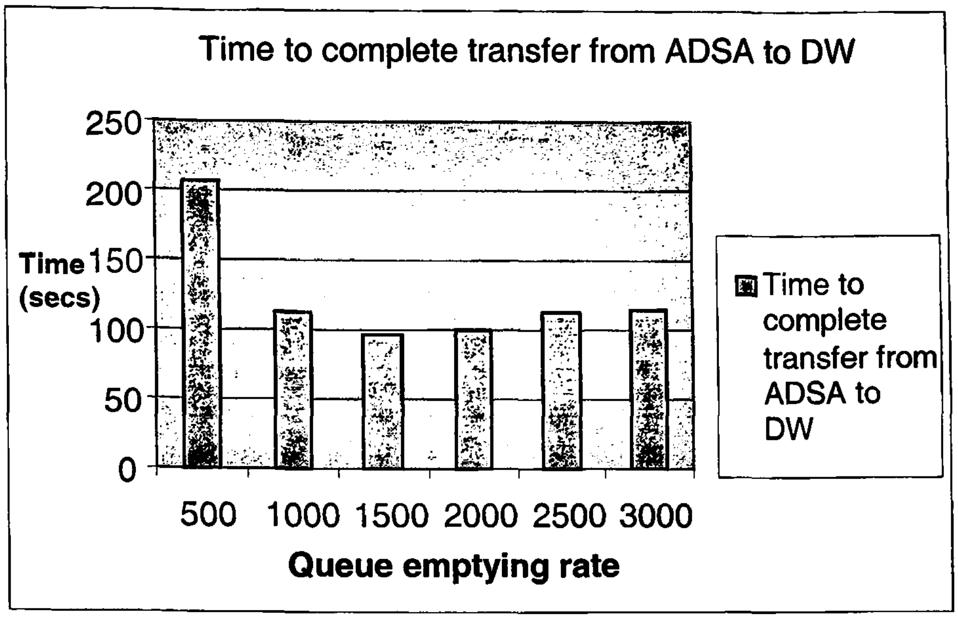


Fig. 5.10. Queue emptying time at the Staging Area using time outs of 1 second.

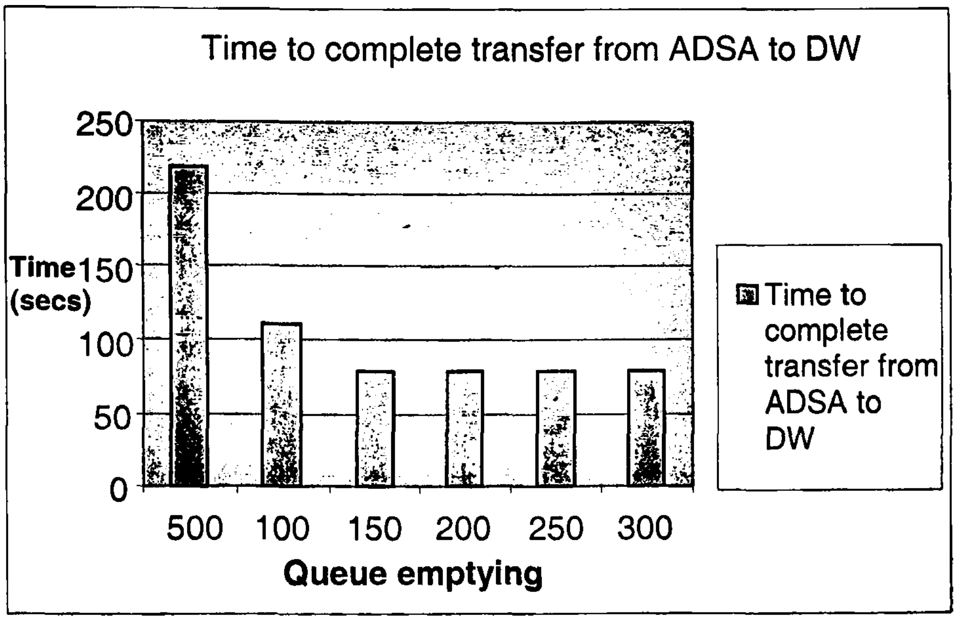


Fig. 5.11. Queue emptying time at the Staging Area using time outs of 0.1 sec

Observing the results of this set of experiments, we are led to the following conclusions:

a) We can achieve data freshness time equal to data insertion time when we continuously empty a small size queue.
b) In this case, the size of queue is equal to the service rate of the Source Flow Regulator, i.e.,,there is practically no delay at the queue.
c) The number of records retrieved from the queue plays a significant role. Even if the actual data rates are the same, (e.g., 150 records per 0.1 second vs. 1500 records per sec), retrieving big chunks of records requires extended locking times and propagation times to the web service.

5.6 Topology and Source Overhead

The aim of this experiment is to examine how the topology of our architecture impacts the Source's performance, i.e., the induced overhead to the Source. We consider the following cases:
a) Using 1-tier architecture.
b) Using 2-tier architecture having the Staging Area placed at the Source's host and the Warehouse on a separate host.
c) Using 2-tier architecture having the Source on a dedicated host and the Staging Area together with the Warehouse on a separate host and
d) having the Source, the Staging Area and the Warehouse on a separate host each.

Each bar in the following figures represents the respective topology, while the y-axis indicates the time required by the Source in Figure 5.12 to insert 100,000 records and in Figure 5.13 to insert 1,000,000 records to the ISAM file. We regard as plain the original source without our alterations.

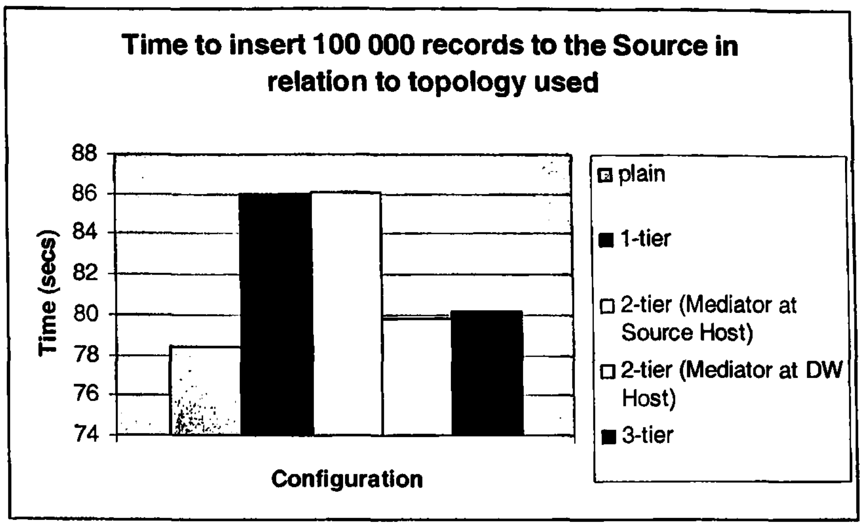


Fig. 5.12 Time to insert 100,000 records using one-tier, two cases of two-tier and a three-tier topology

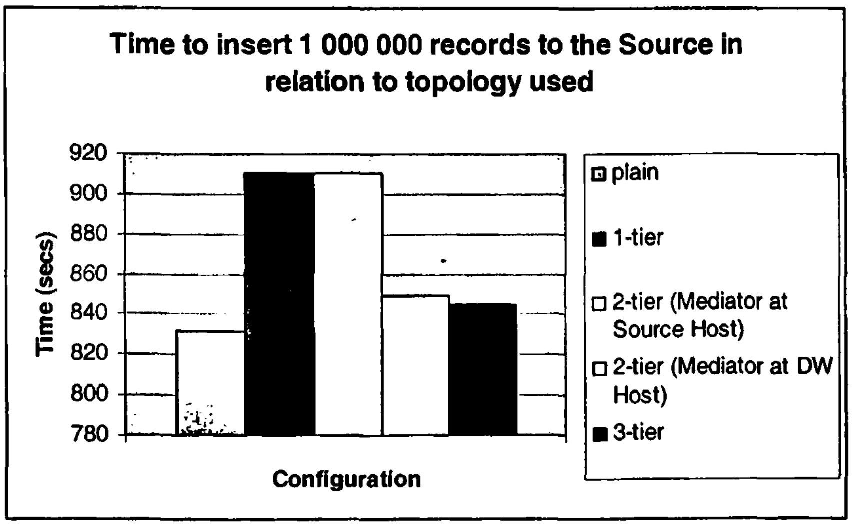


Fig 5.13. Time to insert 1,000,000 records using one-tier, two cases of two-tier and a three-tier topology

Observing these graphs the following conclusions can be made:

(a) The position of the Staging Area is the factor that poses the most overhead to the system's performance. In specific, when the Staging Area is placed at the same host with the Source, no matter where the Web Service is located, the overhead to the system's performance is near 9.5%. On the contrary, when the Staging Area is detached from the Source's host, the overhead fluctuates between 1.5% and 2%.

(b) Using three hosts and placing the Warehouse and the Staging Area separately does not actually reduce the source's overhead, so that the Warehouse can be located on the same host, without this affecting the Source. This observation is very important, since we can save the setup, maintenance and configuration cost of a third computer.

To sum up, the main conclusions derived from the experiments of this chapter are the following:

a) As derived from Section 5.2, TCP is the most suitable protocol for interconnecting the Source with the Staging Area.
b) As shown in Section 5.3, operations of each tier are independent. Regardless of using blocking or non-blocking Web Service invocation at the Staging Area, the Source's throughput is the same in both cases.
c) Using queues both at the Source Flow Regulator and at the Staging Area, as described in Section 5.4, provides the best system performance adding the minimal overhead to the Source.
d) As observed in Section 5.5, the Staging Area service rate should be higher than the Source 's service rate. Moreover, if we want to simulate slower service rates for the Staging Area, small timeouts should be used since they offer stable behavior to the system.
e) Finally, judging from the results of Section 5.6, the most preferable topology for our architecture, is the one of using two tiers and placing the Staging Area at the same host with the Warehouse.

These observations will be used as a guideline for the next chapter. In Chapter 6, we will conduct our experiments under real life data warehousing conditions, where we will assume that the aforementioned parameters are fixed.

6 OPERATIONAL EVALUATION

In this chapter, we will use the architectural guidelines derived from the first set of experiments presented in Chapter 5 to build an active data warehouse where we will also deploy our online ETL operations. The aim of this section is to evaluate the behavior of this fully deployed system and compare its behavior to the theoretical model we have developed.

The difference from the previous chapter is that here we will evaluate our system's performance under real life conditions using as a guideline the results of Chapter 5. To achieve this, we employ four configurations: one configuration with the Staging Area playing a simple intermediate role, where data are just forwarded through a Web Service to the Warehouse and 3 different scenarios with various ETL operations at the Staging Area before delivering our data to the warehouse.

Following the guidelines of Chapter 5 our experimental setup is as follows:

a) We employed a two-tier architecture, placing the Source on one host and the Staging Area with the Warehouse on another host.
b) We used at the Source a Flow Regulator to achieve better performance.
c) The interconnection protocol between the Source and the Staging Area is TCP.
d) We send data from the Staging Area to the Warehouse using non-blocking invocation.

In our configuration, the Source includes two software modules: (a) an ISAM file and (b) an application used to modify data in the legacy data source. In order to manipulate ISAM files, there is a library of ISAM routines that are invoked from the application at the source side. We have modified these library routines to replicate the data manipulation commands and send updates towards the Staging Area. The ISAM library that we altered is the

PBL/ISAM suite [PBL04] available under GPL license. We have used a sample program distributed within the suite as the legacy application. We use two different data sets for our purposes. The first consists of 100,000 records and the second of 1,000,000 records. The ETL queues of the Staging Area have been implemented using the Sun JDK 1.4, whose runtime engine has also been used. As a Web Services platform we have used Apache Axis 1.1 [AXIS04] with Xerces XML parser running over Apache Tomcat 1.3.29. Our Data Warehouse is implemented as a MySQL 4.1 database.

The host we used for the Source was a PIII 700MHz with 256MB of physical memory running SuSE Linux 8.1. The host used as the data warehouse is a Pentium 4 2.8GHz with 1GB of physical memory running Mandrake Linux. This server also hosted the Staging Area. The hosts are interconnected via the switched Fast Ethernet LAN of our department.

Our data were created from the TPC-H data generation tool. We used data of variable size. In this case each row has an average size of 140 bytes.

The roadmap of this chapter is as follows: in Section 6.1 we establish the fact of minimal impact at the Source. In Section 6.2, we measure the throughput of each ETL operation. In Section 6.3, we measure data freshness for four different scenarios, while in Section 6.4, we compare the behavior of our system against our prediction.

6.1 Overhead at Source

The aim of this experiment is to assure that the overhead at the source remains small even though the size of each row is almost 10 times bigger than the previous case.

- First, we measure the effect of using the Source Flow Regulator (Source Flow Regulator) at the Source. We try four values: 1, 10, 25, 50 and 75 records for each packet that the Source Flow Regulator sends to the Staging Area (see the x-axis).

When using one record at a package, we have in fact the case of not using a Source Flow Regulator.

- Second, we experiment with the behavior of the source in terms of transmission rate. In the first case that we call "plain", the source performs its regular operation during normal time. In this case no records are propagated to the Staging Area.

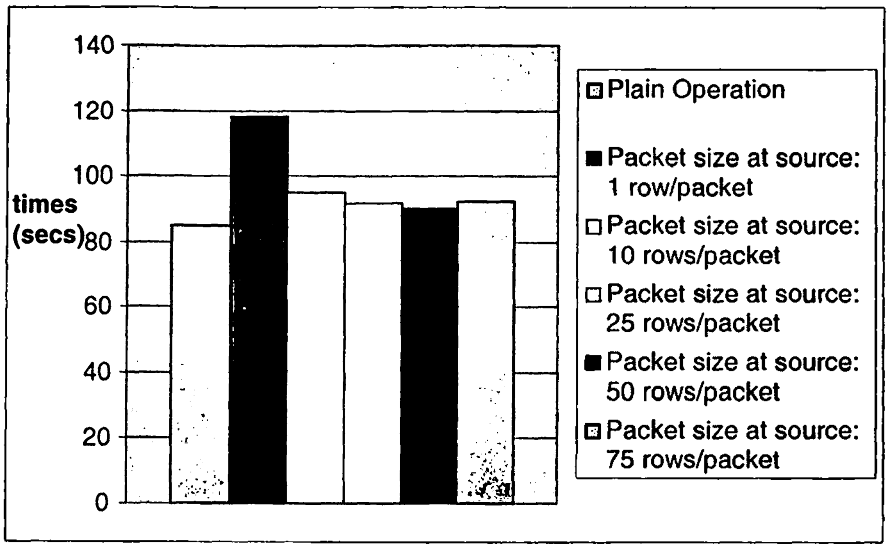


Fig. 6.1. Time to insert 100 000 records in the warehouse.

In Figure 6.1, the x-axis represents the size of the packet sent from the Source Flow regulator to the Staging Area, apart from the first column, where the operation of the Source without our additions is measured. The y-axis of Figure 6.1 represents time. In general, packet sizes of over 25 records offer the least burden to the source. The smallest delay was achieved with a packet size equal to 50, where the source delay was measured to be at 5.8%.

6.2 Throughput Capability of ETL Operations

The goal of this set of experiments is to determine the average throughput of the ETL operations we have implemented. The outcome of this part of our evaluation will be used both to interpret the results of our experiments and to use the measurements for our theoretical analysis.

Our experimental setup was the following for the evaluation of all the operations: The source was hosted separately from the Staging Area and the Warehouse which were hosted together. The Source Flow Regulator size was 50 rows per packet. At the Staging Area only the evaluated ETL operation was executing each time. The measured ETL operations are the following:

a) Filtering of 2% of incoming packets to the Staging Area.
b) Filtering of 6% of incoming packets to the Staging Area.
c) Filtering of 10% of incoming packets to the Staging Area.
d) Aggregate Group by sum.
e) Surrogate Key Transformation.
f) Replacement Transformation

The results were sent and stored to the Warehouse. The emptying (service) rate of each queue was fixed to 1500 rows per second, which is a rate slightly higher than the arrival rate from the Source. In Figure 6.2 our experimental results are displayed. In specific, the x-axis represents the number of packets each ETL operation can process. The y-axis represents each ETL operation.

Concerning the filter transformation, the percentage displayed refers to the percentage of rows rejected from the total number of rows injected into the Staging Area. In this case, since we use a dataset consisting of 100 000 rows, a filter equal to 10% will reject 10 000 rows, meaning that 90 000 rows will be stored in the warehouse.

The throughput capability of each ETL operation, showing the maximum service rate each operation can achieve, is depicted in Figure 6.2.

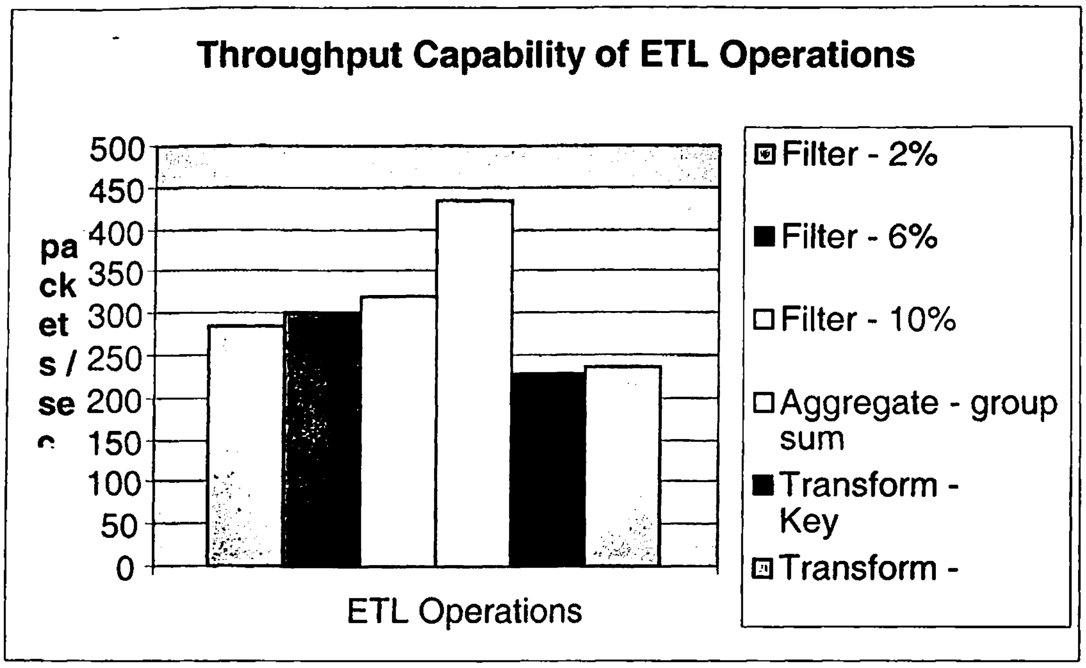


Fig. 6.2. Throughput Capability of ETL Operations.

Based on Figure 6.2 the following conclusions can be made:

(a) Higher number of rejected rows leads to higher throughput.

(b) The aggregate group by sum is surprisingly the operation with the highest throughput. This is because all operations occur in memory and in contrast with the other operations, smaller size of data is produced as the output.

(c) The Replace and Surrogate Key operations have almost the same throughput, which is significantly lower than that of all other operations. This is mainly because both of these operations seek and replace values having to parse the entire row. On the contrary, Filter operations simply check a field.

6.3 Data Freshness

Having established the requirement of minimal source impact, our focus moves towards the data freshness issue. We want to achieve high freshness of data delivered from the Source to the Warehouse through the Staging Area. The goal of the next set of experiments is to measure the data freshness time provided by our application with respect to the Staging Area service rates. We consider the following scenarios:

(b) For scenario (a) as illustrated in Figure 6.3:

a. Simply transferring data inserted into the legacy application to the warehouse.

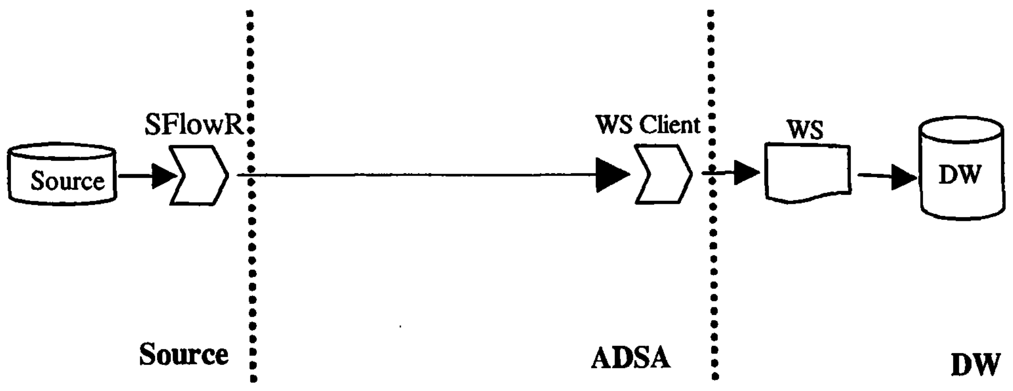


Fig. 6.3. Illustration of Scenario (a)

(c) For scenario (b):

a. Filter 10% of incoming data.

b. A surrogate key operation to the first column of the filtered data.

c. Group by sum.

d. Data are then fed to the warehouse.

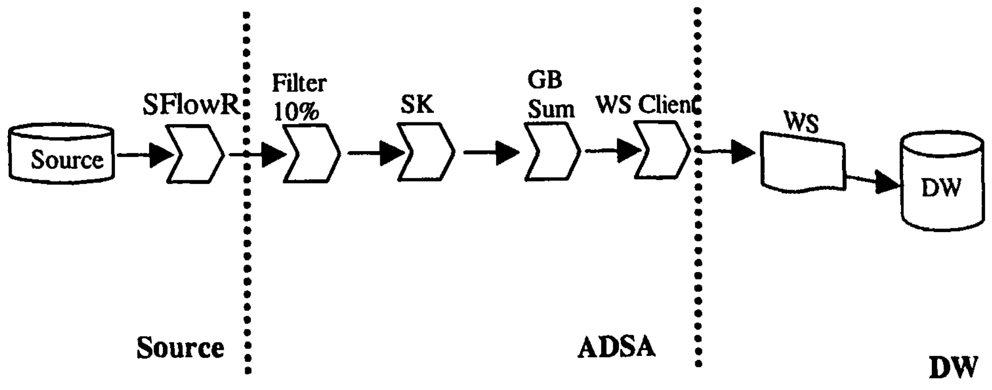


Fig. 6.4. Illustration of Scenario (b)

(d) For scenario (c):

  a. Filter 10% of incoming data.
  b. Additionally Filter 2% of the remaining data.
  c. A surrogate key operation to the first column of the data. Then the stream is replicated along two branches:
  d. For the first branch:
    i. A group by sum operation is performed
    ii. Data are fed to the warehouse.
  e. For the second branch:
    i. Data are fed to the warehouse.

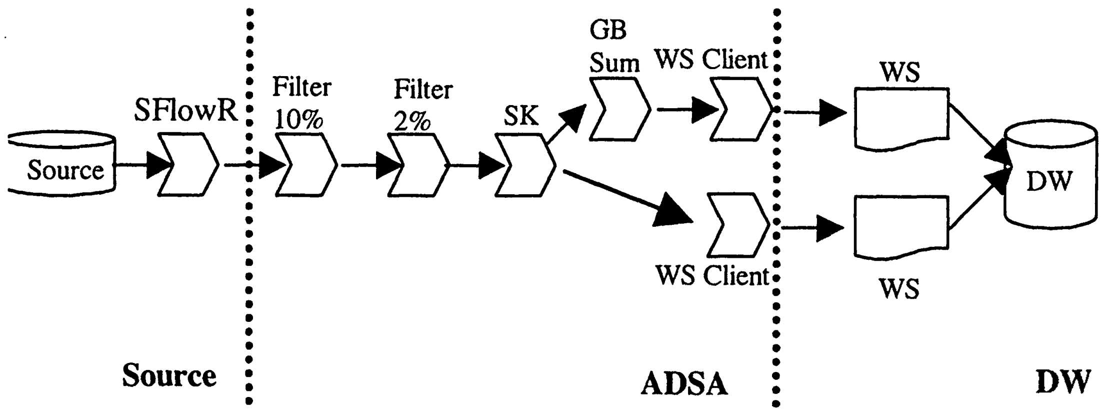


Fig. 6.5. Illustration of Scenario (c)

(e) For scenario (d):

  a. Filter 10% of incoming data.
  b. Replacement of the values of the first field.
  c. A surrogate key operation to the first column of the data. Then the stream is replicated along two branches:
  d. For the first branch:
    i. A group by sum operation is performed.
    ii. A Filter rejecting 6% of input data

   iii. Data are fed to the warehouse.

e. For the second branch:

   i. A replacement of the values of the first field is performed.

   ii. A Filter rejecting 2% of input data is applied.

   iii. Data are fed to the warehouse.

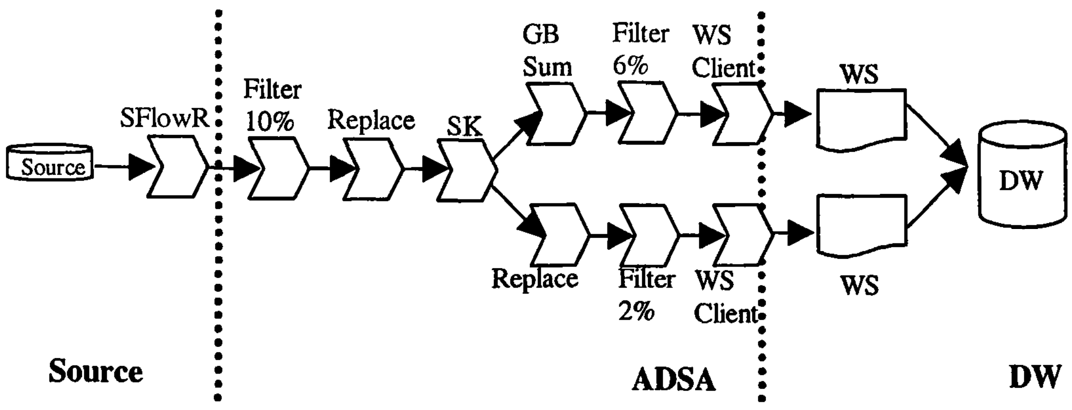


Fig. 6.6. Illustration of Scenario (d)

At this point, it has to be clarified that when the data flow splits, data are fed separately in the two different data flows and separately sent to the warehouse.

Specifically, we trace the queue size as time passes. We do this for each operation separately and for the entire system as a whole. At the same time, at the end of the experiment we have a measure of the overall throughput, i.e., the time needed to empty the Staging Area 's queue after the first record is sent to the warehouse. The freshness is then measured as the time needed to empty the queue, which practically stands for the response time for the last record.

To perform these measurements, we have the following setting: the Legacy Application sends 100,000 records to the Staging Area in packs of 50 records over TCP at a rate of 22 packets per second. Also, we measure the queue length as an indicator of resource consumption at the Staging Area site. We count the queue length each time the queue is not empty and before we retrieve the rows from the queue.

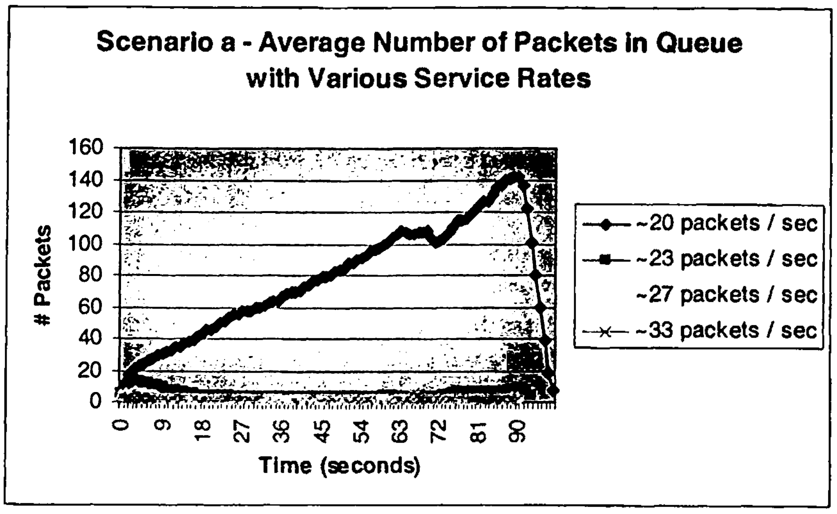


Fig. 6.7. Cumulative results of queue sizes for scenario (a) with service rate close to 23 packets per second.

The results of scenario (a) for various service rates of the Staging Area near the arrival rates from the Source are shown in Figure 6.7. For scenarios (b), (c) and (d), we will examine the behavior of the Staging Area at a simulated service rate of 23 packets per second, in order to study its behavior at marginal conditions. We remind that the arrival rate of the warehouse is at 22 packets per second. Figure 6.8 depicts in a cumulative fashion the average queue lengths for scenario (b). Similarly, the same values for scenario (c) are illustrated in Figure 6.9, and for scenario (d) in 6.10. For all figures, x-axis represents time, while the y-axis stands for the number of packets in each queue before service.

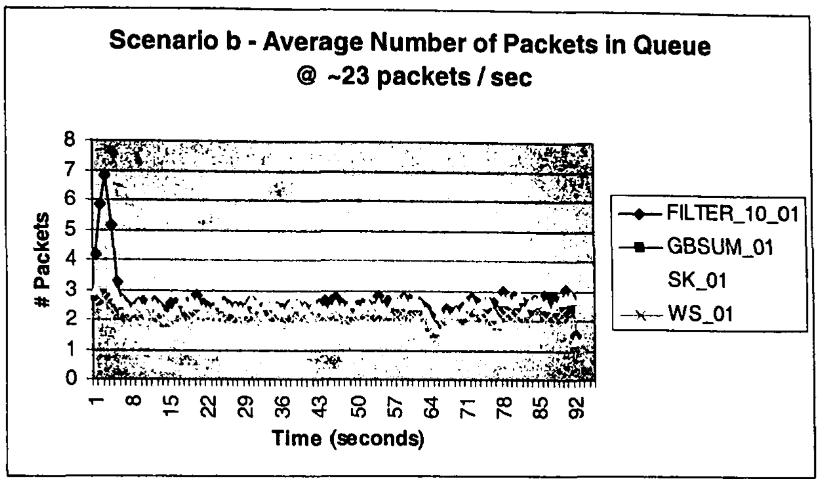


Fig. 6.8. Cumulative results of queue sizes for scenario (b) with service rate close to 23 packets per second.

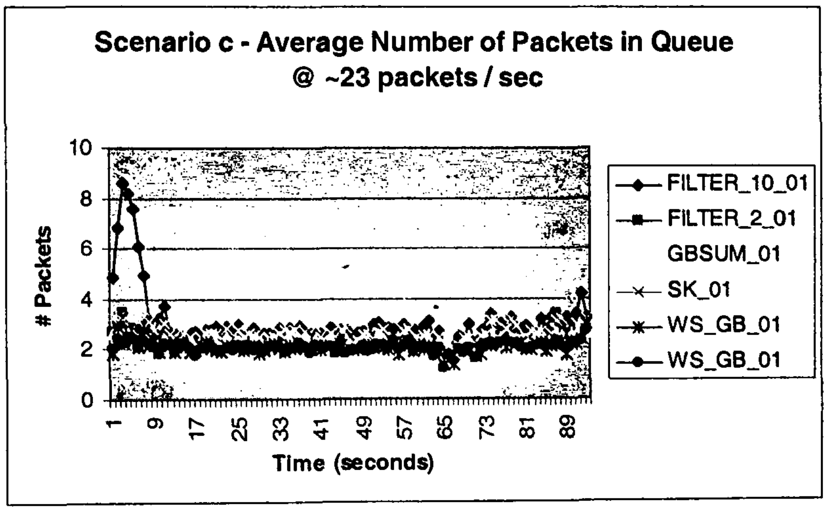


Fig. 6.9. Cumulative results of queue sizes for scenario (c) with service rate close to 23 packets per second.

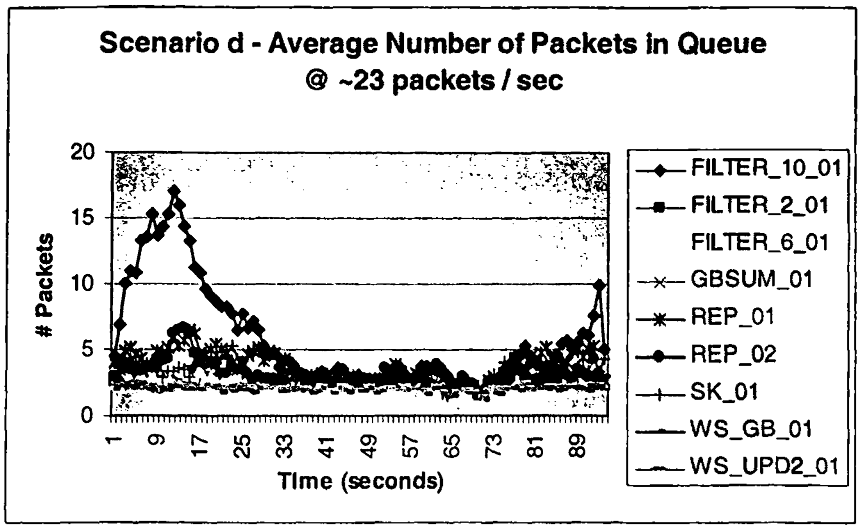


Fig. 6.10. Cumulative results of queue sizes for scenario (d) with service rate close to 23 packets per second.

Finally, Figure 6.11 summarizes the total times needed for the Staging Area to transfer all data to the warehouse, for each scenario of ETL queues. X-axis of Figure 6.8 stands for each scenario examined. Y-axis represents the time needed to complete each scenario.

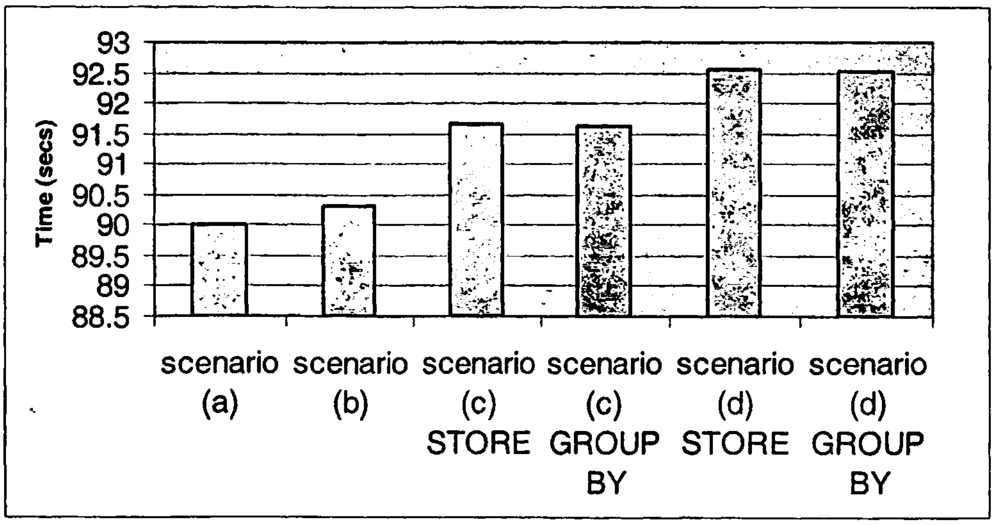


Fig. 6.11. Data freshness for each scenario

Observing the figures, we derive the following conclusions:

1. The source capability is approximately 1100 rows/sec. Since we are using packets of 50 rows, this amount is translated into approximately 22 packets per second. In scenario (a) we are led to queue explosion, when we employ service rate smaller than the Source's arrival rate. Using a service rate of 23 packets / sec, which is a setting close to the arrival rate, we can see that transient effects tend to appear, but the queue converges to steady state. This occurs because the service rate is very close to the arrival rate, thus needing some time to reach a steady state, where the service rate exceeds arrival rate. By using higher service rates, 27 and 33 packets / sec respectively, the queue maintains its steady state.
2. In scenarios (b), (c) and (d) we observe that the entire system, as well as the queue of each operation, maintains a steady state. The number of packets in the queue is less or equal to the maximum number of packets polled simultaneously from the queue. This practically means that after each poll the queue empties and that the Staging Area is only one step behind the Source.
3. In Figure 6.11, the total time needed for the entire dataset to be transferred from the Staging Area to the Warehouse is dependent on the number of the intermediate ETL operations. As the number of intermediate ETL operations that a packet has to visit increases, the total delay increases as well. Nevertheless, in our exemplary scenarios, the increase is rather small, due to the pipelining of data. The average delay per row is around 0.9 milliseconds for all scenarios.

6.4. Theoretical vs. Experimental Evaluation

In Figures 6.9 – 6.30 we present the comparison of our theoretical evaluation of queue length against the observed values. We show the results of scenarios (b), (c) and (d). In all experiments, we simulate various service rates at the Staging Area by using predefined waiting times between successive services. In specific, we employ timeouts equal to 100, 80, 60, 20 and 1 millisecond each time respectively.

In this set of experiments, we will not use as a metric the service rate but the timeout value between successive services. We prefer this approach due to the differences of the service times for each ETL operation (as discussed in Section 6.2). Because of these differences and due to the exponential nature of arrival and service times we cannot fix standard service times. Moreover, the number of ETL operations, i.e., independent threads of the Staging Area for each scenario varies from four in scenario (b), to nine in scenario (d). This way we are led to different CPU scheduling properties, an issue which is beyond our scope. Thus, in order to have a unified view of our experiments, we will use the timeout value as an indirect reference to the service rates.

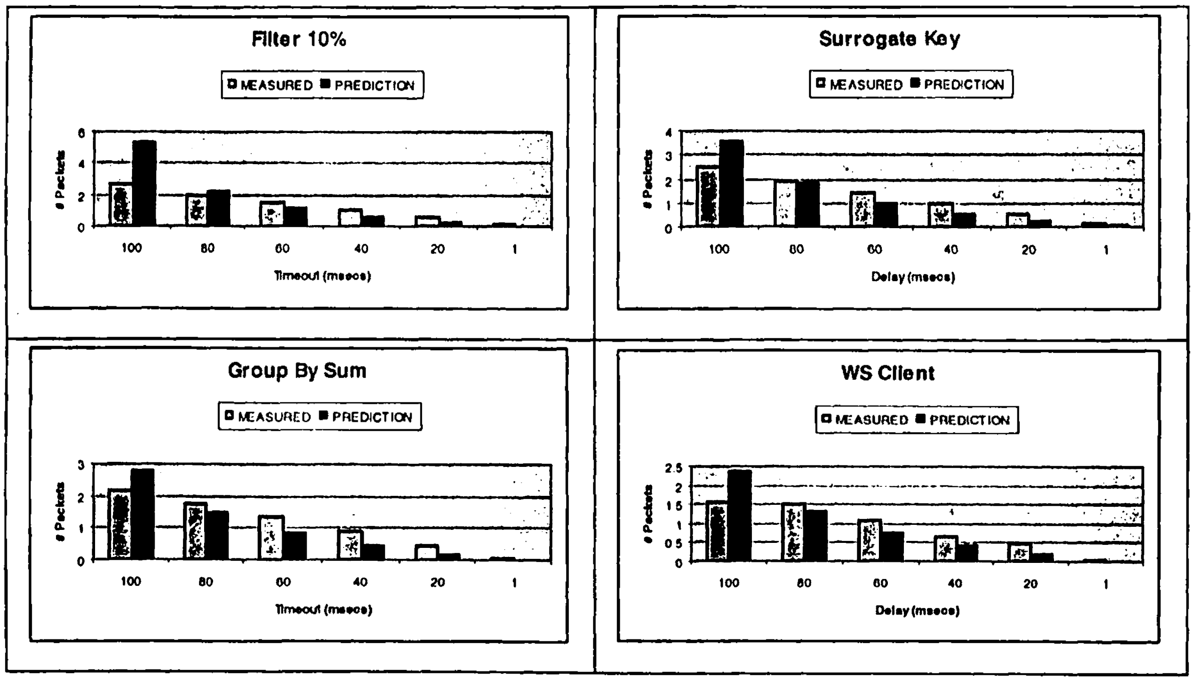


Fig. 6.12-6.15. Queue size vs. prediction for each operation of scenario b

In all Figures 6.12 – 6.30, the x-axis stands for the waiting time in milliseconds between consecutive services. The y-axis represents the number of packets in the queue of each operation. The first column represents the measured value, while the second row represents the predicted value.

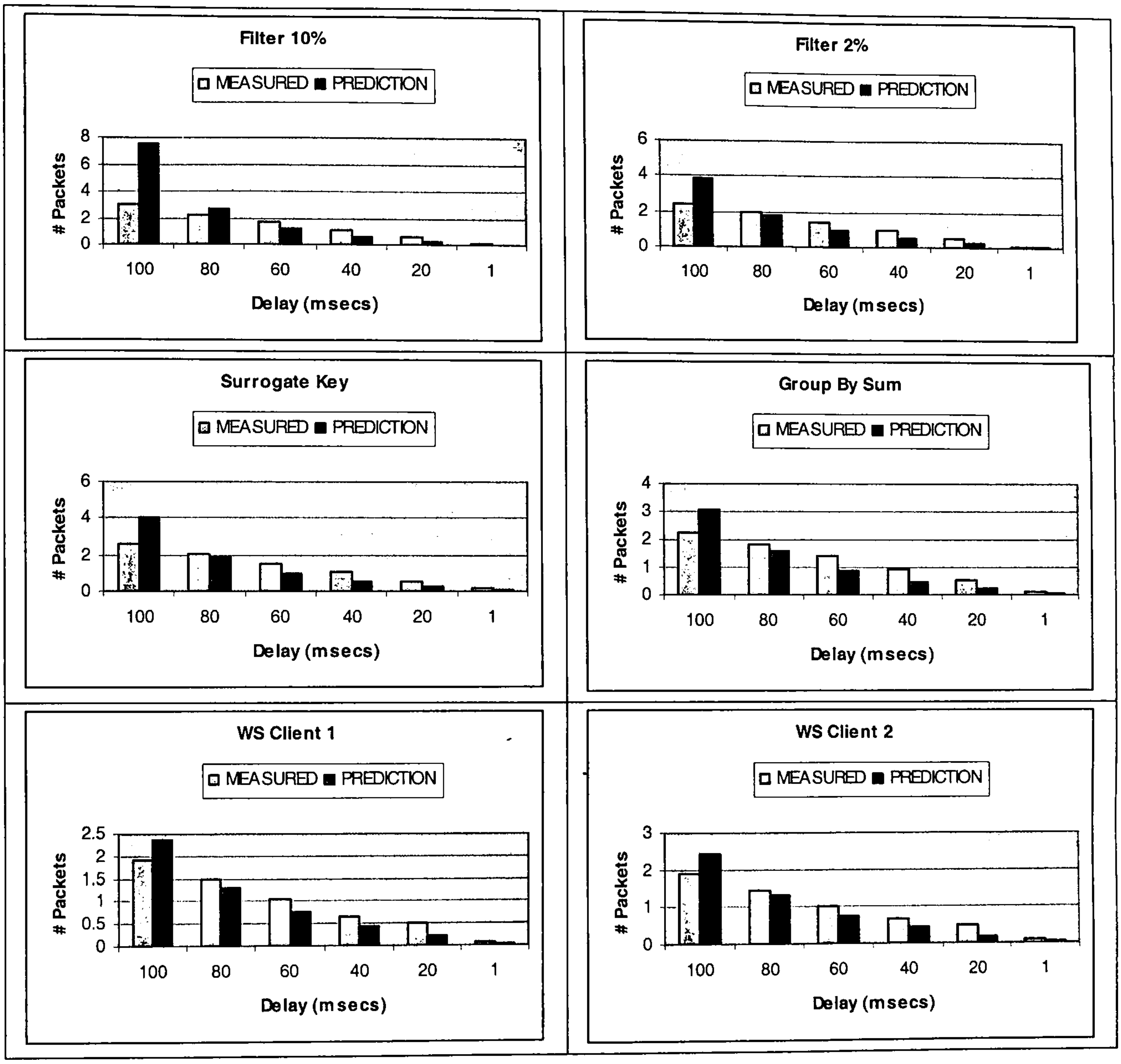


Fig. 6.16-6.21. Queue size vs. prediction for each operation of scenario c

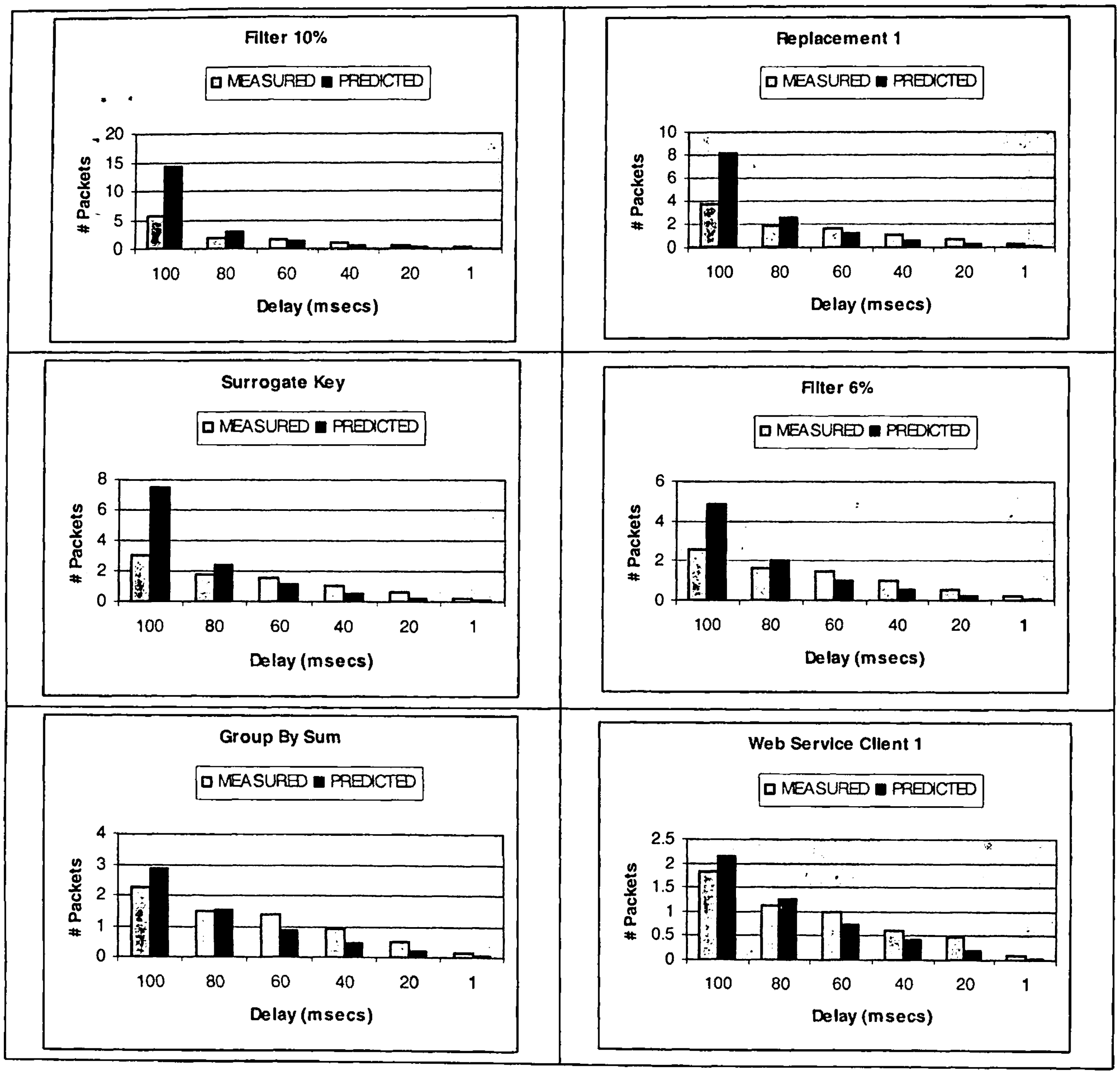
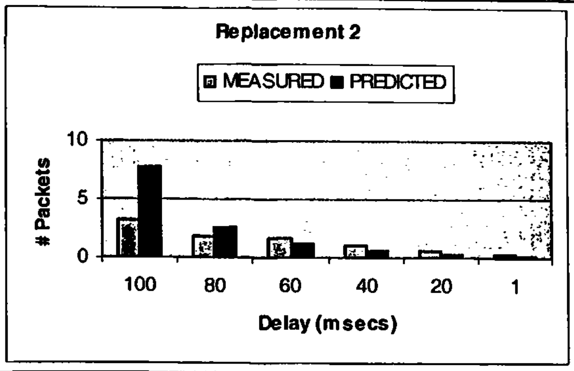
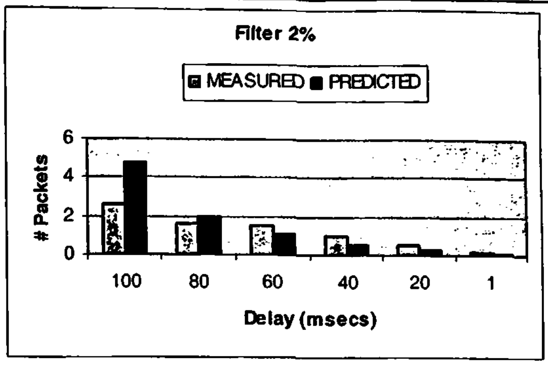


Fig. 6.22-6.27. Queue size vs. prediction for each operation of scenario d





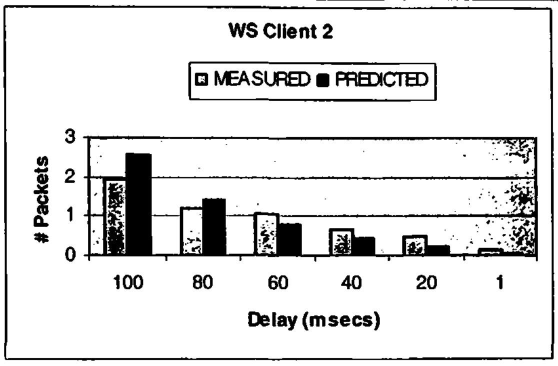


Fig. 6.28-6.30. Queue size vs. prediction for each operation of scenario d (continued)

In these measurements one can easily see that in all cases our prediction underestimates the size of the queue by half a packet in each case. For example in scenario (b) estimations using a delay of 80 milliseconds between successive services leads to underestimation only of half a packet. The same holds for a delay of 60 seconds in scenarios (c) and (d). This observation is very important for our architecture, since our prediction misses only half a packet. Moreover this error remains the same for all operations.

The only discrepancies of this rule are in two cases. The first case is when we utilize high delay times i.e. 100 milliseconds between successive services, meaning at the same time lower service rates. The second case is in scenario d, where we impose a delay time equal to 80 milliseconds. This case is similar to the previous; because if we consider the additional context switch time for the high number of ETL operation, we are led to the conclusion that again is a case of low service rate.

However, in these cases we have overestimates of the queue sizes. This leads as to the conclusion that high server utilization rates result in overestimation of the queue sizes, while low server utilization rates leads to underestimation of the queue size. This underestimation is fixed and no more than half a packet.

These differences between the measured and estimated values for the queue sizes occur due to the following reasons:

a) We simulate lower service rates at the Staging Area by invoking timeouts between successive services. Queue theory is not designed taking care of such issues, so differences between the measured and expected value are expected.
b) In our simulation of slower staging areas, we serve up to three packets simultaneously. Queue Theory end especially Queue Network Theory, were designed assuming service of single packets. Hence, in these cases, differences between the measured and expected value are expected as well.

	MEASURED	PREDICTION	DIFFERENCE (PACKETS)
FILTER_10_01	0.160	0.056	0.104
FILTER_2_01	0.134	0.047	0.087
SK_01	0.154	0.054	0.100
GBSUM_01	0.137	0.048	0.089
WS_GB_01	0.091	0.031	0.059
WS_UPD2_01	0.100	0.035	0.066

Table 6.1. Queue size vs. prediction for each operation of scenario (c) operating at its full capability.

On the contrary, when we do not use sleep times between successive services leading to the maximum service rates, the difference in terms of queue size is significantly small. This is because we both do not use a timeout and due to the high service rates, each time a single packet is served. Thus, our implementation is closer to the theoretical models of Queue Networks and the difference between the measured and predicted values are very small.

In Table 6.1 we present as a reference the comparison of our theoretical evaluation of queue length against the observed values for scenario (c) with the Staging Area operating at its full capability. It is easy for someone to see that the difference in queue lengths between the theoretical prediction and the measured evaluation is small for all operations.

7 CONCLUSIONS AND FUTURE WORK

Active Data Warehousing refers to a new trend where data warehouses are updated as frequently as possible, due to the high demands of users for fresh data. In this thesis, we have proposed a framework for the implementation of active data warehousing, keeping in mind the following goals: (a) minimal changes in the software configuration of the source, (b) minimal overhead for the source due to the "active" nature of data propagation, (c) the possibility of smoothly regulating the overall configuration of the environment in a principled way. In our framework, we have implemented ETL activities over queue networks and employed queue theory for the prediction of the performance and the tuning of the operation of the overall refreshment process. In terms of data freshness, source overhead and minimal impact of software configuration the results seem satisfactory. A summary of the lessons learned is as follows:

- Queue theory can be successfully employed as the theoretical background for the estimation of the response of the active warehouse. The system reaches a steady state quite close to the predicted behavior. Freshness is quite satisfactory too.
- TCP should and can be used instead of UDP, due to the packet loss of the latter. Organization of tuples in blocks, both at the source and the Staging Area side increases performance.
- The overall overhead at the source side remains small despite the size of data transferred to the Staging Area, and the amount of code modification is around 100 lines, without affecting applications.
- The Source Flow Regulator plays a very important role, since its utilization increases performance.
- The way that the Staging Area is tuned does not affect the source. Regardless of using blocking or non-blocking Web Service invocation at the Staging Area, the Source's throughput is the same in both cases. This is a key observation for our architecture,

since it shows that the operations of each tier are independent. Thus, we can examine each tier separately.

- Using both a Source Flow Regulator at the Source and a queue at the Staging Area provides the best system performance adding the small overhead to the Source.
- The position of the Staging Area is the factor that poses the most overhead to the system's performance. The best layout is to use a 2-tier architecture placing the Staging Area together with the Warehouse but separated from the Source.
- In the case of employing ETL transformations at the Staging Area, high server utilization rates result in overestimation of the queue sizes, while low server utilization rates leads to underestimation of the queue size. This underestimation is fixed and no more than half a packet.

Future work includes several directions. A key direction of research would have to do with the failure management of the components of the environment, in order to determine safeguarding techniques and fast resumption algorithms for the event of a failure. Further tuning can be made, by testing multiple concurrent loading sources for the warehouse. In this case, an interesting issue is to determine the required number of flow regulators, together with the number of separate required Staging Areas. Also, the case of materialized aggregate views and schema evolution (as mentioned in the Related Work section) poses interesting challenges in this context. Finally, further experimentation can be made over the interconnection of the Source and the Staging Area by employing a UDP with built-in flow control.

REFERENCES

[Aba+03] Daniel J. Abadi, Don Carney, Ugur Çetintemel, et al. Aurora: a new model and architecture for data stream management. The VLDB Journal, 12(2):120-139, 2003.

[ACCC+03] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, Stanley B. Zdonik: Aurora: a new model and architecture for data stream management. VLDB J.12(2): 120-139 (2003)

[ACKM03] G. Alonso, F. Casati, H. Kuno, V. Machiraju. Web Services: Concepts, Architectures and Applications. Springer-Verlag, 2003.

[AdFi03] J. Adzic, V. Fiore. Data Warehouse Population Platform. In Proc. 5th Intl. Workshop on the Design and Management of Data Warehouses (DMDW'03), Berlin, Germany, 2003.

[AdRe01] Ivo Adan and Jacques Resing, Department of Mathematics and Computing Science, Eindhoven University of Technology, 2001. Queueing Theory notes available at http://www.cs.duke.edu/~fishhai/misc/queue.pdf

[AXIS04] Apache Software Foundation. Axis. Available at http://ws.apache.org/axis/

[BaWi01] S. Babu, J. Widom. Continuous Queries over Data Streams. SIGMOD Record 30(3): 109-120, 2001.

[BBDM+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, Jennifer Widom: Models and Issues in Data Stream Systems. PODS 2002: 1-16

[ChCR02] Jun Chen, Songting Chen, Elke A. Rundensteiner: A Transactional Model for Data Warehouse Maintenance. ER 2002: 247-262

[Duqu03] W. Duquaine Web Services Ruminations. Presentation at High Performance Transaction Systems Workshop (HPTS'03). Asilomar Conference Center, California, October 12-15, 2003. Available at http://research.sun.com/hpts2003/

[GFSS00] H. Galhardas, D. Florescu, D. Shasha and E. Simon. Ajax: An Extensible Data Cleaning Tool. In *Proc. ACM SIGMOD International Conference on the Management of Data*, pp. 590, Dallas, Texas, (2000).

[GrHa85] D.Gross, C.Harris, Fundamentals of Queuing Theory, Wiley series in probability and statistics. (1985)

[GuMu95] Ashish Gupta, Inderpal Singh Mumick: Maintenance of Materialized Views: Problems, Techniques, and Applications. Data Engineering Bulletin 18(2): 3-18 (1995)

[Inmo02] Inmon W. H., Building the data warehouse. John Wiley & Sons, Inc. 2002

[JiCh03] Qingchun Jiang, Sharma Chakravarthy: Queueing analysis of relational operators for continuous data streams. CIKM 2003: 271-278

[LoGe03] D. Lomet, J. Gehrke. Special Issue on Data Stream Processing. Bulletin of the Technical Committee on Data Engineering, 26(1), 2003.

[Magl04] V. Maglaris, online lecture notes on queue theory, available at http://www.netmode.ntua.gr/courses/undergraduate/queues

[PBL04] P. Graf. The Program Base Library. Publicly available through http://mission.base.com/peter/source/

[RaGe02] Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke. McGraw-Hill Science 2nd Edition. Greek Translation – Volume 2, 2002

[RaHe01] V. Raman, J. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB)*, pp. 381-390, Roma, Italy, (2001).

[SOAP03] W3C. SOAP Version 1.2. June 2003. W3C Recommendation. http://www.w3.org/TR/soap12-part0/

[VaSS02] Panos Vassiliadis, Alkis Simitsis, Spiros Skiadopoulos: Modeling ETL activities as graphs. DMDW 2002: 52-61

[VSGT02] Panos Vassiliadis, Alkis Simitsis, Panos Georgantas, Manolis Terrovitis: A Framework for the Design of ETL Scenarios. CAiSE 2003: 520-535

[Whit02] C. White. Intelligent Business Strategies: Real-Time Data Warehousing Heats Up. DM review, August 2002. Available at http://www.dmreview.com/article_sub.cfm?articleId=5570

[Will04] Andreas Willig, Lecture notes on Performance Evaluation. Available at http://www-ks.hpi.uni-potsdam.de/docs/engl/teaching/pet/ss2004/skript.pdf

[WSDL03] W3C. Web Services Description Language (WSDL) Version 2.0. W3C Working Draft. November 2003. http://www.w3.org/TR/wsdl20/

[WSFL01] Frank Leymann. Web Services Flow Language (WSFL 1.0), May 2001. Available at http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[ZGHW95] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, Jennifer Widom: View Maintenance in a Warehousing Environment. SIGMOD Conference 1995: 316-327

[ZhRu02] Xin Zhang, Elke A. Rundensteiner: Integrating the maintenance and synchronization of data warehouses using a cooperative framework. Inf. Syst. 27(4): 219-243 (2002)