

MASTER'S THESIS

Efficient Methods to Compute, Store, and Manipulate Aggregates Derived from Network Management Data Using a Database Tool

by Albino S. Pinho

Advisor: John S. Baras

CSHCN M.S. 99-2
(ISR M.S. 99-2)



The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site <http://www.isr.umd.edu/CSHCN/>

ABSTRACT

Title of Thesis: EFFICIENT METHODS TO COMPUTE, STORE, AND
 MANIPULATE AGGREGATES DERIVED FROM
 NETWORK MANAGEMENT DATA USING A
 DATABASE TOOL

Degree candidate: Albino S. Pinho

Degree and year: Master of Science, 1999

Thesis directed by: Professor John S. Baras
 Department of Electrical Engineering

As telecommunications in the world grow exponentially in importance for businesses, the sizes of networks grow, generating increasingly monitored statistics. To handle this data, efficient systems are required to store and manipulate it. This thesis investigates efficient methods to compute, store, and operate the current and historical aggregates, derived from network management data. The networks of interest are those with star architecture.

This thesis proposes a methodology to process the incoming data from the network elements, store it in a database, and then use efficient techniques to perform the aggregation through the various dimensions, namely by attributes, network containment, and time. This procedure generates data with different levels of granularity, which is suitable for the use of modern on-line analysis

processing techniques, e.g., *drill-downs* and *scale-ups*, and for gradual bottom-up data trimming without losing information, although coarsening it.

This thesis demonstrates that modern databases may be tuned and deployed for the purposes of computing and storing network aggregates.

EFFICIENT METHODS TO COMPUTE, STORE, AND MANIPULATE
AGGREGATES DERIVED FROM NETWORK MANAGEMENT DATA USING A
DATABASE TOOL

By

Albino S. Pinho

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Master of Science
1999

Advisory committee:

Professor John S. Baras, Chairman/Advisor
Professor Nick Roussopoulos
Professor Michael O. Ball

©Copyright by

Albino S. Pinho

1999

ACKNOWLEDGEMENTS

While this thesis is a requirement for graduation of a single student, it is, however, the result of the work of a team of people, who have made me feel very lucky and proud for having the opportunity to work with them on such an interesting topic. Therefore, I would like to express my sincere gratitude to Dr. John S. Baras for giving me the opportunity to work with him on this project, for his guidance and leadership, and for providing the needed resources making this discourse possible. I would also like to thank David Whitefield of the Hughes Network Systems, Inc., for his help in understanding the problem and for helping on the solution design, refinement, and implementation. Also my thanks to Dr. George Mykoniatis for the many professional discussions on the problem, potential alternative solutions and results. Dr. Nick Roussopoulos for his analysis of the problem, suggestions, and comments, and for joining my defense committee and Dr. Michael Ball for accepting a position on my defense committee. Further thanks goes to Spyro Papademetriou and Steve Kelley for helping to establish the environment needed to make this research possible. Hughes Network Systems, Inc., Germantown, MD, supported the research.

I am proud to be an Institute of Systems Research member and a University of Maryland student, and this was possible only because of Mattie Riley's invitation. Dr. Barbara Goldberg, Dr. Stan Hunt, Dr. Mark Austin, thank you all for your support at the difficult moments of my life as an ISR-UMCP student. Thanks to America and the American people for developing such a good educational

system, that contributes so much to the progress of science and enlargement of the frontiers of human knowledge everyday.

Furthermore, I want to extend a special thanks to my wife, Ana, for understanding that to have a better future and a better lifestyle, the holidays of 1998 had to be sacrificed.

Finally, I would like to thank my family and friends, because without their love and friendship, I would never have had the needed motivation to study and work hard enough to succeed in this and other life challenges.

TABLE OF CONTENTS

ABSTRACT	II
ACKNOWLEDGEMENTS	VI
TABLE OF CONTENTS	VIII
LIST OF FIGURES	X
LIST OF TABLES.....	XIII
CHAPTER 1.....	1
INTRODUCTION.....	1
1.1 <i>The Systems Engineering Process</i>	1
1.2 <i>Research Goals and Proposed Aging Schema</i>	3
1.3 <i>Aggregates Dimensions and the Aggregation System</i>	4
1.4 <i>Organization of Thesis Contents</i>	7
CHAPTER 2.....	9
NETWORK MANAGEMENT DATA COLLECTION AND PROCESSING	9
2.1 <i>The Network Model</i>	9
2.2 <i>The Current System</i>	11
CHAPTER 3.....	16
THE PROBLEM ANALYSIS AND SYSTEM REQUIREMENTS.....	16
3.1 <i>System Requirements</i>	17
3.2 <i>Effectiveness Measures</i>	18
3.3 <i>Expected Outputs</i>	19
3.4 <i>Sequential Build and Test Plan</i>	19
CHAPTER 4.....	22
ARCHITECTURE OF THE PROPOSED SOLUTION.....	22
4.1 <i>System Physical Model</i>	26
4.2 <i>Used Networks Configuration</i>	29
CHAPTER 5.....	30
THE LOADING PROCESS	30
5.1 <i>Loading Process Parameters</i>	30
5.2 <i>Loading Process Sections</i>	31

CHAPTER 6.....	34
THE AGGREGATION PROCESSES	34
6.1 Views and Views Materialization	35
6.2 Aggregation Process Sections	37
CHAPTER 7.....	41
DEVELOPMENT AND TESTING ENVIRONMENT	41
7.1 The OLAP Queries	41
7.2 Typical OLAP Query Process	42
7.3 OLAP Queries Set	46
CHAPTER 8.....	48
OPTIONS AND ALTERNATIVES ANALYSIS	48
8.1 Best Versus Worst Case	49
8.2 Alternatives Standard Options	51
8.3 Alternatives Comparison for the Actual Load	59
8.4 Comparison of Alternatives Limits.....	62
8.5 Alternative 3 Tests of Scale	63
8.6 Index-Only Tables Performance	65
8.7 Temporal Aggregation.....	67
CHAPTER 9.....	70
EAGER VERSUS LAZY AGGREGATES COMPUTATION	70
9.1 An Aging Schema for Historical Data	74
CHAPTER 10.....	77
CONCLUSIONS AND FUTURE WORK	77
10.1 Requirements and Achievements	78
10.2 Potential Performance Improvements	80
10.3 Innovative Concepts and Future Work	81
REFERENCES	83

LIST OF FIGURES

Figure 1.1 – From Oliver et al Engineering Complex Systems with Models and Objects – FFBD of the systems engineering core technical process	2
Figure 1.2 – Network aggregates dimensions.....	5
Figure 2.1 – Network statistics generation and processing.....	9
Figure 2.2 – Network conceptual structure	10
Figure 2.3 – Network conceptual structure	11
Figure 2.4 – Current aggregation system modules	12
Figure 2.5 – Structure of functions to process statistics.....	13
Figure 4.1 – Architecture of the proposed solution	22
Figure 4.2 – Network aggregates database system model.....	23
Figure 4.3 – Entity-relationship diagram of the system	24
Figure 5.1 – Loading-process behavior diagram.....	33
Figure 6.1 – Aggregation process behavior diagram	38
Figure 7.1 – Typical query process behavior diagram	45
Figure 8.1(a) – Averages of Best (left) versus worst case (right)	50

Figure 8.1(b) – Trend lines of best (left) versus worst case (right)	50
Figure 8.2 – Performance of Alternative 3 with attributes defined as NUMBER(12,3) (left) and the default FLOAT (126 digits) (right).....	52
Figure 8.3 – Trend lines of serial local and remote aggregation (left) versus parallel local and remote aggregation processing (right)	53
Figure 8.4 – Serial loading and aggregation results (left), parallel loading and aggregation (middle), and total processing time (right)	54
Figure 8.5(a) – Non-clustered indices	55
Figure 8.5(b) – Indices clustered by snapshot	55
Figure 8.6 – Alternative 3 loading and aggregation performance differences when sessionstats table has no primary defined (left) and when it has a primary key defined (right).....	57
Figure 8.7 – Loading and aggregation times using Flat Tables for networks configuration data in the top two charts, and using IOT tables in the two charts at the bottom	58
Figure 8.8 – Alternative 1, 2, and 3, loading, aggregation, and queries averages, as well as averages total at the top; plus the trend lines at the bottom charts for the same data.....	60
Figure 8.9 – Loading plus aggregation capacity limits of the configuration used, for Alternative 1 (left), Alternative 2 (center), and Alternative 3 (right)	62
Figure 8.10 – Scalability of the proposed solution, from left to right network configurations have 1,350, 13,500, and 135,000 sessions	65

Figure 8.11 – Temporal aggregation of six days of network aggregates using standard SQL functions and time series cartridges.....	68
Figure 9.1 – Eager (top) versus lazy (bottom) aggregates computation	73
Figure 9.2 – Proposed network management data aging schema.....	76
Figure 10.1 – In the left chart there is time available (in seconds), used, in the right it has processing requirements in number of sessions, for every fifteen minutes, and what was achieved	78

LIST OF TABLES

Table 4.1 – Networks configuration used.....	29
Table 7.1 – Set of queries used on tests, frequency, number of runs, and type of processing	46
Table 8.1 – Loading, aggregation, and query average response time	61
Table 8.2 - Maximum capacity - no intervals between runs, no queries, serial loading, and aggregation processing	63

Chapter 1

Introduction

Since networks now encompass the world, and their proliferation is bound to increase in the near future, new tools to operate and administer them will be needed. The quantity of statistical data generated is overwhelming operators, network managers, network planners, and even the computers systems storing statistics. The disk space required to store statistical data related to a short period of a medium size network grows easily to the level of gigabytes and terabytes. Therefore, new forms of processing and storage of network statistics need to be developed to help the analysis, storage, and use of this data, not only for the reactive network performance management but also to support the proactive network planning.

1.1 The Systems Engineering Process

The study of the problem and its analysis follows the systems engineering core technical process approach, introduced by David W. Oliver, et al. in the book Engineering Complex Systems with Models and Objects. The core technical process is part of a higher level model, which includes the systems engineering process that covers all the issues of a typical systems development life cycle. The core technical process is presented in Figure 1.1. Each box in the system's engineering core technical process is further detailed and refined, but for a small team, and a short-term project such as this one, the level of detail

represented by the diagram, Figure 1.1, is enough, and does not require further refinements. This element constitutes part of the beauty of the methodology, because it can be applied to short-term and long-term, small or big projects. The approach was especially useful to organize the earlier phases of the research and development. During these phases, the current system was assessed, the requirements were defined, the structure and behavior models were created, the trade-off analysis was performed, and a build-and-test plan was developed.

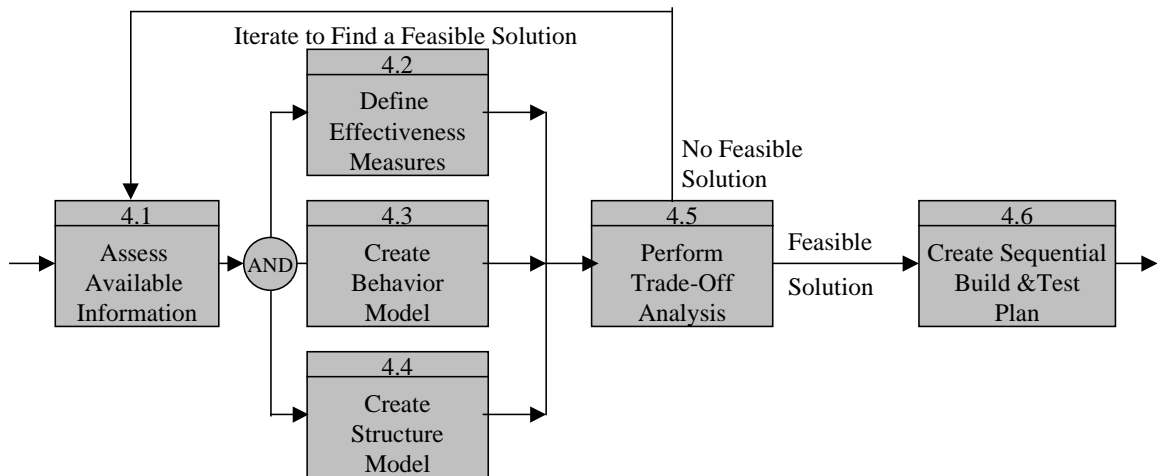


Figure 1.1 – From Oliver et al Engineering Complex Systems with Models and Objects – FFBD of the systems engineering core technical process

The following is a description of the six steps of the core technical process:

1. It evaluates available information, and missing information is gathered.
2. It defines a small subset of the requirements that will measure the success or failure of the system. These requirements are the most important of the system and the criteria for optimization.
3. It defines the desired behavior of the executable model.
4. It defines the alternative structure models with its components from which the system will be built.

5. It performs the trade-off analysis and designates which alternative design or architecture is best, based on the requirements and feasibility of the solution. The best design is selected, based on effectiveness measured values. This activity is part of the optimization process, when looking for a feasible design. Steps 1 to 5 are re-iterated; the requirements re-evaluated, and sometimes relaxed so that a feasible solution can be found. In situations where no feasible solution is found, after successive re-iterations, the project may be terminated for reasons of budget restraint, schedule overruns, or lack of candidate solutions.
6. It creates a plan to refine existing deemed-feasible solutions when they are found, providing an implementation plan for the selected design. The plan covers availability of resources, development of schedules, product versions, and product validation, etc.

Steps 2, 3, and 4 of this process are concurrent, evolving with the results of one affecting the other.

1.2 Research Goals and Proposed Aging Schema

In this study, efficient methods for computing, storing, and manipulating current and historical aggregates, derived from network management data, are investigated. The feasibility of using commercially available database software is evaluated along with its newest tools to perform these tasks.

An aging schema is proposed to keep historical data with finer granularity for recent data and coarser for older data. The nature of the network aggregates granularity also changes from the bottom of the network structure to the top,

becoming coarser and coarser. Holding statistics for the lifetime of a network will be possible without requiring storage of great quantities of data. Stored historical data will be ready and helpful in managing, forecasting, and planning the current and future network resources requirements and deployment. The work, besides supporting the needs of two areas of the ISO network management model, fault management, and performance management, by providing data with varied granularity that is ready to be used and easily retrieved, takes a big step into supporting network planning. For example, by comparing current traffic levels with historical data, network operators and managers can easily determine whether the traffic is normal or potentially faulty. The same information can be used by network managers and planners to draw traffic trend lines, which will indicate when specific resources capacity will be exhausted, and require upgrading. Still, on network planning and deployment, it helps in determining whether new traffic or new services can be absorbed by the current network infrastructure. For sure, this information helps in making the right decisions to achieve the best results.

1.3 Aggregates Dimensions and the Aggregation System

This project involves three dimensions of network management data, attributes dimension, network containment dimension (session, port, etc.), and the time dimension. Figure 1.2 depicts the network aggregates dimensions. Aggregates of network elements are values computed from session statistics collected from the network, through structured query language (SQL) functions sum, average, etc.. After computed, the aggregates are associated with each

network element, according to the network configuration structure. For example, by summing all the errors of all sessions, which use a specific port, a port aggregate is computed, and it gives the number of errors of that specific port. This type of aggregation is called *aggregation by network containment* or *containment aggregation*. The other type of aggregation that is covered here is *temporal aggregation*, which takes statistics collected by the minute and sums or averages them by the day, month, or year. The operation of taking by the minute or hourly attributes and aggregating them by day, week, or other coarser unit of time is called *temporal aggregation* or a *scale-up*.

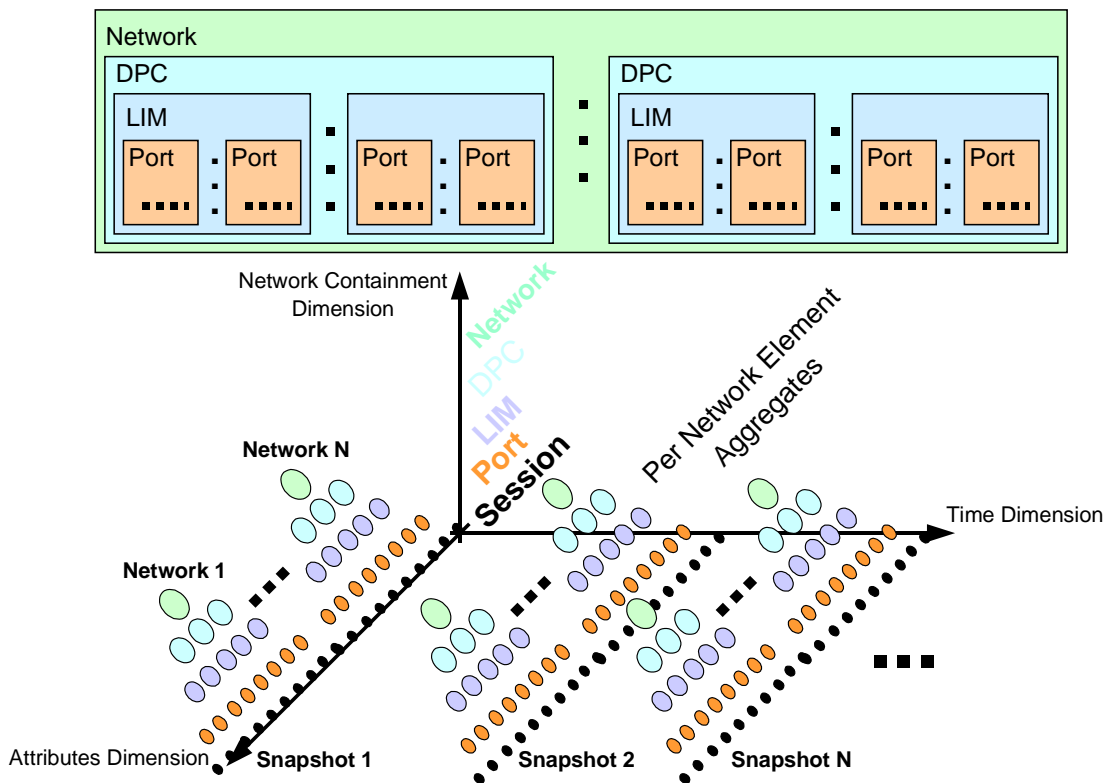


Figure 1.2 – Network aggregates dimensions

Computed aggregates are stored into the database and associated with the network elements to which they are related. Each network element has twenty

different aggregates per snapshot. A snapshot is a collection of network statistics related to a specific period. Each snapshot has an id, which is an integer with the date in seconds of the moment when the generation of the session statistics for the snapshot started. This choice aims to identify the snapshot with a meaningful number.

The developed system simulates incoming network management data, loads it into the database, and performs the aggregation by the network containment dimension, covering eleven levels of aggregation. A set of iterative running queries to simulate users query load also has been developed. The query set comprehends simple queries run against any table of aggregates and complex queries performing *drill-downs*. Temporal aggregation was also tested using views with Oracle8™ time series cartridges functions, and using only standard queries features to compare the performance of both. The performance of temporal aggregation is not as critical, since it is expected to run once a day, weekly, monthly, and yearly, not every fifteen minutes as the aggregation by network containment.

The system developed for the loading and aggregation of session statistics by network element, network, inroute, and outroute will be referred to in this thesis by the name of NADS (Network Aggregates Database System) or alternatively NATS (Network Aggregates Time Series).

1.4 Organization of Thesis Contents

The remainder of this thesis will provide details of the aforementioned work and is structured as follows. In Chapter 2, the architecture of the network used as the basis for this research is described along with the network management data collection model, as well as an abstraction of the current system model. The network of interest is a star satellite network. Chapter 3 introduces the systems engineering analysis of the problem and the derived system requirements. In Chapter 4, the architecture of the proposed solutions to process, load, and perform the network statistics aggregation is described. The behavior diagrams of the proposed solution are presented here, also. Chapter 5 describes in detail the process of generation and loading simulated network management data into the database. In Chapter 6, the aggregation process is described in detail; it is based on eager computation of aggregates using views materialization. Views and views materialization concepts are also explained in this chapter. Chapter 7 describes the development and testing environment, the typical query process, and the set of on-line analysis processing queries used to simulate users load and test query performance. In Chapter 8, performance issues are addressed, and test results are discussed. Results of tests of loading and aggregation of 1,300 sessions per snapshot running together with queries simulating network operators' load are presented. The results of three basic configurations are compared, showing the one with better performance. Then this project shows the test results to determine the capacity limits in megabytes and the number of sessions per hour and the alternative's best performer, which demonstrates the

proposed solution. These tests will also show the scalability of the three basic alternatives. In Chapter 9, the advantages of using an eager aggregates computation approach versus a lazy aggregates computation is analyzed. The paper also explains what eager and lazy aggregates computation means. Finally, Chapter 10 presents conclusions about the results of the developed research work.

Chapter 2

Network Management Data Collection and Processing

This research is based on a satellite network with a star topology. Figure 2.1 presents an abstraction on the macro level of the network management data generation, collection, and processing. The network contains elements that compute statistics, which are sent to the network management software and stored in a memory queue. Then, the statistics are transferred to a system that processes them, computing aggregates and storing them in a database.

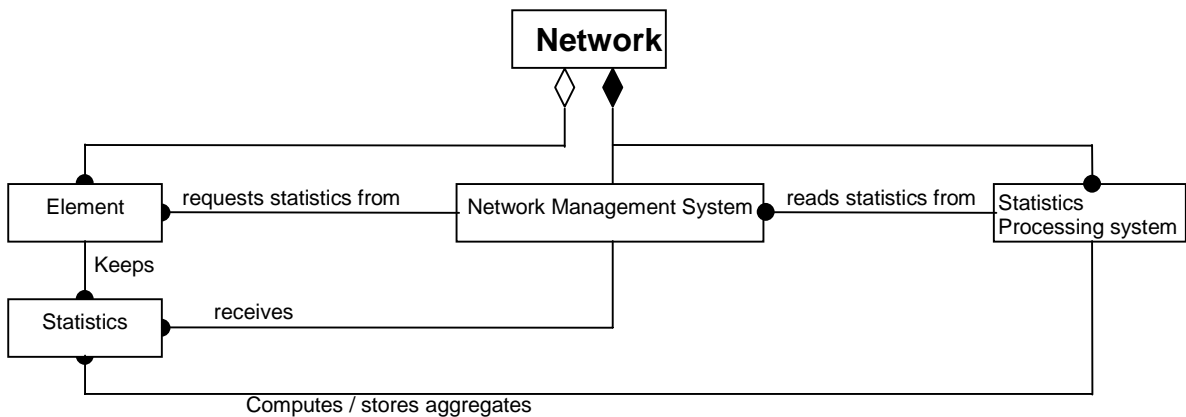


Figure 2.1 – Network statistics generation and processing

2.1 The Network Model

Figure 2.2 illustrates the conceptual network structure model of the network of interest. The network contains local and remote elements. The local network elements are data processing clusters (DPC), which contain LAN interface modules (LIM), which then contain ports. The remote side contains remotes,

each of which contains remote data processing clusters (RDPC), which contain remote ports. The sessions are established between local and remote ports.

The data is exchanged between remote sites through a satellite link to the central hub of each network. The communication from the hub to the remote travels through an outroute, and on the other way (from the remote to the hub), through an inroute. There is only one outroute per network and many inroutes. The outroutes use a much larger bandwidth than the inroutes. Typically an outroute transmits data at a rate of 512 Kbps (kilo bits per second), while an inroute (from a remote to the hub), transmits data at 128 Kbps.

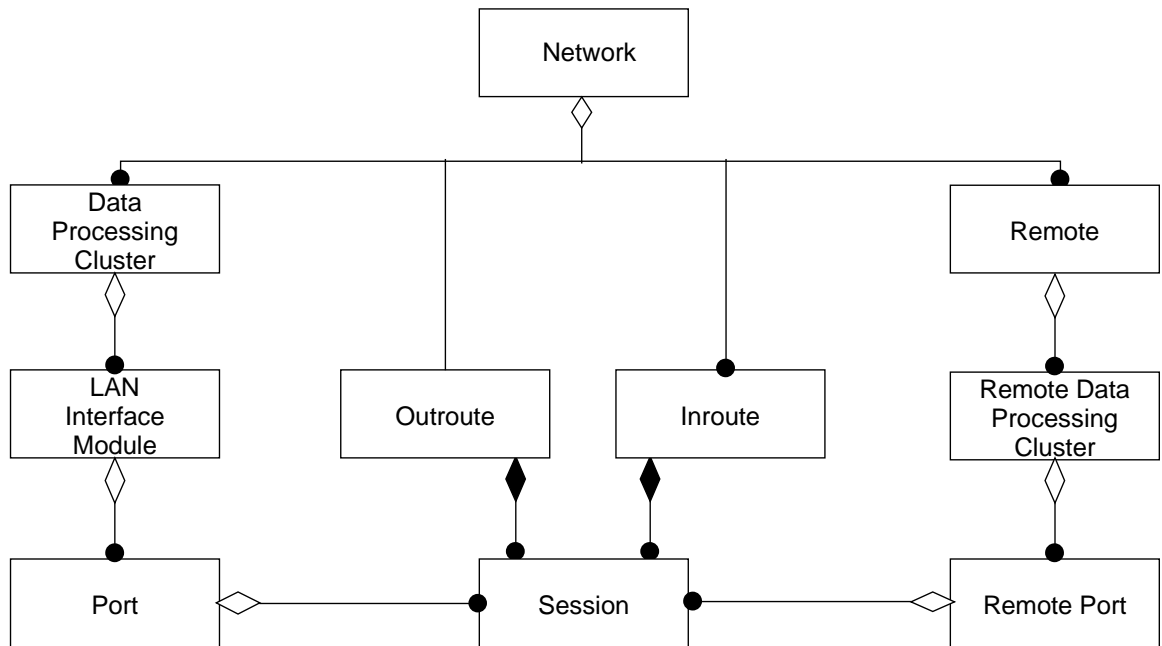


Figure 2.2 – Network conceptual structure

Messages broadcasted by the hub are received by all remotes, but acknowledged and processed only by those with the targeted destination addresses being discarded by the others.

All the network statistics generated and collected on this type of network are related to sessions. From them, the statistics related to the other network elements are derived.

2.2 The Current System

Currently, there is a system running, collecting, and processing statistics on an hourly basis. This research was developed to allow a more frequent collection and database loading of statistics, besides computing the aggregates by eleven network levels and storing them on the same database. In Figure 2.3, the diagram shows the current statistics collection and processing system.

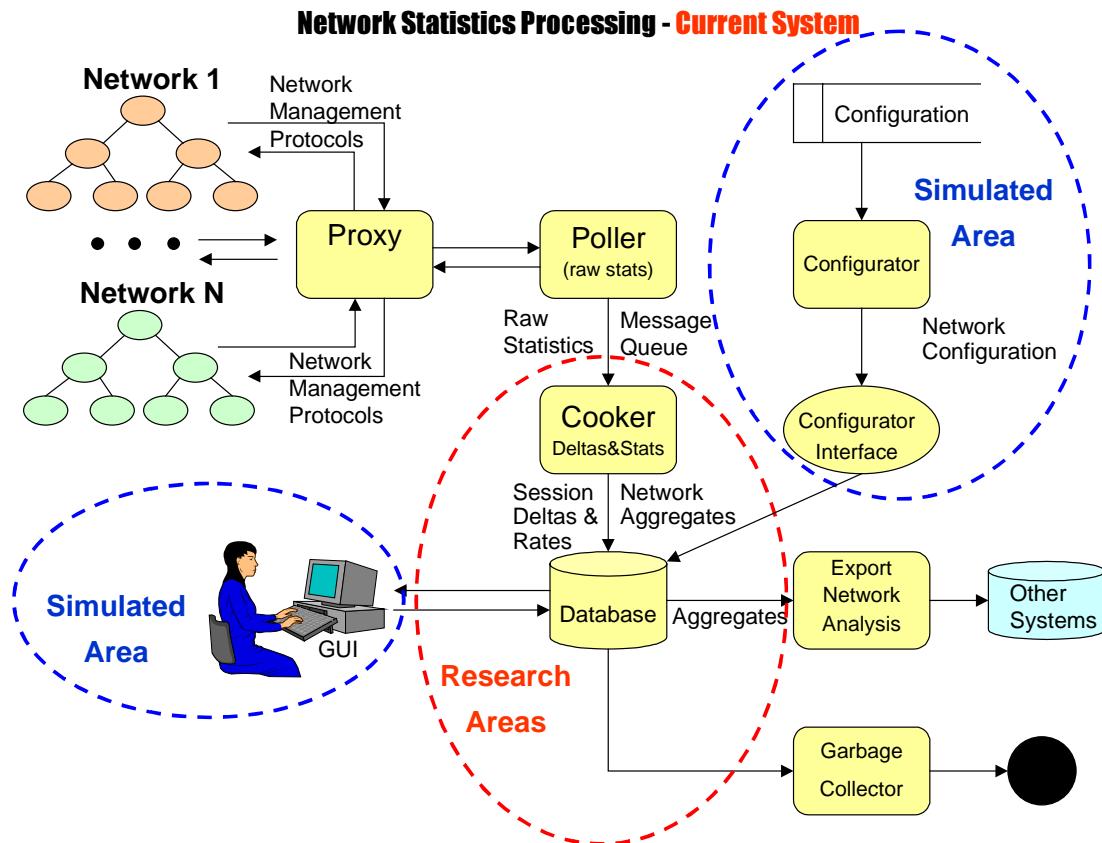


Figure 2.3 – Network conceptual structure

In the diagram of the current system, the marked areas that the research simulates and the areas wherein the research is focused may be found. Figure 2.4 presents a macro level structure of the current system modules, and Figure 2.5 provides more details about the functions of the process session statistics module.

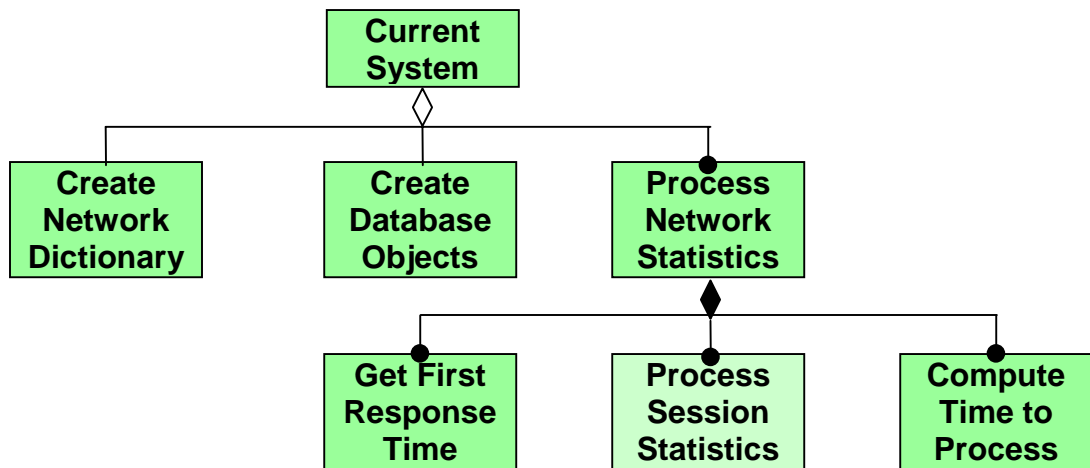


Figure 2.4 – Current aggregation system modules

The current system creates a dictionary of the network objects through the “Create Network Dictionary” module, then the system creates the database objects using the “Create Database Objects” module. In addition, after this phase, it processes the statistics and associates them with each network element. The “Process Network Statistics” module first obtains initial values through the “Get First Response Time” module, and from that moment on, it computes the session’s statistics, using data from two consecutive snapshots. It is this way because the received counters from the network elements are accumulators, and the statistics related to the most recent snapshot are obtained by subtracting the current accumulated values from the previous ones. A more

detailed description of this module comes later. After computing the current snapshot statistics, the “Process Session Statistics” module, computes statistics related to the time spent processing the network statistics through the “Compute Time to Process Statistics” function. Other than that, it contains various functions that are depicted in Figure 2.5. The main function is called “Stats_processSession,” and it processes session statistics response messages by sending out a statistics processing request message to the “Poller,” which stacks the incoming network messages in a queue and then processes the statistics inside it.

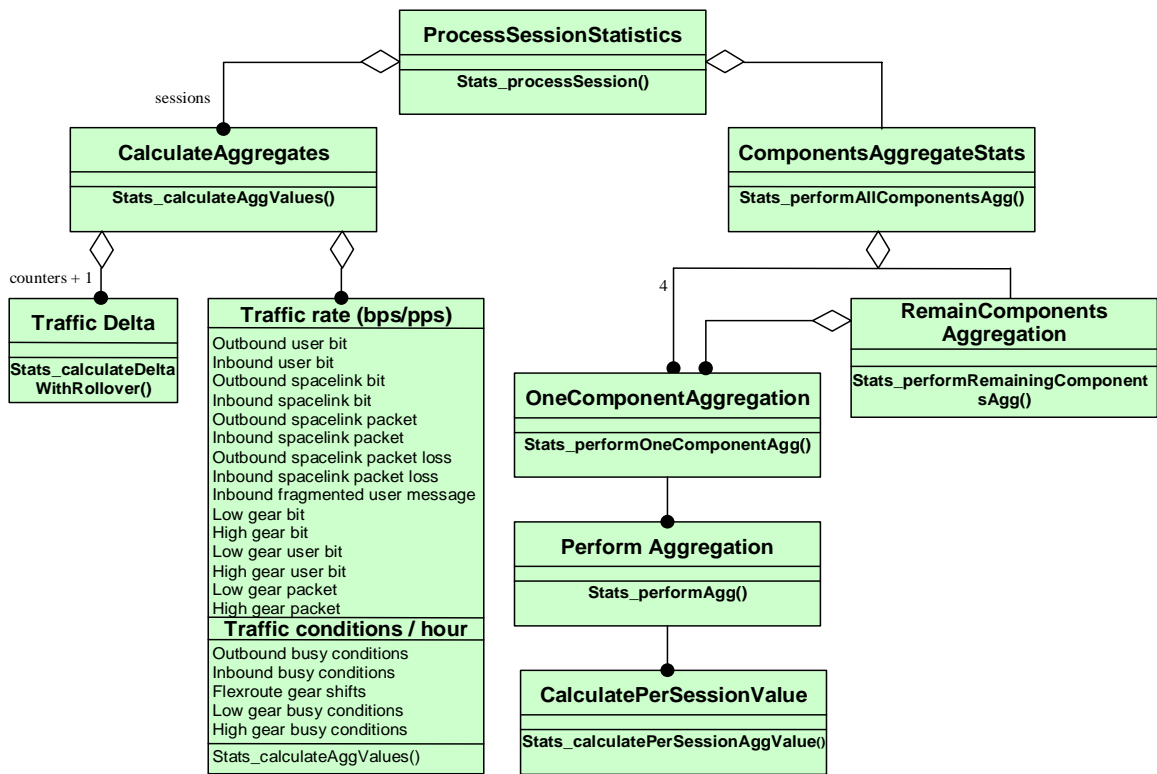


Figure 2.5 – Structure of functions to process statistics

The “Stats_processSession” function processes statistics of every single session. The values of each response message are placed in each session statistics dictionary list before starting the aggregation with those values.

The “Stats_calculateAggValues” function computes session statistics aggregation values for a session. Aggregation operations are of two types in this system: averages or sums. Session statistics aggregates are computed by taking the difference in values (delta values) of two consecutive snapshot statistics resulting from poll cycles. The poll cycle is the process of sending messages to the network elements and requesting accumulated statistics. The computation of aggregates is performed only when statistics related to a session are collected in two consecutive poll cycles. The computation of the delta values, performed by the function described next, takes into consideration the existence of roll over of the network element counters, which are not reset.

The “Stats_calculateDeltaWithRollover” function computes single delta values for session statistics, subtracting the current value from the previous one. Rollover between the two values is taken into consideration. So, if the current value is smaller than the previous one, there was a rollover that happens when the maximum of the counter is reached, and the counter returns to zero. In this case, the delta value is computed by determining the difference between the maximum and the previous value and adding the result to the current value.

The “Stats_performAllComponentsAgg” function performs session statistics aggregation (sums and averages) per network component, using per session statistics collected from the network, once every session depends on and uses a set of network elements.

The “Stats_performOneComponentAgg” function is responsible for retrieving each aggregate of each network component , recalculating it with the last delta value.

The “Stats_performAgg” function performs the aggregation either in two distinct ways depending on the statistical attribute by computing a new weighted average or simply by adding the new value to the existing one.

The “Stats_calculatePerSessionAggValue” function calculates a per session aggregate value. This computation is a weighted average, wherein the current average is multiplied by the number of previous snapshots, added to the new value and divided by the previous number of snapshots, plus one, generating a new average.

The “Stats_performRemainingComponentsAgg” function controls the execution of the aggregation carried out by the “Stats_performOneComponentAgg” function by calling it until the aggregation of every session per inroute, outroute, and network element is completed.

The structure of the above-described functions and their associations are pictured in Figure 2.5.

Chapter 3

The Problem Analysis and System Requirements

After studying the code of the current system, in order to understand the problem and discuss it, three research alternatives were defined involving three different architectures. As the study of the problem and software tools evolved, it progressed to a unique architecture for three different alternatives. Some of the reasons why only one architecture remained are explained next.

Following a systems engineering approach, to discard unpromising solutions, the option of computing the deltas and rates after loading the network incoming statistics on the database was discarded because it would clearly present an undesired worse performance. Computing the deltas and rates when the network statistics are in memory before loading them in the database is clearly better than loading the raw statistics in the database then reading them back to memory in order to compute the deltas and rates, finally storing the deltas and rates in the database. After concluding that, the raw statistics (accumulators) would never be used again after computing deltas and rates, this option was eliminated. Another reason why the initial three different alternative architectures became one at the end of the developed work is because it was found that time series cartridges, an Oracle8™ package, can use the same tables as the alternative of not using this software at all.

3.1 System Requirements

The minimal requirements to consider using any proposed system are

1. The system shall do the following in 15 minutes or less:
 - 1.1. Compute all the deltas and traffic rates for all the 20 attributes of 1,300 session's statistics.
 - 1.2. Load the deltas and rates for each session into the database.
 - 1.3. Perform the aggregation by network containment (Port, LIM, DPC, Network), inroute and outroute.
 - 1.4. Materialize the computed aggregates by writing them in database tables.
2. The system shall store the data in a format that allows a graphical user interface (GUI) to provide updated reports or graphics of the network resources utilization every 15 minutes.
3. The system shall support the storing of (temporal) aggregates for a long period with different levels of granularity for network statistics, providing fine data granularity for recent data and coarser data granularity for older data, following an aging scheme.
4. The system shall handle huge quantities (at least gigabytes) of historical data.
5. The system shall be flexible to allow storage of data for new networks or additional network elements with simple changes.
6. The system shall have SQL tools and allow their use to query the summarized or non-summarized aggregates stored in a database.
7. The system shall provide SQL tools to list top ranked networks, network elements, and sessions, etc.

8. The system shall compute and store aggregates in all dimensions correctly.

3.2 Effectiveness Measures

According to systems engineering principles, effectiveness measures, establish the criteria by which alternative designs will be judged. Effectiveness measures are a small subset of the system requirements that are so important that if they are not met the system fails, and if they are met, the system is successful. Based on this result the set of effectiveness measures for the researched system at hand were defined as the following:

EM1 - The system shall compute all the aggregates in all eleven levels correctly and write them into the database for all the 1,300 sessions with 20 attributes each in fifteen minutes or less.

EM2 - The system shall store the data in a format that allows the graphical user interface to access it to provide updated reports or graphics every 15 minutes.

EM3 - The proposed system tools shall provide means to query the data without requiring any programming.

EM4 - The system shall handle huge quantities (at least gigabytes) of current and historical statistical data.

EM5 – The system shall be more flexible, providing more tools and functionality than the current system.

EM6 – The system must be user-friendly.

3.3 Expected Outputs

1. Efficient methods for the computation and storage of a large number of aggregates of network elements, sessions, inroutes, and outroutes, across network hierarchies and time.
2. A system with the capacity of handling a large number of aggregates, from a variable set of **networks, network elements, and sessions**, defined by the operators. This means a system with a flexible structure to handle new networks with similar topology, network elements, sessions, inroutes, and outroutes.
3. A set of tools that comes with Oracle8™ Time Series Cartridges, a software package, to query and manipulate the data.
4. A database of current and historical data, with different levels of granularity to support network operation, network management, and planning.
5. Aggregates queries of the type: top 10 or top 20 only.

3.4 Sequential Build and Test Plan

1. Prepare Solaris™ 5.1 for Oracle8™ installation.
2. Install Oracle8™ Enterprise Edition (RDBMS), which includes PL/SQL.
3. Configure Oracle8™.
4. Install Oracle8™ Time Series Cartridges.
5. Configure time series cartridges package.
6. Define tables to store networks configuration according to the data diagram.
7. Define tables to store aggregates following the data diagram.

8. Develop and test a program in C using Proc*C to generate simulated network statistics, compute deltas, and per session rates, loading them into the database. The program will have to generate a report with processing times and rates at the end of the processing of each snapshot, which must be recorded outside the database.
9. Create the database views, to perform the aggregation and views materialization.
10. Develop and test a program using C and Pro*C to compute all the aggregates and to store them in database tables, following the network configuration stored in the configuration tables. After the computation and storage of aggregates on each level, the processing time must be reported for performance analysis and must be recorded outside the database.
11. Develop a set of programs that iterate running queries at pre-determined intervals to simulate the load generated by operators and network managers, computing response times of these queries for later analysis. As stated above, the processing times and statistics related to the performance of these programs must be written outside the database to minimize the distortion caused by these operations on the final processing results.
12. Run loading, aggregation, and queries using the many different alternatives to determine which is the best solution.
13. Run loading and aggregation tests without intervals to determine the limits for loading and aggregation processes for the configuration used.
14. Test time series cartridges related functions.

- 14.1. Define calendars table.
- 14.2. Define calendars.
- 14.3. Validate calendars.
- 14.3. Define metadata tables.
- 14.4. Define time series views.
- 14.5 Test time series functions.
15. Test eager versus lazy aggregates computation.
16. Analyze test results.
17. Prepare research presentation.
18. Prepare project (thesis) paper.

Chapter 4

Architecture of the Proposed Solution

As mentioned before, a unique system design was defined for the three alternatives, after the analysis of the problem as well as the software tools. The three alternatives and the testing results, determining the best one, will be presented in the next chapters. Figure 4.1 shows the proposed system architecture.

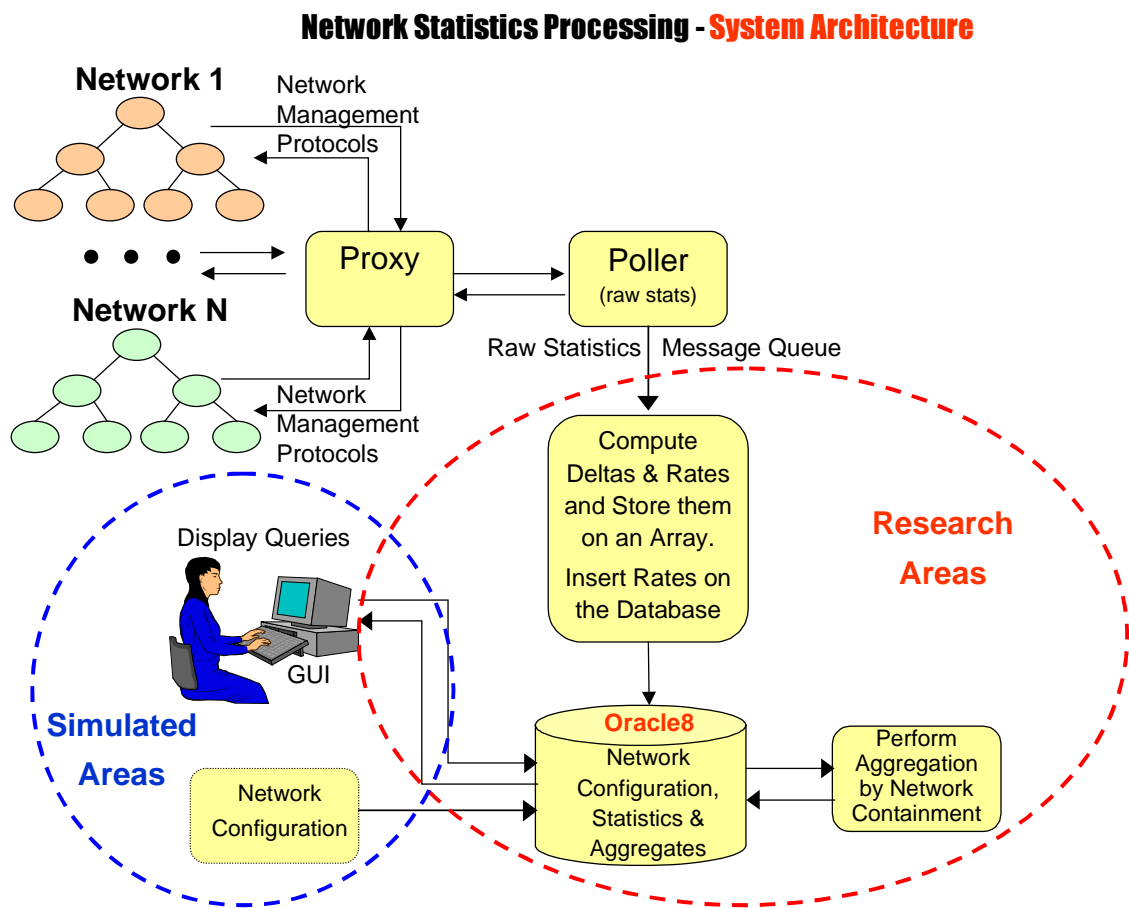


Figure 4.1 – Architecture of the proposed solution

The proposed system has a loading process, which generates simulated network statistics, and loads them into the database. It also has an aggregation process, which reads the loaded-per-session statistics and aggregates them by network containment. To perform temporal (by time) aggregation two other processes were developed, which are similar to the aggregation process mentioned above. One uses standard queries, views, and views materialization, and the other uses time series cartridges functions, on views to perform the aggregates scale-up, before storing them in materialized views. Independent of that some time series cartridges functions were tested. A dozen programs to run queries, simulating the load generated by the ten operators expected to be using the database, were also developed and run. All these processes iterate for as many times as specified at given time-intervals, generating statistics about response times and other performance information. These processes architecture and functionality, will be described in more detail in the next chapters. Figure 4.2 shows the associations of the system processes.

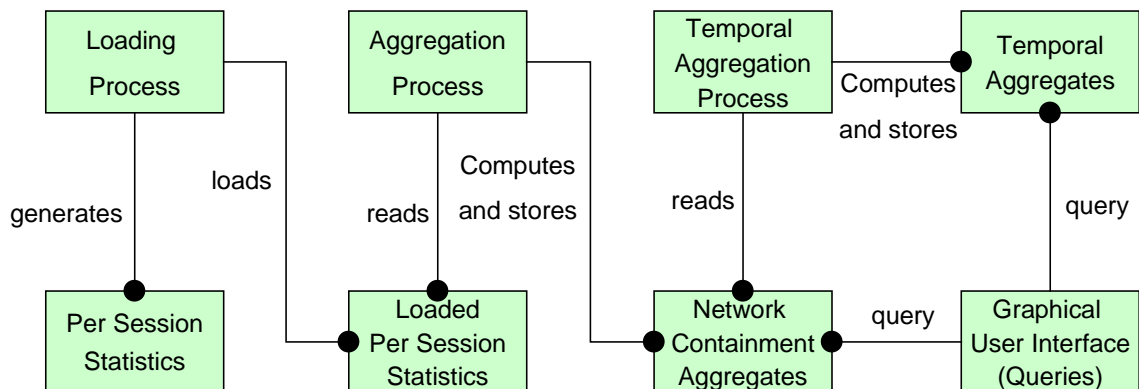


Figure 4.2 – Network aggregates database system model

C was the language of choice in developing the processes to be used in this system, because among other reasons, it is deemed to be a suitable language for this kind of application and has a very good performance record.

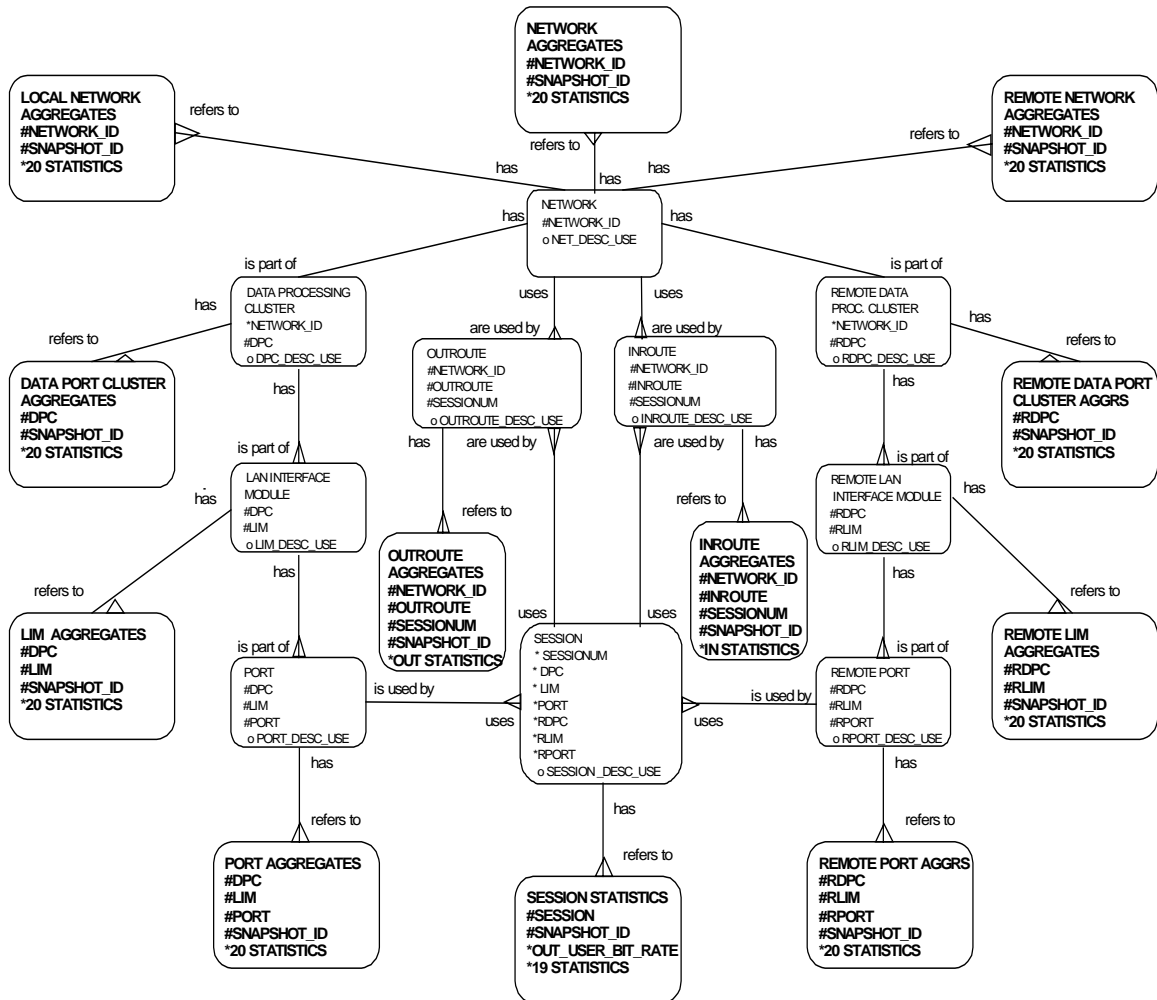


Figure 4.3 – Entity-relationship diagram of the system

The entity-relationship for the three alternatives, as well as the data schema, are the same, using different database options, different types of tables, or different processing approaches. Figure 4.3 depicts the entity-relationship diagram of the system, showing the networks configuration entities and their

aggregates. The data diagram has a structure very similar to the entity-relationship diagram, having a table for each entity. The data schema contains tables for two main purposes. The first set of tables stores data describing the network configuration and the other set stores session statistics and network element aggregates. The writer will refer to the former tables as *configuration tables*, and the later as *aggregate tables*.

Frequently, data warehouses use a *star schema* to represent multidimensional data models. They have a central table called *fact table*, and around it, the *dimension tables*. Herein, *configuration tables* play the role of the *fact table*, and the *aggregate tables*, are the equivalent to *dimension tables*.

There are a total of ten configuration tables, eleven aggregate tables; in addition, the session statistics table, which was named sessionstats. Besides, the eleven aggregate tables mentioned above, a number of temporal aggregate tables may be created, depending on the aging schema. To perform temporal aggregation tests, nine aggregate tables were created to materialized data derived from views. These tables are connected to the *configuration tables* as are other aggregate tables. For temporal aggregation tests, no local or remote network aggregate tables were created, only a network aggregate table.

There is a list of twenty attributes, common to any list of aggregates, not listed in the entity-relationship diagram, to keep it to one page size and readable. The list of attributes is referred to in this diagram as *20 STATISTICS and covers bit rates, packet rates, transmission errors, and other network management statistics.

4.1 System Physical Model

From the entity-relationship diagram presented in Figure 4.3, the physical data model is derived, having 10 configuration tables, 11 aggregate tables, plus the table sessionstats.

Before discussing the physical design, some database technical terms must be introduced:

- Row - any set of fields (or a line) in a table also known as a record.
- Primary key - one or more columns uniquely identifying any non-null row of a table. When defined, the primary key does not allow the creation of a second row with the same identification.
- Foreign key - one or more columns whose values are based on the primary or candidate key values of another table. It is used to associate the row or rows of one table to the row of the other, establishing a relationship. When defined it checks the existence of the primary or candidate key in the parent table to authorize the creation of a row with a specific foreign key in the dependent table.
- Aggregates – are attributes of network elements computed by using SQL aggregation functions and the group-by clause.

Here is the list of the physical tables of the system and their contents:

- NETWORKS – contains network ids.
- DPCS – contains the local DPC names. Each DPC is associated by means of a foreign key (network_id) to its network.

- LIMS – contains the network LIM numbers. Each LIM is connected to a DPC by a foreign key, which, in this case, is the DPC number.
- PORTS - contains the port numbers. Again, each port is related to a DPC and a LIM through a foreign key.
- REMOTES - is a table similar to the table DPCS containing remote names. As for the DPCS tables, there is a foreign key, attaching each remote to a specific network. The developed software may sometimes name this table as RDPCS, and in that case, RDPCS is named as RLIMS.
- RDPCS - is the remote counterpart table of LIMS, containing the numbers of the remote DPCs. Each one has a foreign key associating it to a specific remote. The developed software may sometimes name this table as RLIMS.
- RPORTS - is a table containing the list of existing remote ports and their respective remotes and remote DPCs, which constitutes a foreign key, pointing to the RDPCS table.
- SESSIONUMS – contains all valid session numbers and the respective DPC, LIM, port, remote, remote DPC, and remote port.

By defining the tables with these foreign keys, the consistency of the network configuration is assured. This way, the database does not allow a lower level network element to be defined if the upper elements do not exist or are incorrectly specified.

The table sessionstats, which receives network session statistics, uses a foreign key to assure the session number of the session statistics loaded is valid.

By assuring the validity of the session number, it can be sure that this data is related to known network elements, and the network elements to which it relates are known.

The session number is unique and is used to associate session statistics with the right network elements, using the foreign keys and configuration information stored in the table sessionums and the other configuration tables.

Each configuration table has one or more tables of aggregates associated with it. These aggregates are related to the type of element the configuration table describes. The aggregate tables contain, besides the twenty attribute values, a valid foreign key, which connects each row to a valid network element in the configuration table, plus a snapshot id. The foreign key to the configuration table, plus the snapshot id, uniquely identifies each row of the aggregate tables. The temporal aggregate tables rows are uniquely identified by the foreign key, which connect them to the network elements, and a date, which can be one or many fields, depending on the temporal aggregation model used.

At the top of this hierarchy, there is an aggregate table for the local aggregates of networks and another table for the remote aggregates of networks. These aggregates are then consolidated into one table, having aggregated attributes by network.

The table sessiontstats, where session statistics coming from the network are inserted, is at the bottom of this structure.

As part of the temporal aggregation schema, to perform temporal aggregation tests, one table for each element type was created, similar to the aggregate tables, where daily aggregates were materialized.

4.2 Used Networks Configuration

Four network configurations are defined for the tests, named NET01, NET02, NET03, and NET04. Table 4.1 lists the components of each network configuration used on the alternatives testing and validation. Other configurations were used to test system capacity limits and scalability of the proposed solution.

Table 4.1 – Networks configuration used

Network	DPCs	LIMs	Ports	Sessions	Outroutes	Inroutes	Elements Total
NET01	2	11	19	152	1	10	195
NET02	4	16	34	272	1	16	343
NET03	6	44	90	720	1	44	905
NET04	2	16	32	256	1	16	323
Total	14	87	175	1,400	4	86	1,766

The total number of sessions is 1,400. The networks have dissimilar configurations deliberately to try to reflect real network configurations. Initially, network configurations were defined for the tests with one session per port. This type of configuration is not the same as the real world and resulted in longer aggregation times, mainly due to generation of more disk I/O, which then became a bottleneck. In the end, the same aggregation process was tested for a configuration with eight sessions per port and the processing times were, as expected, much smaller.

Chapter 5

The Loading Process

The loading process was written in C and SQL Pro*C to generate simulated session statistics, loading them into an Oracle8™ database, reporting processing times and database insertion rates after that. This information was used to accomplish performance analysis to produce the tables and graphics presented in this report. Usually the loading process iterates while running in the background, generating statistics at pre-specified intervals and loading them into the database.

Next, a complete description of the parameters and sections of the whole process is attained.

5.1 Loading Process Parameters

The activity of the loading process is controlled by five different parameters, which are read at the start. The parameters and their functions are

- Period – defines in minutes how often snapshots of session statistics are generated. The default is 15 minutes.
- Records rate – determines the number of sessions per snapshot to be generated. Default is 1,400 sessions per snapshot.
- Iterations – controls how many snapshots are to be generated before resuming processing. Default is 96, which is the number of snapshots for one day, when the period is 15 minutes.

- Percent active - indicates the percentage of the total number of sessions (records rate) that will have a record generated and loaded. This simulates non-responding sessions. Default is 95%.
- Serial - This last parameter is used on tests of capacity limits, where the interval is zero. It determines where the loading and aggregation processes are run serially or in parallel. Zero indicates that both processes will run simultaneously; one indicates that loading and aggregation processes run sequentially, one after the other.

5.2 Loading Process Sections

After defining C and SQL (Pro*C) variables, the process allocates memory pointers, which will be used to allocate and store session statistics in memory. Then the process reads the control parameters, validates them, and allocates memory sufficient to store statistics for all the specified number of sessions per snapshot. After these operations, the process connects itself to the database, using a database-defined user name, and a valid password. Next it initializes time variables then prints the starting date and time.

It is at this point that the loop generating session statistics starts. Every time a random variable, between 0 and 100, is higher than the percentage of sessions active, the generation of the session record is omitted. The process uses random functions, and some formulas trying to generate attribute values for the session statistics, at least proportional to the real world. To simulate the processing load of a real situation, per session statistics are accumulated in memory, the opposite of what happens normally but requiring the same memory and

processing power. After this operation, a session statistics row is inserted into the database. Counters are updated and the generation of session statistics followed by insertion goes on, until the specified number of session statistics per snapshot is reached. The insertions are committed to the database in batches, according to the value of a variable used for that purpose.

After all the session statistics for a snapshot are generated, accumulated, and loaded into the database, the snapshot id is stored into a table called `last_snapshot_loaded`. However, before this insertion is performed, this table is locked into an exclusive mode, and its content is checked. This operation is done to determine if there is an aggregation process running to avoid having more than one aggregation process active at any one time. Activation of more than one aggregation process simultaneously causes resources contention, degrading the performance of all active processes. This disposition results in many problems and longer snapshot aggregations with chances of overloading the system, starting processes that never end, as experienced during system tests. Therefore, if this table has any substance (snapshot id) before inserting the snapshot id of the last snapshot loaded, it means that an aggregation process must be running, so there is no need to start a new one. On the other side, the aggregation process re-starts itself aggregating and deleting snapshot ids from this table, until all snapshots loaded are aggregated. The locking mechanism is to assure the perfect synchronization of these two processes.

After the previous operation and as soon as the process disconnects itself from the database, statistics related to this run are computed and printed, the

locked table is released, and if appropriate, another aggregation process is started. Then the loading process either enters a sleep state, waits for a started aggregation process to end (for serial runs), or starts another snapshot session statistics generation.

The last two sections of the loading process report statistics and handle SQL errors, respectively.

The behavior diagram of this process is presented in Figure 5.1.

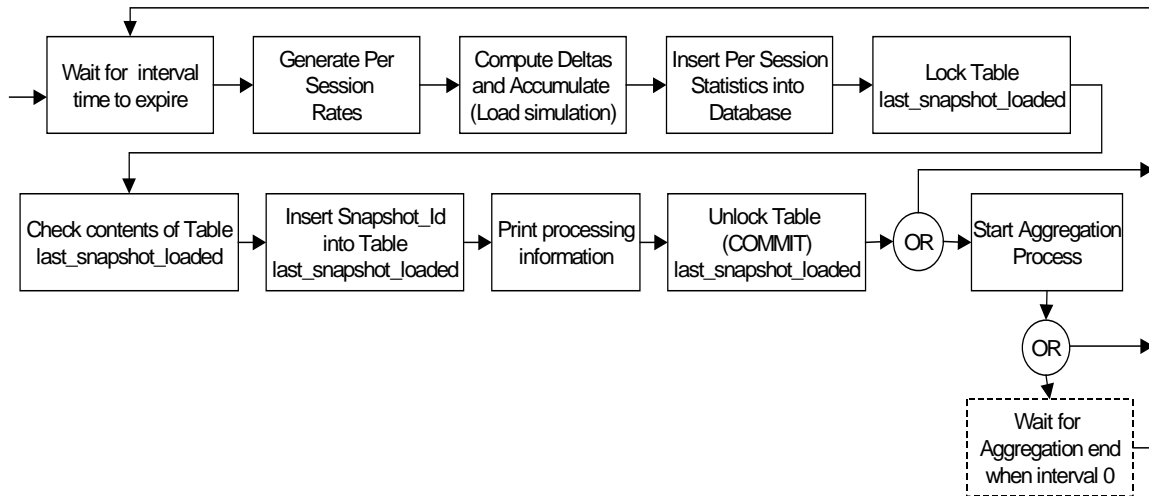


Figure 5.1 – Loading-process behavior diagram

Chapter 6

The Aggregation Processes

The network containment aggregation process, as well as the temporal aggregation processes, were written in C and SQL Pro*C. It aggregates session statistics by port, LAN interface module (LIM), data processing cluster (DPC), and network by the local side. Also tested running in parallel, it is suggested running the remote aggregation by remote port, remote data processing cluster, remote, and network (remote side) after the end of the local aggregation. Then computing inroute and outroute aggregates takes place before the local and remote per network statistics are consolidated.

At the end of each aggregation, the processing time is computed and printed. This information was the basis used to accomplish the process performance analysis and to produce the tables and graphics presented later in this thesis. The aggregation process is either activated by the loading process or restarted by itself at the end, when there are snapshots already loaded, but not aggregated.

Two temporal aggregation processes, very similar to the process just described, were developed to test temporal aggregation. One of these processes uses the time series cartridges package functions to perform the aggregation by time, the other uses standard SQL functions. The main difference here is that the first, uses a date type field and time series scale-up functions to accomplish aggregation by time, the second uses independently defined fields for year,

month, day, and hour in order to use the standard SQL group-by clause to achieve the aggregation by time. Unlike the aggregation by network containment process, these processes do not re-start themselves, since they are expected to run, at most, once a day. As they do not run frequently, their performance is not as critical. Meanwhile, two processes were developed using different tools to compare and determine which one performs better.

The temporal aggregation processes to compute temporal aggregates use aggregates of network elements computed previously, as the base data, for performance reasons.

The aggregation process by network containment will be referred to as the aggregation process, while the others will be referred to as temporal aggregation processes.

6.1 Views and Views Materialization

In this research, a very practical approach involves bringing to life the solution of real problems, using views and views materialization, described as “the most important asset of the relational model,” as stated in “Materialized Views and Data Warehouses” by Dr. Nick Roussopoulos.

Computation of aggregates in this system is based on views, and views materialization.

Views are database objects in the form of queries that logically represent a table. They do not have any stored data, and the data resulting from views is derived from one or more physical tables that occupy their own storage and have their own data. Many times, they are used as tables, but views have to derive the

data they represent every time they are used. This situation requires doing the same I/O and the same computations repeatedly, and when this happens frequently, against the same data, lots of resources are used to perform the same repetitive work, making them inefficient.

Views materialization overcomes the problem of redoing the same process, reading the same data every time a view is referenced by storing the results of the view processing in a real table. In some instances, materializing a view has the drawback of taking storage space. In the case studied herein, the used space becomes an advantage. As views are materialized, the data from whence this summarized or aggregated data was derived can be deleted without loss of information, although losing in granularity but saving disk space.

The aggregation processes described here materialize views by inserting the results of queries run against these views into tables. The views are extensive queries, using SQL aggregation functions, and the group-by clause to compute the aggregates, which are then materialized, as they are inserted into tables. SUM and AVG (average) are the most common SQL aggregation functions used in this system to compute the aggregates of rows grouped by the SQL group-by clause, achieving in this way the results of aggregation by network containment or by time. The operation of taking attributes and performing computations in conjunction with a group-by to derive the next level attribute values of coarser granularity is called a *roll-up* or a *scale-up*.

Besides aggregating attributes by grouping tables' rows and using aggregation functions, the aggregation by network containment process based

on views joins statistics of the previous network level with current network configuration level data, producing the current level aggregates. The aggregation processes use no parameters.

6.2 Aggregation Process Sections

The aggregation process begins by defining C and SQL (Proc*C) variables. Then the process connects itself to the database, using a database-defined user name and a valid password, as does the loading process.

Consequently, the program selects the oldest snapshot id from the last_snapshot_loaded table where the loading process inserts them after each snapshot is loaded. As mentioned, the snapshot id is an integer representing the data in seconds of the starting time of the generation of the session statistics for that specific snapshot. Once the snapshot id is retrieved, the table snapshot_to_aggregate is cleaned, and it is inserted in this table. This table has only one row, a unique column, which is the id of the snapshot being aggregated. All the aggregation views refer to this table to identify the rows of the snapshot to be aggregated and perform the aggregation by network element. To store the snapshot id, a table was chosen to simplify and speed up the queries of the aggregation views.

Next the process initializes time variables and prints the starting date and time. What comes next is the insertion into aggregate tables of the aggregates derived from the views, materializing them. As mentioned earlier, serial aggregation is suggested. The aggregation process first performs the aggregation by local port, LIM, DPC, and network. Then by remote port, remote

DPC, remote, and network (remote side) by inroute and outroute. Finally, it consolidates the local and remote per network aggregates by averaging them. At the end of the aggregation by each network element, it reports the time spent doing the aggregation.

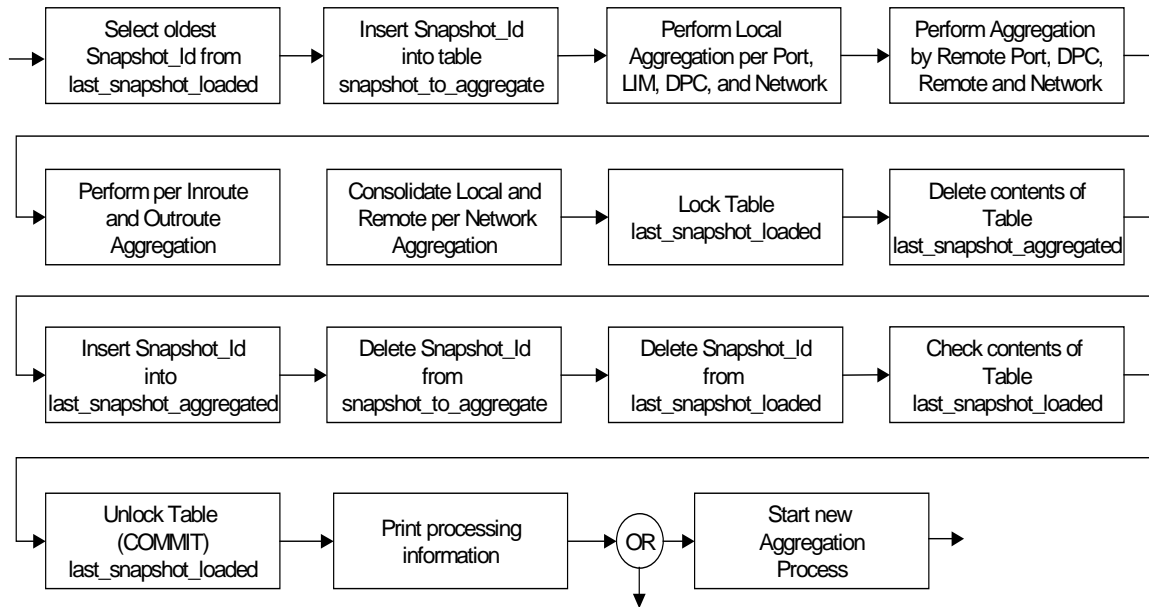


Figure 6.1 – Aggregation process behavior diagram

Once the eleven classes of aggregates computation and materialization are complete, the last_snapshot_loaded table is locked in exclusive mode. At this point, the previously aggregated snapshot id is deleted from a table called last_snapshot_aggregated, the latest is stored there. This way, there is no need for the operators, network planners, and network managers to run a query scanning the index of the aggregate tables to determine the snapshot id of the last snapshot aggregated. Referencing this table in the *where* clause of their queries to identify the last snapshot aggregated will be much faster and more efficient. This is a suggestion to reduce disk I/O improving overall performance,

not to mention operators and other professionals' time. From the snapshot_to_aggregates table, the id of the last aggregated snapshot is then deleted along with the last_snapshot_loaded table. Before committing all these changes to the database, unlocking the last_snapshot_loaded table, and disconnecting from the database, the existence of loaded snapshots not yet aggregated is checked in the last_snapshot_loaded table.

After committing all these changes and unlocking the last_snapshot_loaded table, the total aggregation time is computed and printed, and, if there are snapshots loaded to be aggregated, another aggregation process is started before resuming this process execution. The process is designed in this manner to catch up on aggregation by network containment after a delay. A server problem, a system congestion due to running daily or monthly temporal aggregation, or system and database maintenance are examples of possible causes for delaying the aggregation process.

As it happens in the loading process, the last section of the aggregation process is the one that handles SQL errors. The behavior diagram of this process is presented in Figure 6.1.

6.3 Temporal Aggregation Process Sections

The temporal aggregation processes developed are very similar to the aggregation process, except that they do not use any auxiliary tables, since they perform the aggregation by time, of all the contents of the network elements aggregate tables. They were developed to provide examples of how aggregation

by time can be accomplished, to test time series cartridges performance against standard SQL tools, and to provide performance data for temporal aggregation.

The temporal aggregation process has to be refined to meet the aging schema requirements, not clearly defined at the current time.

One suggestion is to break the aggregation by time in three independent processes, started one at a time, after aggregation by network containment is complete, at appropriate hours, probably soon after midnight. The suggestion to break this process in three, is to reduce the chances of running containment and temporal aggregation together. Therefore, the temporal aggregation should be broken in local, remote, and routes plus network aggregation. In this manner, it will be possible to run temporal aggregation in between aggregations by network containment.

Chapter 7

Development and Testing Environment

This research was developed using the following software: Solaris™ 2.6, Oracle8™ Enterprise Edition 8.0.5 (including Time Series Cartridges), C, Pro*C, and embedded SQL. The hardware used was a Sun® Workstation Ultra 10 with a Sun® Ultra5_10 sparc sun4u processor, 128 MB of RAM Memory, and one disk. The disk has a practical capacity of approximately 50 I/Os per second, depending on the characteristics of the I/O. One gigabyte of the disk space was dedicated to the database, which was called Network Aggregates Database System (NADS). NADS is the name also of the tablespace used. The allocated SWAP area size occupies 217MB of disk space.

7.1 The OLAP Queries

To simulate the load and to test the response time of network management data analysis queries, a set of processes was developed to try to emulate the load of a real day-by-day network operation. As with the loading process, these processes iterate at specified intervals of time for as many times as required, according to a specified list of parameters. Still following the same track after each run, these processes compute and report processing statistics of response time and number of fetched rows. They were also developed in C and SQL (Pro*C).

The queries on these processes run always against the data of the last snapshot aggregated. The id of that snapshot is retrieved from the table `last_snapshot_aggregated`, instead of being derived every time it is required. This reduces disk I/O, processing, and load in general, providing better performance.

The on-line analysis processing (OLAP) queries, developed to simulate the operator's work, query top sessions, ports, LIMs, and DPCs. Four programs also perform *drill-downs* from local and remote DPCs, down to LIMs, then to ports, and some of them to sessions, selecting the DPCs within which a chosen attribute has the highest rate or sum per session, and then choosing the network element with highest value, for that attribute. The drill-down operation is the reverse of a *rollup* or aggregation.

To compare the performance of queries against materialized views with queries without using materialized views, a similar set of processes were developed to generate the same output results, performing the aggregation each time the queries were run. The results will be discussed later in the performance issues and eager-versus-lazy computation of aggregates sections.

Next, a thorough description of the parameters and different sections of a typical query process will be provided.

7.2 Typical OLAP Query Process

As with other processes, query processes begin by defining C and SQL (Pro*C) areas and variables. For each class of network elements queried, a different structure with different header fields and one attribute is defined. The

header fields are the ones that vary according to the element type; the list of attributes is common to all types. Following the definition of the data structures and working variables, memory to store data fetched from the database is dynamically allocated.

The next step is to connect the process to the database, using a valid database user identification and a password.

These processes were designed to query any one of the twenty attributes common to any network element, but one at a time. Therefore, after the connection to the database, the process displays a list of the twenty attributes available for queries, asking for three parameters:

- Interval – to determine how often the process (queries) will run.
- Number of iterations – which defines the number of iterations or how many times the process will run.
- Attribute – a number from 1 to 20, corresponding to one of the attributes on the displayed list.

After reading and validating the parameters, the processes select the correct queries to query the right tables, retrieving the header fields and the chosen attribute values.

The next operation these processes perform is to start a loop that runs for the given number of iterations. Inside the loop in some cases, this process begins by initializing variables like top DPC, top LIM, and top port. They are used to select top network elements by any of the twenty attributes. Using a drill-down as an example, what is usually done - and this can be done using other methods - is to

compute a per session rate or attribute sum of each DPC. Then the one with the higher per-session value is determined. The LIMs of these DPCs are then queried, and the top one selected using the same method, doing the same for port, finally querying and listing the top sessions of the top port. This way the sessions that are responsible for the highest values of the chosen attribute can be found.

Typically, the query text is copied to an SQL area in memory, and then a cursor is prepared, declared, and opened. After this, the header of the query report is printed before entering an insider loop, which fetches the rows selected by the cursor from the database and then prints them on the report. In this loop, it also computes the per session values of the specific attribute to determine the uppermost LIMs, ports, or sessions. When the insider loop ends, the cursor is closed, and another cursor is opened to perform the queries, retrieving data of the uppermost network element on the next lower level, and the process goes on. Other criteria can be used to select specific network element statistics, depending on the needs.

A query simply looking for the top elements of a specific class of network elements is completed by querying and sorting them in descending order, then fetching from the top only as many as desired. SQL is not used to select the uppermost elements because it would require more resources than simple fetches. It is suggested to have this type of query pre-formatted, which will provide better performance with lower use of resources, leaving it to the user to write queries that are not as frequent and common.

At the end of the query or drill-down response time information and number of fetched rows is printed for performance analysis, and the process enters sleep state until the running interval of time expires.

Once the query loop is run for as many iterations as required, the process disconnects itself from the database and stops. As on any developed process, at the end there is the section that handles SQL errors.

It was not the purpose of this research to test and use a graphical user interface tool, and as it was not available, text reports were printed with the results of the queries instead.

Figure 7.1 presents the behavior diagram of a typical query process, developed for the purpose of this research.

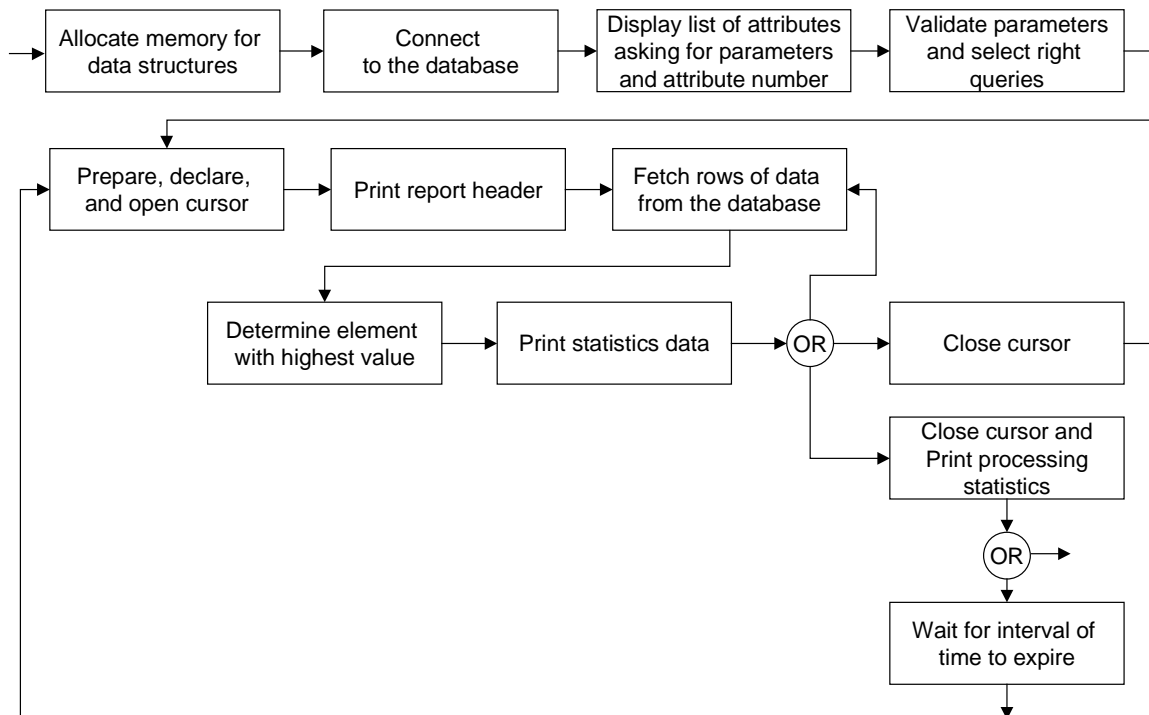


Figure 7.1 – Typical query process behavior diagram

7.3 OLAP Queries Set

A set of queries was put together to simulate operators, managers, and network planners query load, and further to assess queries performance, either using or not using materialized views. Table 7.1 lists the set and type of queries used, running frequency of the queries in minutes, along with the number of runs used to compute the average response times. The last column indicates that queries were run in parallel (P) with loading and aggregation and incidentally with one another. As in the real world, aggregation runs in sequence (S) and after the loading process ends.

Table 7.1 – Set of queries used on tests, frequency, number of runs, and type of processing

Process Type	Frequency (e.g. every 15 minutes)	Number Of Runs	Sequential or Parallel Processing
Loading	8	96	S*
Aggregation	8	96	S*
Queries:			
Top Local DPCs	2	406	P
Top LIMs	5	162	P
Top Local Ports	5	162	P
Top Remotes	2	406	P
Top Remote DPCs	5	162	P
Top Remote Ports	5	162	P
Top 20 Inroutes	5	162	P
Top Outroutes	5	162	P
Top 20 Sessions	5	154	P
Drill Down by DPC, LIM, Port	5	162	P
Drill Down by DPC, LIM, Port, Session	8	101	P

* - Loading and Aggregation run in sequence, but both in parallel with queries.

Top queries, when using materialized aggregates, extract data from one table. Drill-downs perform a sequence of queries against many tables of aggregates.

For the tests not using materialized views, the set of the queries used was the same, but the data was aggregated by each query, every time it was run.

To gather the processing and response times while comparing the different alternatives, they were run by processing the equivalent to one day of the session statistics, which is equivalent to 96 snapshots. Loading and aggregation were run every 8 minutes, instead of 15 to accelerate the collection of statistics. This operation makes it harder than the system requirements, meaning that for a real situation the results are expected to be even better, since queries running together with the loading and aggregation processes happen less frequently.

Since the running frequency for the processes and the many queries are different, more response times for those queries running more often were gathered and used.

An average of 63 queries were run every 15 minutes, which means 4.2 single queries per minute.

Chapter 8

Options and Alternatives Analysis

Three basic alternatives were defined to compare their performance. Variations of these alternatives were tested, and all the test results are presented here.

The following is a description of main characteristics of the three alternatives, which differ mainly by the type of tables used or by the combination of tables with logging or nologging:

❖ Alternative 1

- Flat Relational Tables, which are referred to as Flat Tables.
- LOGGING – on the aggregate tables. This database option indicates that objects creation and transactions against these objects will be logged in the redo log files. Logging is useful mainly to undo transactions not committed to the database, and to redo transactions lost after a backup, made before these transactions were run, is restored over the database to fix a problem.

❖ Alternative 2

- Index Organized Tables – also known as Index-Only Tables or IOT. These tables are a new type of tables provided by Oracle8™, having very distinctive characteristics. They keep their data sorted by primary key; all data is stored in the index as a standard Oracle™ B*-tree; no additional indices can be created (only primary key unique indices are

allowed). IOT requires a primary key constraint, and pseudo column RowID cannot be selected. As indices are not kept separately, this does happen for flat tables, disk space is saved, mainly when a good part of the rows is part of the primary key.

- LOGGING – is mandatory because the nologging option is not yet available for this type of table.

❖ Alternative 3

- Flat Relational Tables.
- NOLOGGING - Since restoring aggregate tables updates through log files usually would be less efficient than re-aggregating them from the base data stored in the sessionstats table; aggregate tables do not use logging. The sessionstats uses logging because it receives the data coming from the network, which is not recoverable from anywhere else, the aggregate tables use nologging. This way, it was assured that no unrecoverable data is lost, while reducing database overhead.

8.1 Best Versus Worst Case

Later in this chapter, graphical results of the tests will be used to prove, why the Alternative 3 was found to be the best one. To make a best-versus-worst case comparison, an alternative considered the worst was tested with

- Flat tables.
- Primary keys not clustered.
- Snapshot id as date type.
- Logging.

- Sessionstats without primary key.

The results are in the graphs of Figure 8.1. The meaning of some of these options, not explained until now, will soon be described.

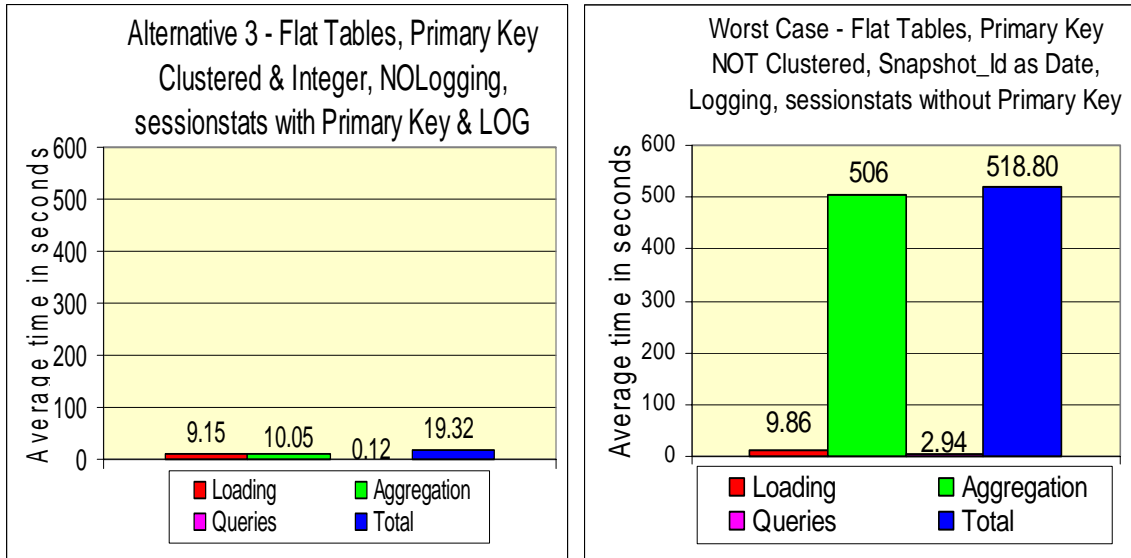


Figure 8.1(a) – Averages of Best (left) versus worst case (right)

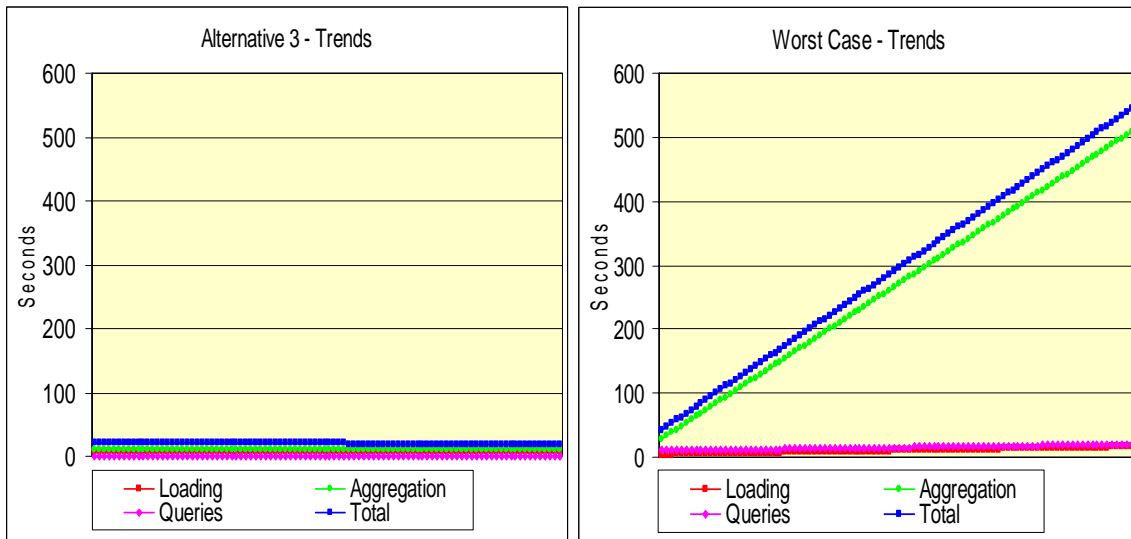


Figure 8.1(b) – Trend lines of best (left) versus worst case (right)

In Figure 8.1(a) the average times used to perform the loading and aggregation of 96 snapshots, the average query time of the set of queries,

previously defined, and the averages total is displayed. At the left side of Figure 8.1 (a) and (b), the graphs of the best alternative with the worst at the right are shown. The trend charts clearly show that while the average times of the best alternative tend to remain constant, the average times of the worst alternative keep degrading steadily at a high rate.

8.2 Alternatives Standard Options

Some of the tested options were incorporated by all alternatives, once they proved to be the best, to better compare the performance of the different characteristics. The tests of some of the incorporated options are, most of the times, presented as a variation of the same alternative to more clearly show the difference in performance. The results presented in this section, except when otherwise stated, were collected from 288 processed snapshots with approximately 1,350 sessions each. Examples of options incorporated by all alternatives are

- Definition of attribute precision as NUMBER (12,3), instead of simply as FLOAT with 126 digits of precision – the default – was one of the first changes applied to all aggregate tables, as well as to the table sessionstats, which had its attributes defined as NUMBER (10,3). NUMBER data type variables defined with this specific precision are still floating-point variables, although with less accuracy, which takes less disk storage and computer memory and requires also less computing power to operate them. NUMBER (12,3) means the attribute has 12 digits of precision with three of them being decimal places. This precision is considered good enough not to lose network

management data accuracy. The precision reduces memory and storage requirements by approximately half. The difference in performance was very clear when testing the database with a configuration having one session per port, which is not close to reality. Anyway, with a configuration of eight sessions per port, more similar to the reality, it was observed, as shown on Figure 8.2, a longer aggregation time, with the loading and aggregation time clearly going up, affecting the scalability of the system. The charts in Figure 8.2 compare the Alternative 3 using the default FLOAT precision for aggregate tables attributes, and sessionstats table, with NUMBER (12,3) precision for aggregate tables, and NUMBER (10,3) for sessionstats table.

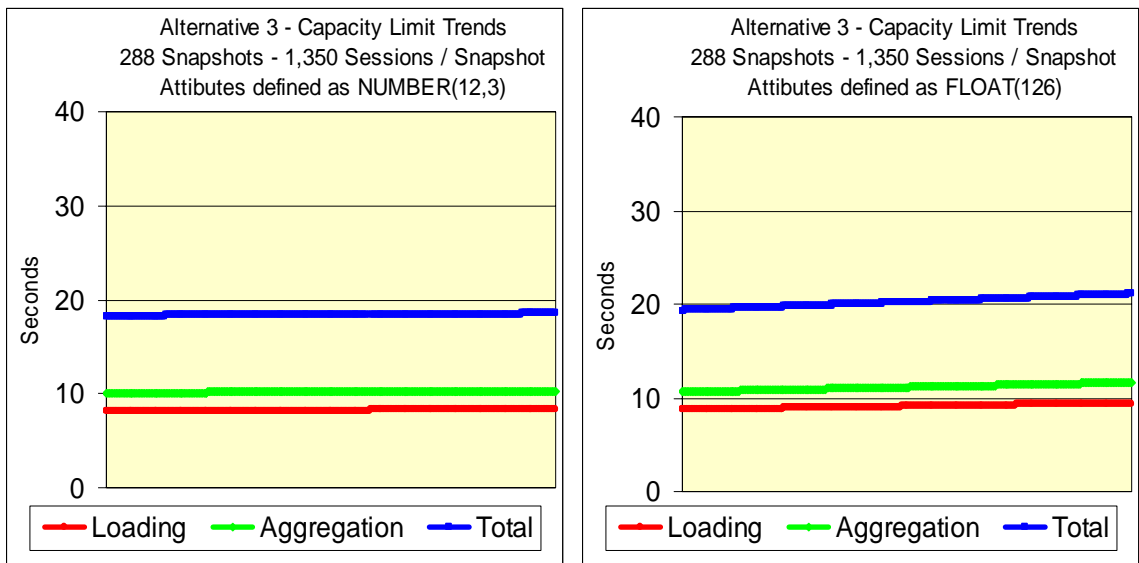


Figure 8.2 – Performance of Alternative 3 with attributes defined as NUMBER(12,3) (left) and the default FLOAT (126 digits) (right)

- Serial processing of aggregation for local and then remote network elements – was defined also as a standard option for all three alternatives, but it was tested running in parallel. To perform the tests, the aggregation

process was broken in two, one performing the aggregation by local network elements containment, and the other performing the aggregation by network containment of the remote network elements. At the end, the local and remote results by network are consolidated. Once more, it was clearer on the preliminary tests with a network configuration with one session per port that performing local and remote aggregation in parallel was much worse than when done serially. As it happened on the parallel loading and aggregation tests, what is still clear on the tests with the last network configuration with eight sessions per port is the steadily growing loading and aggregation time. This demonstrates that doing local and remote aggregation in parallel does not scale well. These tests were also run as part of a set of tests to determine the maximum capacity of the proposed solution. The charts in Figure 8.3 exhibit the results of the tests.

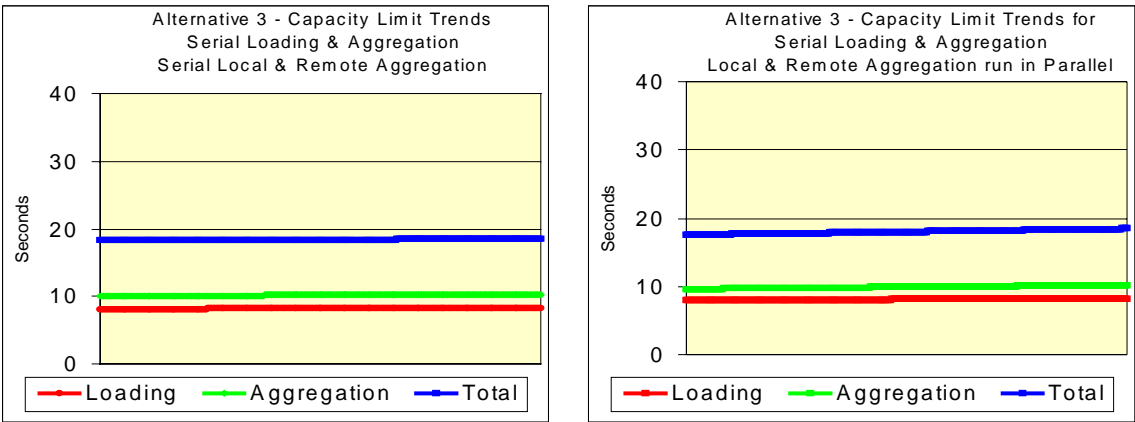


Figure 8.3 – Trend lines of serial local and remote aggregation (left) versus parallel local and remote aggregation processing (right)

- Serial processing of loading and aggregation – this is expected to happen on a daily basis. Loading and aggregation were tested in parallel by

having loading and aggregation running continuously without interval. This procedure is part of the tests to determine the capacity limits of the system. Again, the results were not as drastically different as the ones observed in the tests with the network configuration having one session per port. Therefore, the loading and aggregation time could be observed growing uniformly, again demonstrating that this processing would not scale as well, if such processing were possible. Figure 8.4 presents the results of this operation. The total processing time for serial processing is slightly higher, due to introduced delays to synchronize the loading and aggregation processes.

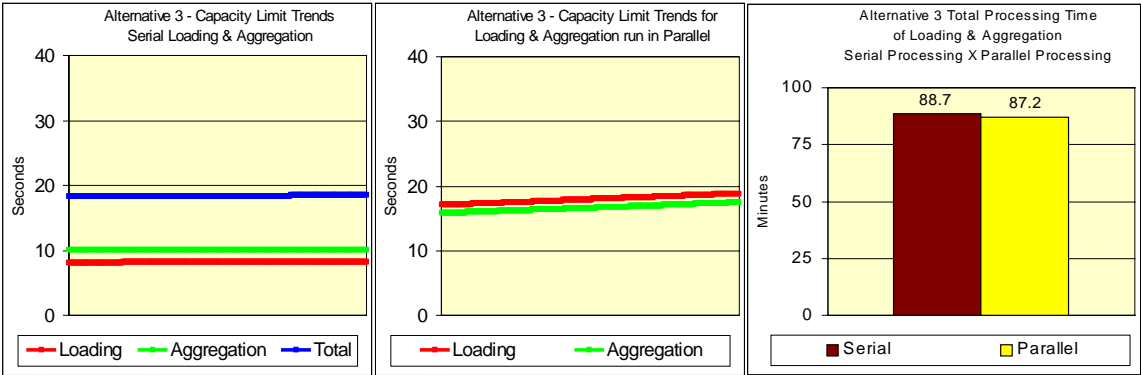


Figure 8.4 – Serial loading and aggregation results (left), parallel loading and aggregation (middle), and total processing time (right)

- Clustered indices – after the initial tests, it was observed that the aggregation time was growing as the number of rows per table was going up. These results were showing that the indices were not used or were not effective. Upon aggregation process and data analysis, it was concluded that placing the snapshot id as the first column of the primary keys of the sessionstats and aggregate tables should make the difference. This

conclusion was impressive because it was known that when the aggregation is done by network containment, it always processes the rows of one specific snapshot. Therefore, placing this column as the first column of the index, clusters all the data normally used during aggregation by network containment, reading and writing it together. This procedure makes the indices updates also easier. Figure 8.5 depicts the difference of the indices tree structure, for both situations. In Figure 8.5(a), there is the non-clustered indices, and in Figure 8.5(b), the clustered indices are shown.

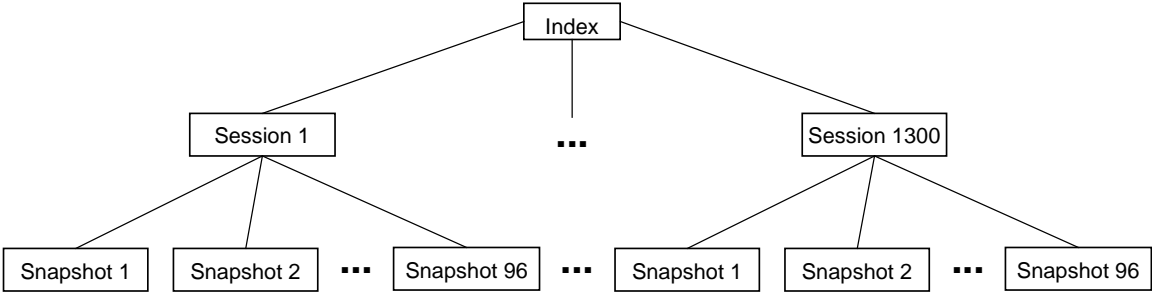


Figure 8.5(a) – Non-clustered indices

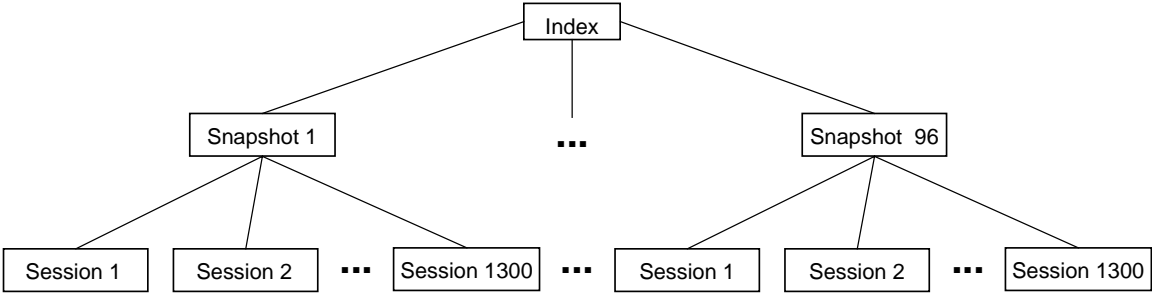


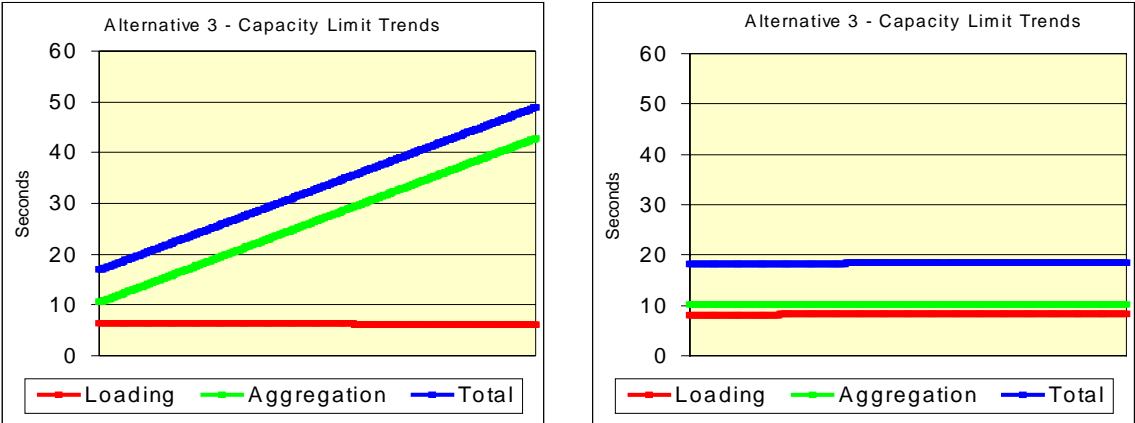
Figure 8.5(b) – Indices clustered by snapshot

- Snapshot id as integer – was another option, which was also incorporated by the three alternatives, instead of using a database date type field. On

the best-versus-worst case that is presented in Figure 8.1, this option was one of those used by the worst case. What makes it so unfavorable, is first, its size (a long character type field), and second, the frequent conversions required to compare it to the desired snapshot id, during the aggregation of a specific snapshot. The snapshot id used is not only shorter, nine digits, but is also an integer, which performs better on comparison operations. Besides, this integer is still a date, the date the snapshot loading into the database started in seconds, although as said it is not defined on the tables as a date-type, but as an integer. The date, using a date-type field, although not used by the aggregation process to identify the rows of a specific snapshot to be aggregated, was maintained because it is required for this purpose by a new package provided with Oracle8™, called time series cartridges. This package can be used to perform temporal aggregations. Meanwhile, it was determined by this research that time series cartridges package has performance problems to accomplish massive data temporal aggregation using views and views materialization. It performs reasonably well, aggregating one attribute at a time of a limited number of rows, so it should be restricted to perform *ad hoc* queries. Test results about this investigation are presented later on this chapter.

- Sessionstats table with primary key – was clear after the initial tests that the definition of a primary key, and consequently the creation of indices was increasing the loading time by thirty percent or approximately three

seconds. The increase in time was due to updates of the indices as the insertion of session statistics was made on the table. Meanwhile, it was later found that without indices on the sessionstats table, the average aggregation time for the first day was, on average, six seconds higher, and growing as the number of rows of the table sessionstats grew. This led the researcher to the decision of incorporating the primary key for the table sessionstats in all alternatives. Figure 8.6 shows the difference of



performance of the Alternative 3 by using or not using a primary key on the table sessionstats.

Figure 8.6 – Alternative 3 loading and aggregation performance differences when sessionstats table has no primary defined (left) and when it has a primary key defined (right)

- Flat Relational Tables for networks configuration data – is another option adopted as standard. When fully realized, the strengths and weaknesses of Index-Only Tables (IOT) they were deemed perfect for storing the network configuration data. Surprisingly, the actual test results did not confirm this fact. Using IOT tables to store the network configuration is in

fact clearly unfavorable. Figure 8.7 depicts the actual difference in performance of Alternative 3, using the two types of tables.

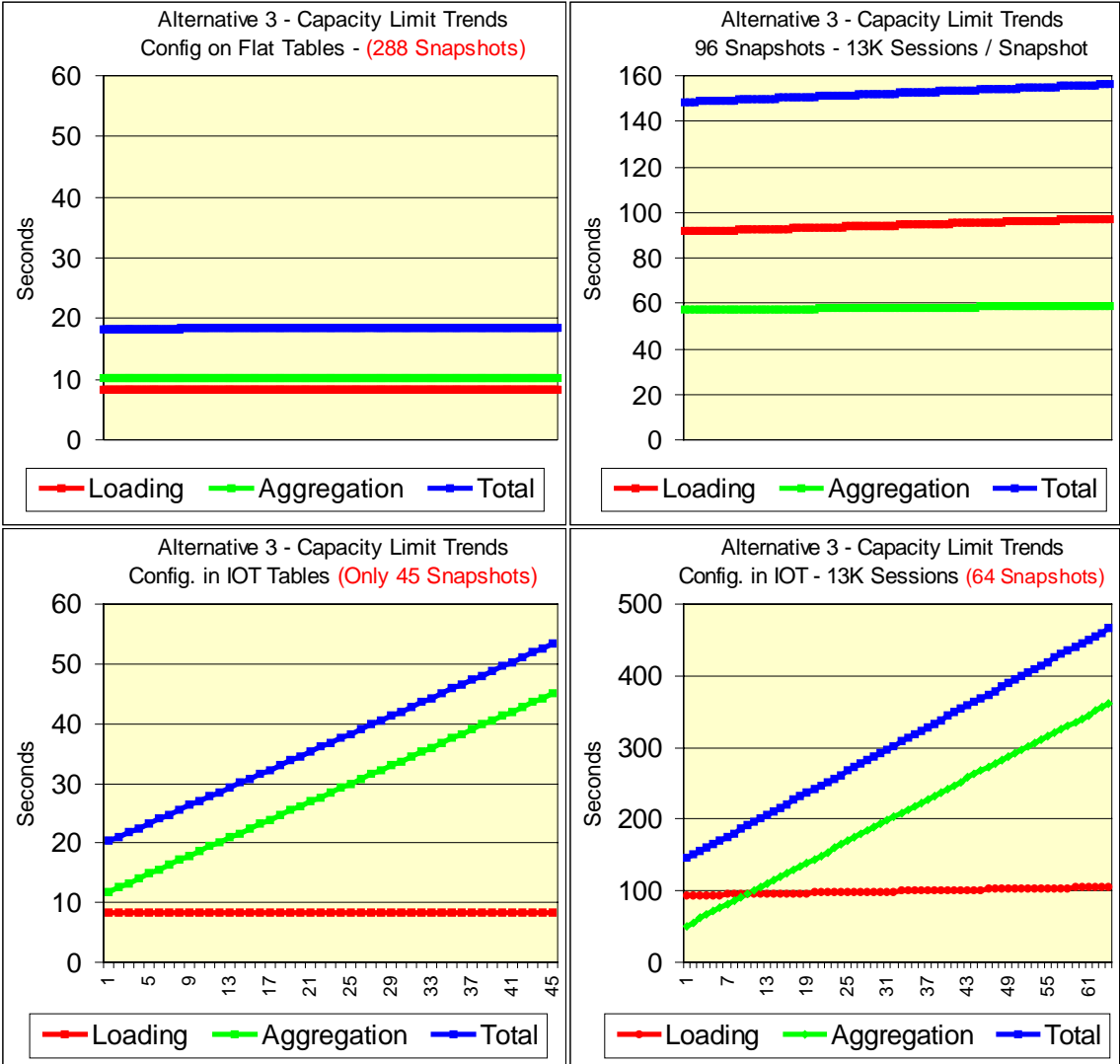


Figure 8.7 – Loading and aggregation times using Flat Tables for networks configuration data in the top two charts, and using IOT tables in the two charts at the bottom

On the left side of the figure, there is the test results for snapshots with 1,350 sessions per snapshot and on the right side for 13,500 sessions per snapshot. Not only is the performance poor when IOT tables are used to store network configuration, but it also does not scale well because the

aggregation times keep on growing strongly. The trend was so clear and steady that there was no need to run these tests for the usual 288 snapshots. On the top right side, a slight increase in aggregation and loading time can be observed for the standard Alternative 3, when processing snapshots with approximately 13,500 sessions. But it is important to realize another important result this same chart shows: that is, although there are ten times more sessions per snapshot, the total processing time is not ten times the first. The scalability of Alternative 3 is therefore proven.

- Aggregate tables without foreign keys – was also adopted as standard, once their definition would not add value to the system, performing redundant checking repeatedly.

8.3 Alternatives Comparison for the Actual Load

Alternative comparison is threefold: Loading process, aggregation process performance, along with queries response time. In Figure 8.8, the top graphs show the average processing and response times, as well as the averages total. The lower charts of this figure show the trend lines of the same data. These lines indicate the solution scalability, and it is clear that Alternative 3 is better than Alternatives 1 and 2. On the graphics seen before, to determine loading and aggregation capacity limits, where no queries were running, Alternative 1 seems to have a similar performance, if not better than Alternative 3. However, here, it is easy to see how much better Alternative 3 is, than Alternative 1. The averages, totaling 19.32 seconds in Alternative 3, compared with the 37.73 seconds of

Alternative 1, says it all. However, this observation is worthwhile, showing that the aggregation time for Alternative 1, with queries running in parallel, is almost three times the aggregation time of Alternative 3. Alternative 1 queries processing time is also fifty percent longer. As can be noticed, Alternative 2 has a comparable performance to Alternative 1 for loading and aggregation, but increasingly worse on queries response time. As Alternative 2 has no data separate from indices, it has the best loading performance of the three alternatives. Alternative 3 is what is proposed as a solution for network management data aggregation by network containment.

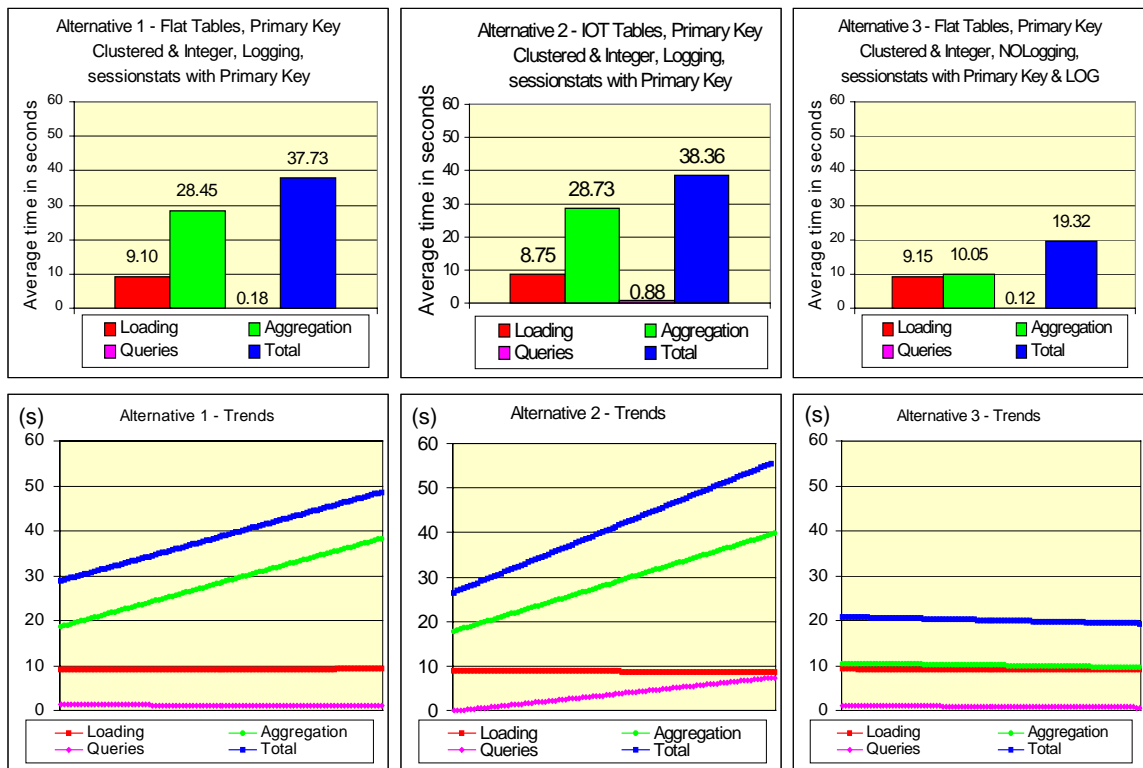


Figure 8.8 – Alternative 1, 2, and 3, loading, aggregation, and queries averages, as well as averages total at the top; plus the trend lines at the bottom charts for the same data

The statistics presented in this section were gathered by loading and aggregating 96 snapshots, while running the set of queries presented in Table 7.1 in Chapter 7. That table also shows the frequency and number of queries run to assemble the results presented in Figure 8.8. Table 8.1 shows response times of processes and queries. Only the response times for a subset of the query set is shown. Response times of the queries by remote network elements, as the configuration used is symmetric, are similar. This reason reflects why it is not worth showing both. The overall average was obtained by adding the loading, aggregation, and query average time dividing the total by 3.

Table 8.1 – Loading, aggregation, and query average response time

Process or query name	Average processing times in seconds		
	Alternative 1	Alternative 2	Alternative 3
Loading	9.10	8.75	9.15
Aggregation	28.45	28.73	10.05
Queries:			
Top DPCs	0.00	0.03	0.01
Top LIMs	0.05	0.44	0.02
Top Ports	0.04	1.03	0.03
Top Sessions	0.59	2.68	0.44
Top Inroutes	0.04	0.43	0.03
Drill Down by DPC, LIM, Port	0.04	0.78	0.01
Drill Down by DPC, LIM, Port, Session	0.50	0.78	0.29
Overall Average	12.58	12.79	6.44

8.4 Comparison of Alternatives Limits

The capacity limits of each alternative were determined, based on their hourly loading and aggregation capacity. Loading is measured in megabytes per hour and aggregation in session statistics per hour. No queries were run in parallel, and there were no intervals between runs. The test results are based on 288 snapshots processing, each one having 1,350 session statistics to aggregate.

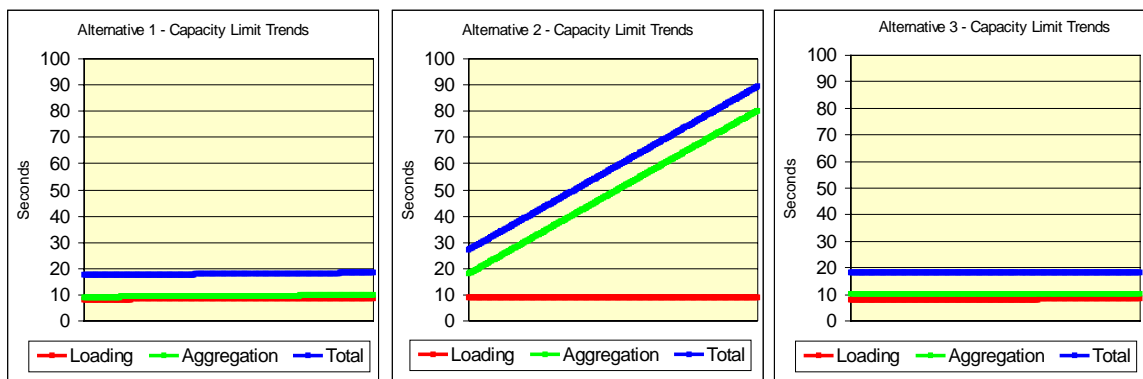


Figure 8.9 – Loading plus aggregation capacity limits of the configuration used, for Alternative 1 (left), Alternative 2 (center), and Alternative 3 (right)

As already mentioned, Alternative 1, with logging, performs slightly better than Alternative 3, but only when no queries are run in parallel. Alternative 3, as can be seen, scales slightly better as aggregation time of Alternative 1 grows faster. Alternative 2, using IOT tables, does not perform well. The analysis of the reasons why IOT tables do not perform well is in the next section. Table 8.2 presents the numbers related to the statistics used to produce Figure 8.9 charts. On Table 8.2, the first column has the process type and the corresponding number of the Alternatives (1 to 3), column two has the average processing time per snapshot, in column three there is the capacity limit for each option. Loading capacity is expressed in megabytes per hour, and aggregation capacity is

measured in sessions per hour. The number of snapshots processed and the total number of session statistics processed comes next. The last two columns indicate the type of tables used by each alternative, Flat (F) or IOT (I), and whether LOGGING or NOLOGGING is used. *N, for Nologging means, aggregate tables were defined with the NOLOGGING option, and sessionstats, as well as the configuration tables are defined with LOGGING.

Table 8.2 - Maximum capacity - no intervals between runs, no queries, serial loading, and aggregation processing

Process type and Alternative number	Average time per snapshot (secs)	MB or sessions per Hour	Number of snapshots	Total number of sessions processed	Flat or IOT tables	LOG Or NOLog
Loading 1	8.43	62MB	288	387,089	F	L
Aggregation 1	9.55	269,110	288	387,089	F	L
Loading 2	9.17	19MB	288	387,089	I	L
Aggregation 2	49.14	82,980	288	387,089	I	L
Loading 3	8.23	61MB	288	387,089	F	N*
Aggregation 3	10.15	263,254	288	387,089	F	N*

The capacity limits and aggregation rates achieved for the similar to actual load are much higher than the minimum requirements of the project. The final chapter will present the requirements compared to the achievements.

8.5 Alternative 3 Tests of Scale

To validate the capability of the proposed solution of handling network statistics of tens of thousands or hundreds of thousands of sessions, Alternative 3 was run with a network configuration of 13,500 sessions, and the corresponding network elements, as well as for a network configuration with

135,000 sessions. As can be verified by the charts of Figure 8.10, the solution not only scales well but also scales with economies of scale from the network with a configuration for 1,350 sessions to the network with 13,500 sessions. The economies of scale occur because on the bigger network configuration, there are relatively more sessions per LIM, DPC, and network. From the configuration with 13,500 sessions to the configuration with 135,000 sessions, the loading and aggregation process times are slightly higher than ten times, but less than one hundred times the configuration with 1,350 sessions. Therefore, as can be seen in the graphs, the total processing time does not go up significantly as the snapshot session statistics are loaded, the aggregates computed and inserted into the database. The loading and aggregation tests with 135,000 sessions per snapshot were run until the database disk space was completely full, degrading the performance. Consequently, the results, as can be seen, were good. This indisputably proves the scalability of the proposed solution. Figure 8.10, on the top, has graphs showing loading average in KB/h (Kilobytes per hour), computation of aggregates in S/h (sessions per hour), and the total number of sessions processed to derive these statistics for network configurations with 1,350, 13,500, and 135,000 sessions, from left to right. A trend line connecting the number of handled sessions per hour gives a perspective of the scalability. The bottom charts of Figure 8.10 show for the three configurations, the loading, aggregation, and total trend lines. Due to limits of disk space, the number of snapshots processed for each configuration was different. From left to right, 288 snapshots for the network configuration with 1,350 sessions, 96 snapshots for

the network configuration with 13,500 sessions (in the middle), and only 11 snapshots for the configuration with 135,000 sessions.

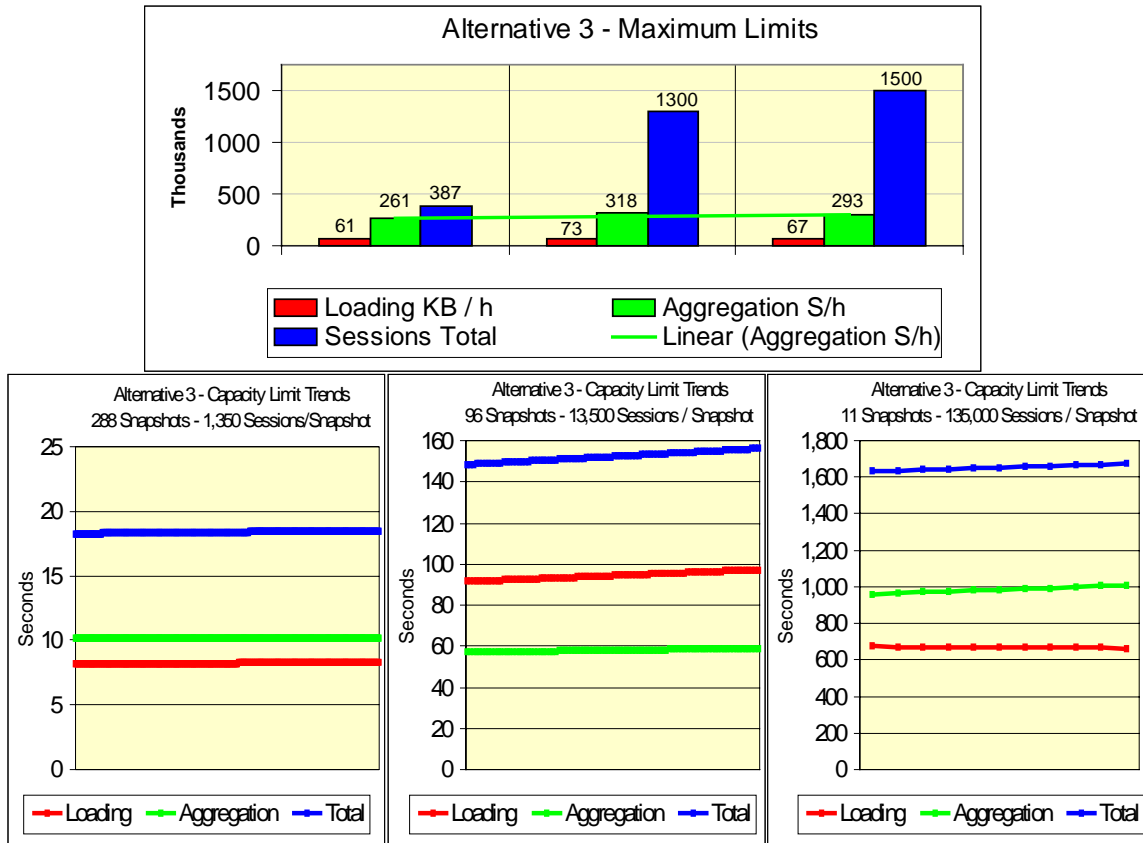


Figure 8.10 – Scalability of the proposed solution, from left to right network configurations have 1,350, 13,500, and 135,000 sessions

8.6 Index-Only Tables Performance

Disregarding the fact that performance of IOT tables was surprisingly poor, it can be explained. What can be concluded is that they do not perform well in the problem to be solved. Besides expecting that their performance may be improved in the future, what must be understood that explains the inferior performance compared to flat tables is

- As indices and data are stored together, a smaller quantity of index data is stored per disk block.
- The aggregation process depends on indices to locate the snapshot session statistics to be aggregated.
- Queries also need to use indices to locate the data to be read.
- In addition, data and indices are stored together. When indices are needed alone, as in the two frequent cases mentioned above, more disk blocks must be read to retrieve the same amount of indices.
- Requiring more disk I/O to retrieve the same amount of index data is disastrous because disk I/O is the resource taking more time, consequently, causing the biggest performance degradation in the system. It is imperious to reduce I/O to the minimum to reach the best performance of this system.
- Finally, as the data is stored in index format, the table rows have to be reconstructed from the indices when they are read, causing additional overhead.

Following a careful analysis and after observing the results presented in Figure 8.7, shows the performance results of IOT tables storing networks configuration data, it was concluded that IOT tables are definitely not a good fit for this type of system with this kind of application.

8.7 Temporal Aggregation

Temporal aggregation performance is not as critical as the performance of aggregation by network containment, which aggregates statistics used in continuously monitoring the network traffic and performance. It is not as critical because temporal aggregation is expected to run sparsely. Once a day, or even once a week to perform aggregation by day, once a month to perform monthly aggregation, and so on. This is very different than running it every fifteen minutes or less, as it needs to be done for network containment aggregation.

Anyway, tests were run to perform aggregation by day using standard SQL and time series cartridges. The option using standard SQL functions used standard sum, and average functions plus group-by year, month, and day, which were defined as independent fields. Time series cartridges used scale-up functions of date type fields, plus sum and average functions. Both alternatives, of course, used also network element ids to accomplish the temporal aggregation by each network element or network. The same techniques used to perform the aggregation by day, can be used to perform aggregation by the hour, month, and year.

Six days of aggregates (576 snapshots with 1,350 sessions each) were used to perform the computation and materialization of daily aggregates. The daily aggregates of each network element class were derived from the aggregates computed by the aggregation by network containment, of the same network element class. It was observed that the aggregation time for the option using

standard SQL grows linearly with the quantity of aggregated rows, while the aggregation time for the option using time series cartridges grows exponentially.

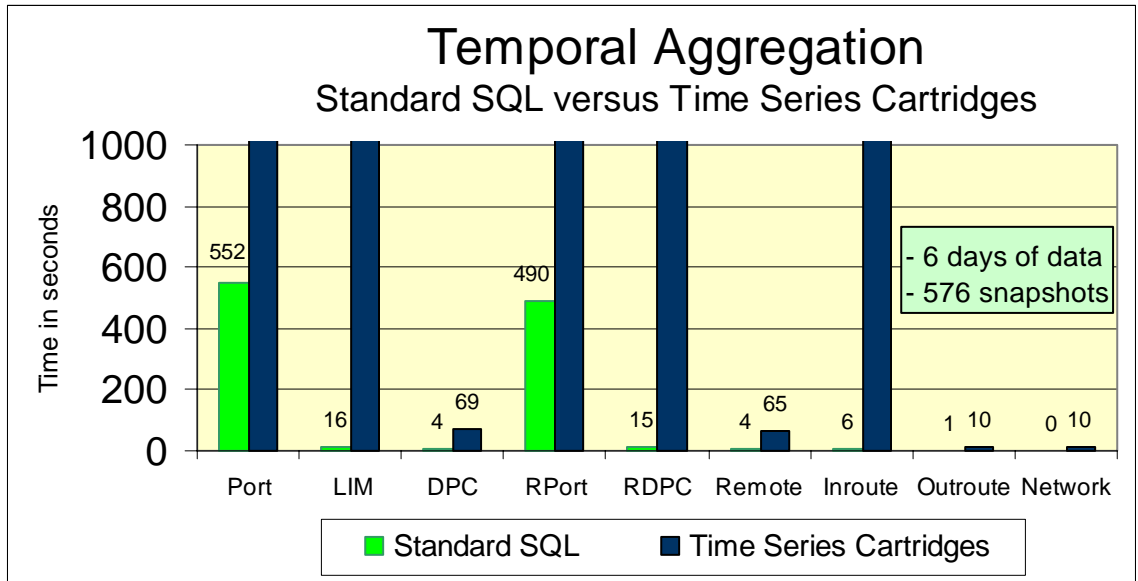


Figure 8.11 – Temporal aggregation of six days of network aggregates using standard SQL functions and time series cartridges

The aggregation by port, LIM, remote port, and remote LIM, and inroutes, using time series cartridges, aggregating six days of data would never end. It would require much more memory than the 128MB of RAM available on the testing system configuration. The aggregation option using standard SQL, performs the aggregation by day of the six days of data, in approximately eighteen minutes. If ran daily it will not take more than three minutes to perform the daily aggregation of attributes by port, LIM, DPC, remote port, remote DPC, remote, inroute, outroute and network. If it is decided to run the process weekly it is suggested to break it in three, local, remote, and inroute, outroute, and network aggregation. Once it is broken in three, each one should be run at the end of the aggregation by network containment process. In this manner, the temporal aggregation will

not compete for resources with aggregation by network containment, running in between them. Figure 8.11 shows some of the results. The columns going over 1,000 represent those aggregations that would never end. They were run for more than twelve hours never ending.

It is obvious that, for big quantities of data, which is common in this kind of application, using a schema that allows the use of standard SQL functions to perform temporal aggregation is the only feasible solution.

Chapter 9

Eager versus Lazy Aggregates Computation

Eager and lazy aggregates computation strategy is a technique introduced and discussed by many specialized papers, some of which are mentioned in this report and listed in the references. These two techniques are related to another popular organization for summary data, and data warehouses, the data cube. Presenting the aggregation dimensions, this work followed the popular data cube multidimensional structure, which contains at each point an aggregate value. Data cubes use a hierarchical organization, and the aggregates on the lower levels of the hierarchy are used to compute aggregates with coarser granularity at the higher levels.

It is to compute higher level aggregates with coarser granularity that one of the two strategies, eager or lazy, has to be chosen. These strategies use different approaches to compute the coarser aggregates.

The lazy strategy does not pre-compute aggregates, deriving them from the cube base data, in this case sessions statistics, only when the aggregates are referenced by a user. This means that if the same data is referenced by many users or by the same user many times, the same reads, computations, writes are repeatedly performed. While no disk space is permanently used to materialize aggregates, once the results are not saved for future use, the base data, with finer granularity and consequently bigger volume, has to be maintained. It has to be maintained to generate the coarser aggregates as they

are requested, and to keep the only existing information available about network traffic and resources utilization, which often cannot be done for a period big enough to satisfy, namely, network planning activities.

The eager strategy uses a different approach, pre-computing all the aggregates up-front, materializing them by inserting them into real tables saved onto a disk. This means the data is aggregated once for all operators, network managers, and planners to use. When this strategy is deployed, the queries against the higher network levels are very light and fast because the bulk of the work was done in advance. The results obtained by this research prove that, on average, the queries are eight times faster when using the eager-computation-of-aggregates approach. Another very important aspect is that once the eager strategy is used, there is no need to keep the base data to derive the coarser aggregates, once they are already saved on disk. This leads to disk storage savings with the ability to keep network management data, covering longer periods.

The focus of this research was to prove, as was proven, that the eager computation of aggregates can be used to derive the data cube for network management data, using commercially available software. This project focused on the practical aspects of the data cube implementation to materialize and store network management data. It is realized that the materialization of the data cube with eleven different planes, or levels, is completed in about ten seconds. It takes less than twenty seconds to load and compute all the aggregates, using an eager

strategy. This achievement makes it possible to gather network statistics often, providing frequent updates by any of the eleven defined dimensions.

The bottom line is an efficient sharing of resources, and resources sharing implies savings by reusing materialized aggregates. By sharing pre-processed information, the reduced competition for resources leaves more system power available, increasing the overall system capacity.

As network operations do not allow analysis of all the fine-grain data generated by a big network, the eager approach provides the means to easily control the higher network levels without losing the ability to drill-down and perform a detailed analysis, when a problem or abnormality is detected on higher levels.

A lazy approach requires the aggregation all the way up to control the higher network levels, partly repeating the same aggregation operations to perform a drill-down analysis, when a problem is detected.

Materialized views make queries processing simpler, faster, and lighter, saving resources while optimizing network operations, management, and planning. Figure 9.1 presents the results of comparing performance using eager and lazy strategy for aggregates computation. On the top left side, there is the averages of loading, aggregation, and queries of the eager approach. The chart on the bottom left shows the average loading, and query time with the aggregation time being zero for the lazy approach. A total of the averages is shown on both graphs. On the right side, Figure 9.1 shows the total loading, aggregation, and query time for a typical fifteen minutes period, as well as the

total processing time of all of them. It evidently shows, that even for a load as small as two queries of average complexity per minute, the total processing time for the eager approach on the top right graph is much smaller. The benefits of using an eager computation of aggregates only grows as the query rate increases.

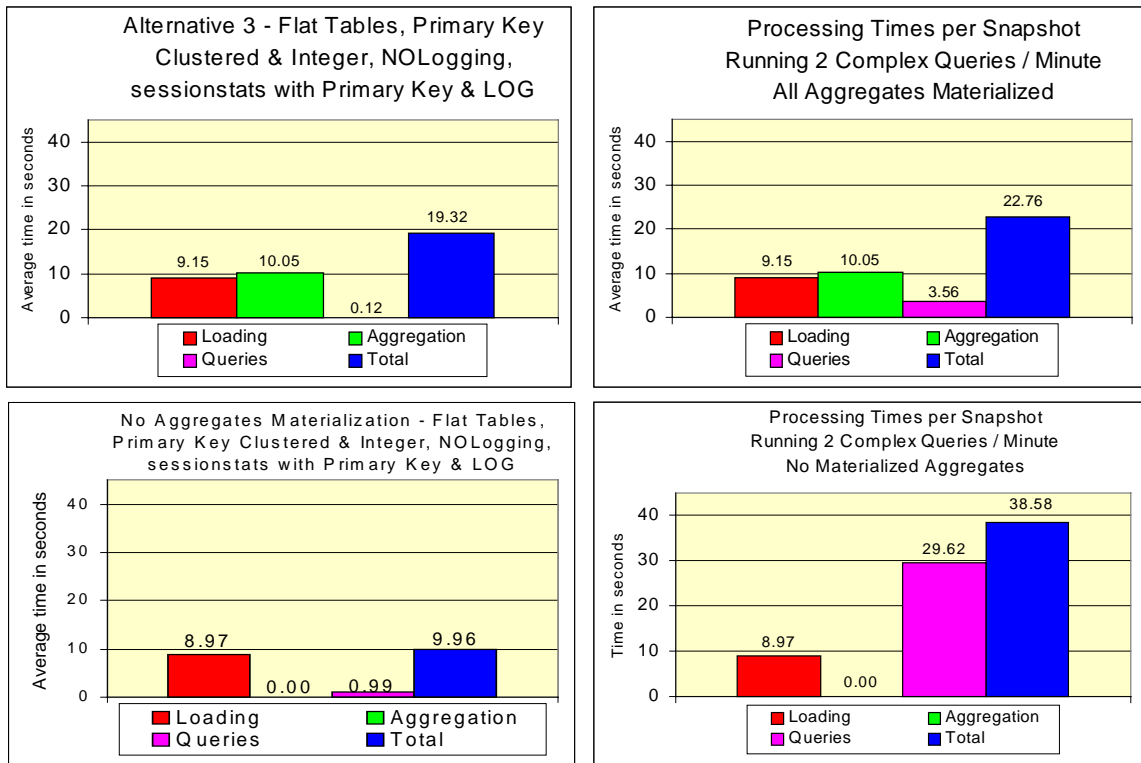


Figure 9.1 – Eager (top) versus lazy (bottom) aggregates computation

The results presented on Figure 9.1 were collected from processing 96 snapshots and a set of queries as defined on Table 7.1. The eager approach used materialized views, while the lazy approach performed all the aggregate computations every time a query was run. The query output generated was similar; while the loading time for the lazy option is slightly better, the query response time is eight times longer.

A crucial advantage of the eager strategy over the lazy approach is the capability provided by the eager strategy to implement easily an aging schema of historical data, which is one of the proposals here.

9.1 An Aging Schema for Historical Data

Network planning without historical data would be an inaccurate divining exercise, if possible at all. Long-term resources allocation cannot take advantage of existing idle resources without using historical data. Therefore, network planning, as well as an efficient long-term resources allocation, is highly dependent on historical data. Nevertheless, there is no way to keep all detailed network statistics related to a long period for a small to medium size network. To solve the trade-off between the data that can be stored versus the data needed an aging schema is proposed. The aging schema may assume different forms, covering different periods with varied data granularity for each schema. The aging schema ultimately will depend on the needs of the institution implementing it. The initial aging schema, presented here, may cover very long periods. The top network level statistics, that are equivalent to the top planes of the cube, have coarser granularity. The lower and lower levels have finer and finer data granularity covering shorter and shorter life spans. However, some managers may need statistics with coarser granularity and longer life spans for the low-level network elements on this research case ports and sessions. An aging schema can address this and other needs. Any aging schema requires additional aggregation by the time dimension, which is expected to be accomplished once a

day or a week, plus once a month, etc.; in other words, the temporal aggregation need not to be done so often.

The proposed aging schema maintains

- Data on the base level of the cube, in this case, session statistics for a week. No further aggregation is required. Session statistics data is dropped afterwards.
- On the next level, by port, the data is aggregated daily, from the already-calculated aggregates (for every fifteen minutes) to hourly aggregates. These aggregates are then kept for three months.
- LIM aggregates, are aggregated once a day to daily granularity, from previously computed LIM aggregates and stored for one year.
- For the DPC level, the proposal is to use LIM daily aggregates to calculate monthly DPC aggregates, once a month, keeping them for five years.
- On the top level, the network level, monthly network aggregates are computed from monthly DPC aggregates, once a month, and stored for the life span of the network or more.
- Per snapshot aggregates at all levels are maintained for a week, then dropped.
- Please note that the researcher sometimes suggests using previous level aggregates to compute the coarser aggregates of the current level, to reduce the quantity of data read and processed.

This schema reduces the data to a minimum, while maintaining information on all levels with different granularities. This proposal fits those situations where,

after some time has passed, the network lower level detailed data loses importance. Figure 9.2 depicts the proposed aging schema, showing that although covering a shorter period, detailed data is more voluminous.

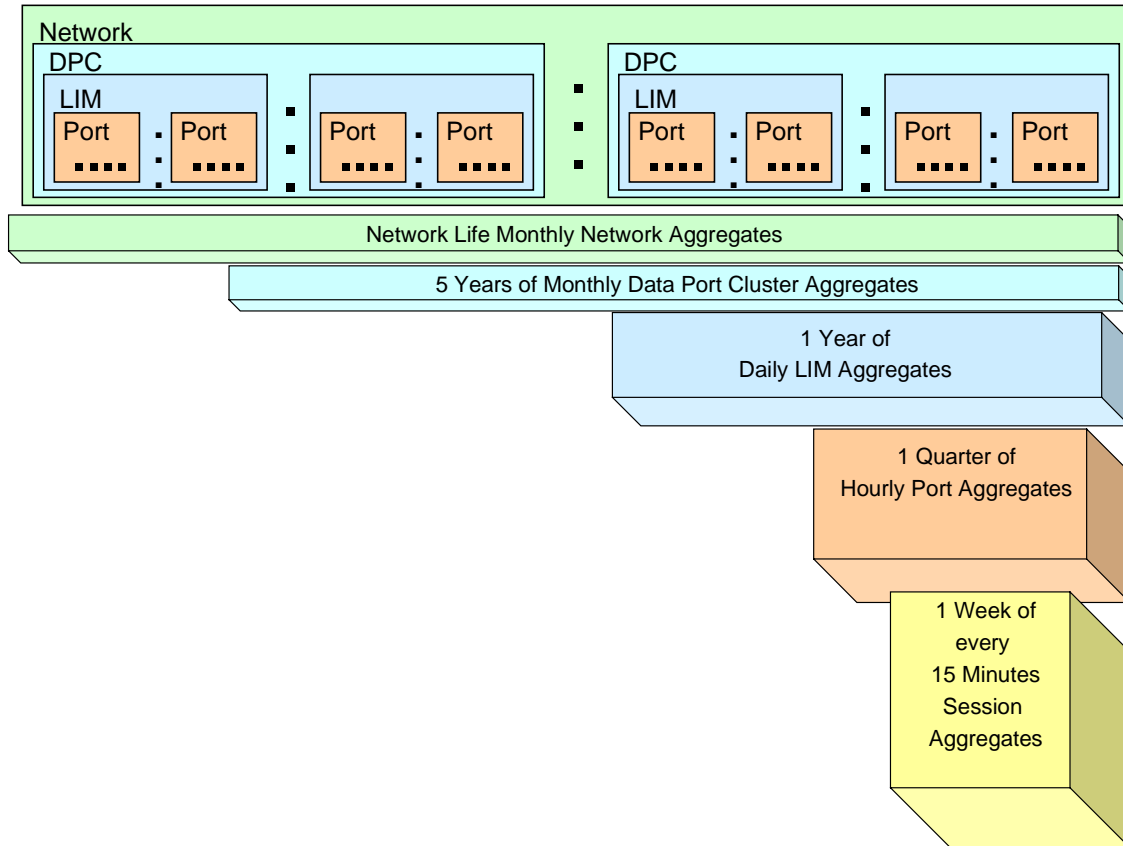


Figure 9.2 – Proposed network management data aging schema

Depending on the requirements, successive temporal aggregations can be performed at all levels, if it is important to keep a historic record for the lower network elements also. This option has to be treated in a case-by-case basis; nevertheless, the suggestion here is to keep not hourly aggregates but daily aggregates minimum, average, maximum, and the variance. This method is a sure-fire approach to avoid heavy processing and at the same time, bulky data storage.

Chapter 10

Conclusions and Future Work

The competition to provide network services is becoming fiercer among telecommunications businesses with larger and larger investments. Resource allocation of high bandwidth networks, as well as network planning and management, must be further developed. This research proves that data cube techniques, using views and views materialization, can be used to compute, manipulate, and efficiently store current and historical network aggregates, which are of primary importance to achieve higher stages of development in these areas.

It is also proven that using an eager approach for computation of aggregates is a better strategy for this kind of application than the lazy approach, and this can be done using Oracle8™ database software.

Finally, this study demonstrates that the proposed solution of using flat relational tables, primary keys clustered on sessionstats, and aggregate tables; furthermore, using snapshot ids as integers and nologging for the aggregate tables, not only is a solution, but it is a solution that performs and scales proficiently very well.

All the basic requirements of the project were met. A list of effectiveness measures follows in order to compare what was required to accomplished results.

10.1 Requirements and Achievements

EM1 - The system shall compute all aggregates at all eleven levels correctly and write them into the database for all the 1,300 sessions with 20 attributes each in fifteen minutes or less.

The system involves loading the session statistics and computing the aggregates correctly at all nine levels, for 1,300 sessions with 20 attributes each, and storing them into the database in 19.2 seconds. Figure 10.1 depicts goals versus achievements in a graphical format.

Sixty thousand sessions can be processed by this system in up to fifteen minutes.

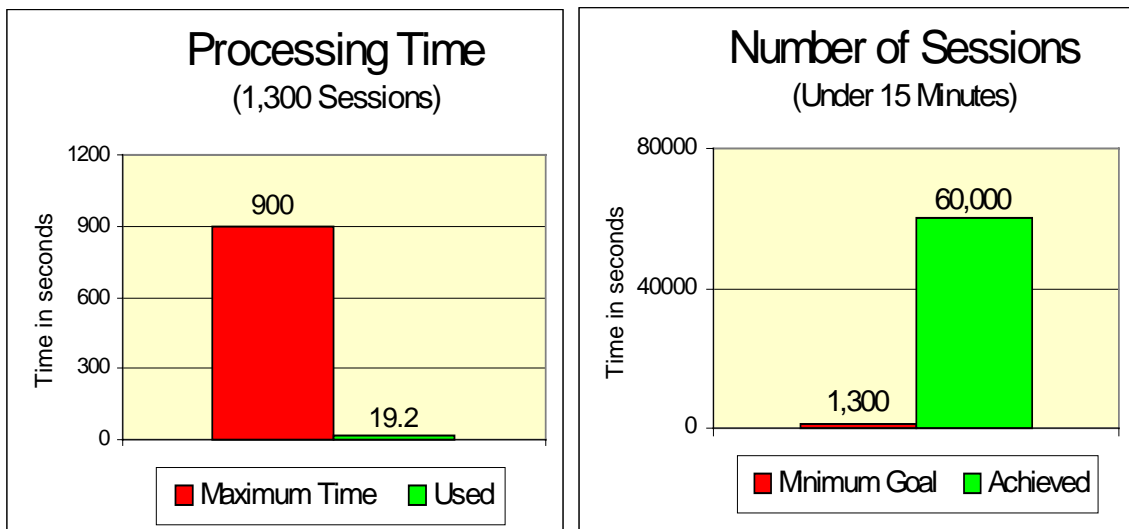


Figure 10.1 – In the left chart there is time available (in seconds), used, in the right it has processing requirements in number of sessions, for every fifteen minutes, and what was achieved

EM2 - The system shall store the data in a format that allows the graphical user interface to access them and provide updated reports or graphics every 15 minutes.

Data is stored in the database in a format readable by queries and is consequently readable by a graphical user interface. Updated reports and graphics may be provided every fifteen minutes.

EM3 - The proposed system tools shall provide means to query the data without requiring any programming.

All the database standard tools, including SQL, are available to query the data. Specifically, all the standard Oracle8™ tools plus times series cartridges are at hand.

EM4 - The system shall handle huge quantities (at least gigabytes) of current and historical statistical data.

Oracle8™ database can handle databases with terabytes of data. The tests prove that up to 61MB of statistics can be loaded and aggregated by the system in up to fifteen minutes.

EM5 – The system shall be more flexible, providing more tools and functionality than the current system.

The system has more flexibility than the current system and provides more tools, e.g., standard SQL tools, time series cartridges, and aging capabilities.

EM6 – The system must be user-friendly.

The system is simple and has the entire modern user-friendly tools provided by Oracle8™ software.

10.2 Potential Performance Improvements

Potential improvements of the presented results may be achieved by investing in the following areas:

❖ Disks:

- Disks with independent paths
- Faster disks
- More cache memory
- Disks exchangeable on the fly are strongly recommended for this type of application.

❖ Processing unit:

- More RAM memory
- More processing power.

❖ Database software:

- Reconfiguration, separating in different disks with different paths, data extents, software extents, and log
- Reconfigure the database to use a bigger database page size, which is highly recommended for data warehouse systems.

❖ With more powerful configurations, especially with more than one processor and multi-disk-paths, local and remote aggregation in parallel shall be considered.

10.3 Innovative Concepts and Future Work

It is clear to the writer that network planning needs are growing and increasing in importance. Therefore, what is proposed here is the creation of a new functional area on the ISO model for network management, and this area is network planning. Network planning will go beyond network management by performing network resources administration and allocation not for the next minute, not for the next hour, but for the next month, the next year, and years to come.

The proposed historical data model is a tool that makes it possible to gather data not only to be used in resource allocation but also to forecast their exhaustion and the need for more or to add more capacity to the existing resources. What has recently been often mentioned as a requirement of the modern network management policies is a proactive network management, to the extent possible, minimizing the requirements, urgencies, and emergencies of the reactive network management.

Some research was conducted to discover the best methods and techniques to perform temporal aggregation, but they need additional development to refine and complete the optimal aging schema. Part of the aging schema is deleting undesired data, which performance will depend on the model used. From the results observed during the course of this research, conclusions infer that the best strategy is to develop a model in which data cleaning is based on dropping tables of unnecessary data and creating new empty tables, rather than deleting specific rows from the tables, whenever possible.

Implementing a data cleaning schema based on table profiles defined in a table in the database, which would hold maintenance information such as how many snapshots or how long data should be maintained in each table, is probably a sound idea worth pursuing, implementing, and testing. Such a plan would support a self-maintained database for historical data. A spinning circular database with new data coming in and aged data going out would probably be a well-founded abstraction of such a database.

References

- [1] N. Roussopoulos. Materialized Views and Data Warehouses. In *SIGMOD Record, ACM Press, Volume 27, Number 1, 1998*.
- [2] I. Mumick, D. Quass, B. Mumick. Maintenance of Data Cubes and Summary Tables in a Warehouse. In *Proceedings of the 1997 ACM SIGMOD, International Conference of Management of Data, Tucson, Arizona, USA, 1997*.
- [3] D. Barbara, M. Sullivan. Quasi-Cubes: Exploiting Approximations in Multidimensional Databases. In *SIGMOD Record, ACM Press, Volume 26, Number 3, 1997*.
- [4] S. Guha, R. Rastogi, K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the 1998 ACM SIGMOD, International Conference on Data Management, Seattle, Washington, USA, 1998*.
- [5] C. Dyreson. Information Retrieval from an Incomplete Datacube. *Proceedings of the 22nd VLDB Conference Mumbai (Bombay), India, 1996*.
- [6] C. Dyreson. Using an Incomplete Data Cube as a Summary Data Sieve. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997*.

- [7] Y. Kotidis, N. Roussopoulos. An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. In *Proceedings of the 1998 ACM SIGMOD, International Conference on Management of Data, Seattle, Washington, USA, 1998*.
- [8] N. Roussopoulos, Y. Kotidis, M. Roussopoulos. Cubetree: Organization of and Bulk Incremental Updates on the Data Cube. In *Proceedings of the 1997 ACM SIGMOD, International Conference of Management of Data, Tucson, Arizona, USA, 1997*.
- [9] S. Chaudhuri, U. Dayal. An Overview of Data Warehousing and OLAP Technology. In *SIGMOD Record, ACM Press, Volume 26, Number 1, March 1997*.
- [10] G. Kock, K. Loney. Oracle8™ the Complete Reference. Osborne/McGraw Hill, 1997.
- [11] D. Oliver, T. Kelliher, J. Keegan Jr.. Engineering Complex Systems with Models and Objects. McGraw Hill, 1997.
- [12] A. Alomari. Oracle8 & Unix Performance Tuning. Prentice Hall PTR, 1999
- [13] Hughes Network Systems, Inc. - Integrated Satellite Business Network™ (ISBN™), LANAdvantage Reference Manual, Release 7.6B, 1993.
- [14] Hughes Network Systems, Inc. - Integrated Satellite Business Network™ (ISBN™), Statistics Report and Database Description, Revision B, 1991.

- [15] J. Valluri. Database Models and Architectures for Hybrid Network Management. Center for Satellite and Hybrid Communications Networks, University of Maryland, 1996.
- [16] P. Viswanathan. Automated Networks Fault Management. Center for Satellite and Hybrid Communications Networks, University of Maryland, 1996.
- [17] Oracle8™ Time Series Cartridges. Release 8.0.4 User's Guide. Oracle®. Oracle Corporation, 1997.
- [18] T. Connolly, C. Begg, A. Strachan. Database Systems – A Practical Approach to Design, Implementation, and Management. Addison-Wesley, 1998.