Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 53(67), Fascicola 1, 2008

Advanced production integration service using temporary tables and SQL optimization via neural networks

Vasile Corniţă, Rodica Strungaru, Sever Paşca[1]

Abstract- This paper presents an advanced production system integration techniques using neural networks for optimization purpose. One important aspect to consider when realizing the integration component between two or more systems is the data structure passing technique, taking into account specific system implementation issues like: data structures organization, storage, retrieval and dynamic requests. Nowadays there are many dedicated applications for specific business to consider, but when there is no such software application with all required functionalities; integration between existing applications should be considered.

Keywords: software application integration, SQL query optimization, neural networks, object oriented programming, database management system kernel, enterprise application integration, artificial intelligence.

I. INTRODUCTION

Nowadays, specific businesses has acquired production software applications, mostly when necessary and solving a particular set of necessities. In our globalization period, as business expands, is appear the need for geographically separated departments, associated business processes to be connected together.

As corporate dependence on technology has grown more complex and far reaching, the need for a method of integration disparate applications into a unified set of business process has emerged as a priority. After creating islands of automation through generations of technology, users and business managers are demanding that seamless bridges be built to join them. In effect they are demanding that ways be found to bind these applications into a single, unified enterprise application. The development of Enterprise Application Integration(EAI), which allow many of the stovepipe applications that exist today to share both processes and data, allow for an answer to this demand. [1]

From practice, when considering the integration between two software applications, let's denote these applications AppA and AppB, good results are obtained when the integration is carried out by the software company that produced either, AppA or AppB, because only one application data details are to be learned from scratch.

In order to integrate two software applications, two important aspects are to be taken into account:

- The data model: The exchanged data are in fact business documents and not simple character strings. It is highly probable that these documents (the two application documents) will contain lots of identical data, but it will not necessary be in the same format. A conversion job from one model to another is therefore needed.
- The communication system: In this context it is very important the communication protocol used to exchange data. Here will come to the role of middleware. A large part of software application integration is about the different technologies and techniques implementing this exchange. [2]

Typically, in internet contexts, the protocol used is HTTP (Hypertext Transfer Protocol).

Also data availability and data security policies have to be taken into account when integrating software applications.

[1] Faculty of Electronics, Telecommunications and Information Technology Applied Electronics and Information Engineering Department Bucharest - 1, Polizu street, nr.1, Building D, Floor 1, Room D110, emails: cornita_vasile@yahoo.com, rodica.strungaru@elmed.pub.ro, sever.pasca@elmed.pub.ro

The main purpose of the proposed production integration service is to provide with real time data from a production application that formulates a considerable number of SQL queries per second to a relational database, to the proprietary business management reporting tool application.

The integration service can be configured to start manually or automatically, and, in case of a system failure the service is configured to start automatically. One of the advantages of using a service for the integration task is the service independence and ease of maintenance operations like service version update, starting and stopping the service.

The service comprises functions for giving real time data to reporting part of the management application as well as commands formulated by the management application for the production system to execute.

The production system logic is controlled by either the business management application (external control via the integration service) or by its own logic module if activated.

The service integration module uses temporary tables as buffers in order to exchange data between production application and business management tools. Reading and writing from/into these tables is done with access rights for both of the applications. The service integration module is a part of a database system kernel server which works in a conjunction with a client application that connects to the mentioned server and formulates requests.

II. DATABASE MANAGEMENT SYSTEM KERNEL SERVER(DBMSK)

The database management system kernel represents the general server which has as an application integration configurable module.

The server uses a typical architecture. The remote client sends a request to the Database Server. For a particular request, the server decides on and takes the appropriate action.

This can be represented graphically as below:

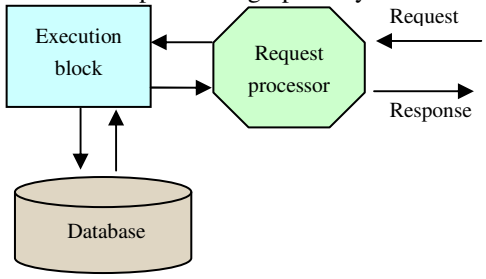


Fig.1. DBMSK general architecture

The main functions of each block are explained below:

- Request processor takes as input a request expressed in structured query language and interprets it. After all request characteristics have been determined, it is up to the execution block to act appropriately.

- Execution block communicates with both the request processor and the Database layers in order to execute the client request. It must be mentioned that this layer executes when the request characteristics have been determined by the request processor layer and only then.

Both, requests and responses can take message or file transfer forms, depending on the specific context.

A simplified graphical representation of the Client application architecture is depicted below:

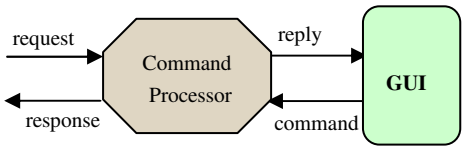


Fig.2. Client general architecture

As it can be easily seen, all that happens is driven by user commands, making use of a well-designed graphical user interface.

The command processor is a logical entity that transforms client requests into an appropriate format for the server to understand and process.

Request can take the following two forms:

- structured query language requests
- standard requests

All these requests are transported between client and the server encapsulated in a general request type, this ensuring both, flexibility and extensibility for the request transport level. The communication mechanism between client and server uses both messages and files. The data interchange operation is realized via file transfer making use of standard XML language.

For implementing the Server and Client software applications general programming books [4] [11], C++ programming books [3] [5] [8] [12], database books systems [6] [7] [8] and socket programming books [9] [10] have been used.

III. SERVICE INTEGRATION MODULE

We present in Fig.3 the general architecture of a production system. Generally, the integration must be accomplished in order to give a reporting system or management software application with real data from the production system.

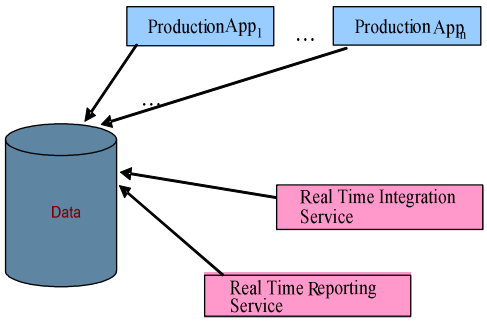


Fig. 3 System Integration – General Architecture

The demand of the enterprise is to share data and processes without having to make sweeping changes to the applications or data structures. Only by creating a method of accomplishing this integration can Enterprise Application Integration be both functional and cost effective. [1]

In general, software application integration must take into account all application specific aspects, communication protocol and various ways of storing data structure (proprietary methods for manipulating data or general relational databases like Oracle, Microsoft SQL Server).

In considered integration, production system uses a relational database to which several production applications formulate continuously SQL requests. These requests are general request made by production application like: insert, select, update, delete. It is well known the fact that every relational database management system kernel achieves lock database operations on some specific table when executing specific SQL requests.

In this context, appears the problem of generating data for management applications via a real time system integration component of the proposed database management system kernel.

A very important aspect is that management data is needed on a constant base for decision purposes. The client application, which commands the kernel server, is responsible for business data representations and integration workflow.

The integration service main responsibility is to get the desired data from the production system taking into account specific data format and production business logic and provide the management software application with necessary data in required format and according to management application's logic. To execute this function the kernel server formulates SQL queries to production relational database. The problem to solve is that due SQL requests generated by production software application: App1 …Appn, the production relational database is overloaded and kernel server SQL requests can not be executed in a specific, fixed period of time set by kernel server at query setup phase. To solve the problem, dynamic values for that timeout period of time, depending on production relational database overload are to be determined for specific periods of time.

The integration SQL has to get required data from tables pertaining to proprietary software application AppA:

Select [Field1, Field2, Field3, Field4 … Fieldn]
From
T1
Inner join T2 on [specific fields]
Left join T3 on [specific fields]
…
Inner join Tm on [specific fields]
Where [Integration condition]

where:

- *T1, T2, … Tm* represent specific tables associated with software application AppA.
- *[Field1, Field2, Field3, Field4 … Fieldn]* represent required integration fields from AppA.
- *[specific fields]* represent fields on which join between tables is carried out.
- *[Integration condition]* represents specific SQL integration condition, used to select integration data

The data generated using integration SQL is used to populate the tables of software application AppB. AppB usually writes specific data to its associated tables. Hence, data writing to AppB must be synchronized between the integration module and AppB itself.

As production AppA generated a considerable amount of data as a daily basis, which is reflected in the data composition of the associated tables (the number of records in AppA's tables is growing faster), integration SQL execution time does not satisfy integration time requirement. Proper index structure was considered when executing integration SQL.

A solution would be to change significantly database structure associated to AppA, but this is not the case because we are dealing with proprietary system. Also, the cost to design and build from scratch the whole necessary software is very high.

Thereby, a timestamp field will be added to

large tables pertaining to AppA. Let us denote this field TS. This is a field used only by integration module. After the integration for a particular set of data is achieved, the corresponding TS field will be updated with the integration moment of time. We also need an index associated to this field.

The modified integration SQL takes into account the TS field. The extra condition is as follows:

- TS > T_1
- TS > T_1 and TS <=T_2

where T_1 and T_2 are specific moments of time for which the integration will be carried out.

Depending on the database load, we must adjust integration SQL query time and T_1, respectively T_2 when necessary in order to successfully execute integration SQL in a reasonable amount of time (the target SQL execution time is at least 12 seconds).

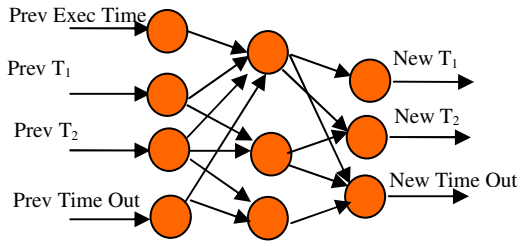
Without this parameters adjustment, normal SQL execution times are:

TABLE 1. Integration SQL execution time

No. Days	***1***	***10***	***15***	***20***	***30***	***45***	***50***	***60***
Seconds	***2***	***5***	***10***	***20***	***25***	***30***	***40***	***>60***

As it can be seen from this table, after a production system run for 60 days, the integration SQL does not execute in due time. With the proposed adjustment, SQL execution time is between 2-10 seconds, depending on the database load.

A typical back-propagation neural network [13][14] is capable of solving the problem in order to constantly adjust query timeout at setup phase and T_1, respectively T_2 timestamps, based on previous SQL execution time on the production database.



The neural network will have as input T_1 and T_2 corresponding to the integration moment of time, previous SQL query execution times, previous SQL query timeouts (the time interval for which the kernel server will wait for a response from production relational database for a specific query). The output layer will be used to adjust timeout interval for the integration module, T_1 moment of time and T_2 moment of time when necessary.

The neural network, the server kernel and client as well as the described integration service were implemented in C++. As compared to other proprietary integration systems (IBM, Oracle) proposed solution, besides implementation cost has the advantage of having less middleware levels which translates in smaller execution times, but has not many configuration options like proprietary systems mentioned above.

IV. CONCLUSIONS

Presented application pertains to software application integration class. Using SQL parameters modification via proposed neural network the integration SQL successfully executed in a period of time 3-10 seconds. This execution time compared with static integration SQL execution time is much lower.

The Database Kernel, which contains the software applications integration module, was designed with futuristic thoughts. Therefore, it is suitable for any kind of application that involves custom data storage, retrieval and processing. In addition, the file transfer component of the server can be used in remote backup applications.

REFERENCES

[1]. David S. Linthicum, Enterprise Application Integration, Ed. Addison-Wesley Professional; first edition ,(1999)

[2]. Daniel Serain, Middleware and Enterprise Application Integration: The Architecture of e-Business Solutions, Springer; 2nd ed. Edition, (2002)

[3]. Shaharuddin Salleh, Albert Y. Zomaya, Sakhinah A. Bakar, Computing for Numerical Methods Using Visual C++, 1st edition, (2007)

[4]. Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest, Introduction to algorithms, 2nd edition, The MIT Press, (2001)

[5]. Bjarne Stroustrup, The C++ Programming Language: Special Edition, 3rd Edition, (2000)

[6]. Richard A. Bassler, Jimmie J. Logan, The Technology of Data Base Management Systems College Readings; 3d ed edition ,(1976)

[7]. Peter Rob, Carlos Coronel, Database Systems: Design, Implementation, and Management, Eighth Edition, (2007)

[8]. Lyn Robison, K. David White, Database Programming with Visual C++ in 21 Days , Pap/Cdr edition, (1998)

[9]. Dave Roberts, Developing for the Internet with WinSock, Bk&CD-Rom edition, Coriolis Group Books, (1995)

[10]. Jeffrey Richter, Christophe Nasarre, Windows via C/C++ (Pro - Developer), 1st edition, (2007)

[11]. Anthony Jones, Jim Ohlund, Network Programming for Microsoft Windows, Microsoft Press, (2002)

[12]. Douglas C. Schmidt, Stephen D. Huston, C++ network programming, 1st edition , Addison-Wesley Professional, (2001)

[13]. Toshinori Munakata, Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More (Texts in Computer Science), Feb 4, 2008

[14]. Jeff Hawkins, Sandra Blakeslee, On Intelligence, 1st edition, Numenta Inc, 1st edition, (2005)

Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 53(67), Fascicola 1, 2008

Redundancy and Testability in Digital Filters

Horia Carstea[1], D. Margeloiu, O. Mitariu

Abstract – Threat issues in specific applications of digital filters are investigated. Since these redundant faults tend to appear in the same general location as test-resistant faults, the presence of many redundant faults can hide significant untested faults despite high overall test coverage. Classes of redundant faults that arise in digital filters are described and we propose a suite of technologies for identifying and eliminating the most common redundancies based on arithmetic optimization.

I. INTRODUCTION

With more commercial products incorporating digital signal processing (DSP) functions, testable design has become a "pressing issue among DSP designers". Developing high coverage tests for application-specific is considerably complicated by the presence of these random-test-resistant faults.

We will focus on the finite-impulse response (FIR) filters since they are perhaps the most widely implemented class of digital signal processing applications and are a basic building block of many more complex systems. However, the general approach is geared towards any system that can be described as a network of shift, add, delay sign extension, and truncation elements.

In order to gauge the efficacy of the approach across a fairly broad slice of designs, we will use fire filter specifications selected from the literature [1],[2] that include three lowpass filters, a wide-band bandpass filter used in video processing, and a predistorsion filter.

The overall approach is shown in Fig. 1, where we will be focusing on the shaded partition.

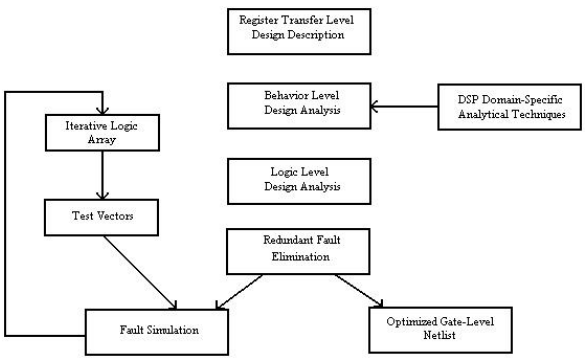


Fig. 1 Overview of design flow

The register-transfer-level (RTL) design description is analyzed to identify structures that are redundant, and the logic is marked to indicate the specific redundancies that are embodies. The designs will be implemented using three different common architectures: cascaded ripple-carry adders, carry-save pipelines, and adder tree structures.

II. FAULT MODEL

Since the principal active element in all the designs was the full-adder cell, the fault model used for this cell is of a same concern. We used the common gate-level model shown in Fig. 2, where the faults modeled are the stuck faults at gate input and output pins. Often, the hardest tests to apply using pseudorandom techniques are those associated with overflow conditions at the next-to-MSB full adder.

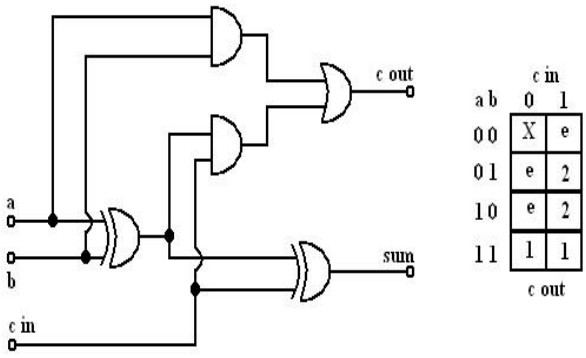


Fig. 2 Full adder gate-level model and associated carry logic test

If the c input is the carry input to the full adder, under this fault model, testing the carry logic requires fire tests: the three labeled e (essential) on the right half of Fig. 2, and one from each of the two equivalence classes 1 and 2. Under this model, overflow test 6 is nonessential, since the logic test by it can also be easier (more probable) test 7.

III. SCALING

The single most important design for testability optimization that can be performed on filters is to scale signal widths to the minimum width needed

[1] Facultatea de Electronică şi Telecomunicaţii, Departamentul de Electronica Aplicata, Bd. V. Pârvan Nr. 2, 300223 Timişoara, e-mail horia.carstea@etc.upt.ro

eliminating redundant sign bits. Computing the minimum width needed to hold a signal can be performed using any member of the standard fixed-point scaling techniques [3]. In redundancy elimination we use L_1 scaling since it is the most conservative scaling technique, guaranteeing that the circuit behavior will not be altered. Mode w_k can be characterized using the idealized impulse response of the subfilter that outputs at the interval mode:

$$w_k[n] = \sum_{i=0}^{M_k} h_k[i]x[n-i] \quad (1)$$

Where $h_k[i]$ is the impulse response of the subfilter and M_k is the order of the subfilter.

Using the property that the magnitude of a sum is less than or equal to the sum of the magnitudes of it's terms, and then replacing $x[n-i]$ with x_{Max} (the maximum input signal magnitude) we obtain:

$$|w_k[n]| \le \sum_{i=0}^{M_k} |h_k[i]||x[n-i]| \le x_{Max} \sum |h_k[i]| \quad (2)$$

This gives an upper bound on the signal amplitude at the internal mode. Without knowing more about the characteristics of the input signal, we assume that it is capable of swinging through the full range available to it ($x_{Max} = 1$). The upper bound on the signal amplitude is then:

$$|w_k[n]| \le B \quad (3)$$

and:

$$B = \sum_{i=0}^{M_k} |h_k[i]| \quad (4)$$

IV. APPLICATION

The target applications will focus on where the finite impulse response (FIR) filters are. To provide a brief review of terms and definitions, FIR filters essentially perform a weighted moving average of a sequence of input sample. This is described by the linear constant coefficient difference equation:

$$y[n] = \sum_{i=0}^{M} h_i x[n-i] \quad (5)$$

Where M is the filter order, $y[n]$ is the output signal at time n, $x[n]$ is the input at time n, and $h_i, 0 \le i \le n$ is the set of filter coefficients, which also corresponds to the impulse response of the filter.

The designer frequently has some flexibility in choosing the filter coefficients, and it is often possible to select the h_i, such that they can be expressed as a power of two or the sum or difference of two powers of two, which leads to efficient VLSI implementations.

V. CONCLUSION

Redundant fault can be an obstacle to gauging the true effectiveness of any test scheme, particularly in application specific digital filters where these faults can be hard to distinguish from highly test resistant fault. Analysis of the register-transfer-level design using arithmetic techniques based on scaling theory and signal phase and magnitude constraints provides an efficient means of identifying and eliminating most redundant faults in these designs.

Elimination of these faults can be done as a preprocessing phase for more accurate fault simulation or it can be used to eliminate redundant logic from the design itself, in which case it is possible to make significant area reductions as compared to moderately optimized designs.

In many cases, smaller than the area of the logic removal by redundant fault eliminations for a net reduction in area over the nonoptimized design with no self test capabilities.

REFERENCES

[1] T.O. Powel, K.M. Butler, M. Ales, R. Haley and M. Perry "Correlating defect level to final test fault coverage for modular structured design" *Proc VLSI Test*, 1994 pp 152-196

[2] P.C. Maxwell, "Reductions in quality caused by uneven fault coverage of different areas of an integrated circuit*" IEEE Trans. Computer-Aided Design*. Vol. 44 pp 603, 606, May 1995

[3] L. Goodby and A. Orailogln, "Pseudorandom pattern test resistance in high-performance DSP data paths", *Proc 33rd Design Automation Conf* . 1996 pp 813-816

[4] H. Carstea "*Strategii de inspectie si testare in electronica*", Editura de Vest, Timisoara, 2007.