

A Database System for Real-Time Event Aggregation in Telecommunication

Jerry Baulier Stephen Blott Henry F. Korth Avi Silberschatz

Information Sciences Research Center
Bell Labs, Lucent Technologies
600 Mountain Ave, Murray Hill, NJ 07974
jdb,blott,hfk,avi@research.bell-labs.com

Abstract

Telecommunication networks process very-large numbers of events in real time. In this environment, database applications demand both high throughput (at reasonable costs), and predictable, millisecond response times. Conventional disk-based database systems were not designed to meet such requirements. This paper sketches some real-time telecommunications applications, describes their database requirements, and then introduces Sunrise, a specialized, database system for real-time event processing and aggregation in telecommunication. Sunrise's architecture features shared-nothing parallelism, a main-memory storage manager (DataBlitz), and a programming tool which allows new services to be authored and installed in an on-line system without interrupting event processing. Sunrise is an industrial-strength system, and has been used as the platform for a number of telecommunication applications with real-time event-processing and aggregation requirements.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

**Proceedings of the 24th VLDB Conference
New York, USA, 1998**

1 Introduction

Telecommunication systems process very-large numbers of calls in real time, and with a remarkable degree of reliability and predictability. While the principal goal of the telecommunication network is establishing circuits and carrying network traffic, other tasks such as billing, fraud detection or prevention, and network management are equally critical to the operation of those networks. Consider, for instance, the case of billing. Billing is clearly fundamental to revenue collection; however, it has also become important as a marketing differentiator. With little other service differentiation, carriers in the US (and increasingly also elsewhere) compete based on the novelty and appeal of their billing plans. In addition, prepay billing capabilities (where charges are deducted immediately from a prepaid account) also open up the large market for which credit-based billing is inappropriate. As of 1997, approximately 30% of new applications for cellular telephones in the US were rejected because of inadequate credit history.

A billing application, in its simplest form, assigns a charge for each call. However, determining the charge usually requires accessing and maintaining a database of customer and billing information. First, a database access is usually required to read a customer's subscription information. Subsequent accesses may then be necessary to determine, for example, whether the caller and the callee have a special discount relationship (such as MCI's 'Friends and Family' program in the US), or to establish a rate based on the distance of the originating number from the terminating number, etc. In addition to these read-only database accesses, it is also common to maintain summaries over events. In billing, summaries typically include per-customer usage aggregations, such as the total number

of calls, the total call duration, or the total charge in each billing cycle. This information is used to calculate usage-based rates and discounts.

In fraud detection or fraud prevention applications, summaries, termed ‘fraud signatures’, are stored for each customer. Fraud signatures capture each customer’s typically usage pattern, and are updated with each new call. They can be used to detect irregular usage patterns—with similarity to known fraudulent patterns—when and if they occur. If a call can be identified as potentially fraudulent, then the call may be rerouted to a voice recognition system or an operator for authentication. This requires a fraud-signature database to be accessed during call setup (that is, between the time when the caller completes dialing and when they receive the ring-back tone), and updated in a timely manner upon call completion. Moreover, fraud signatures and their algorithms must be adapted over time as fraud patterns change.

Another telecommunication application requiring database processing during call set-up is intelligent toll-free number mapping. In the US, so-called ‘toll-free numbers’ are those where the charge for a call to that number is assigned to the callee (rather than to the caller). However, a toll-free number is often also a virtual number in the sense that it is mapped within the network to different terminating numbers based on criteria such as the day, the time of day, and the load-balancing policies in effect. For example, calls using the same toll-free number may be routed to an eastern location during the morning, and to a western location in a different time zone during the evening and at night. During the day, a load-balancing algorithm may be used to distribute calls between these two locations. This application requires a database of number mappings and algorithms for each toll-free number. Moreover, the database is accessed as part of call set-up, and must be programmable since the mapping can be non-trivial and may change frequently.

These types of telecommunication applications require many of the standard data-management features of conventional database systems such as data independence, high-level, declarative programming interfaces, and the ACID correctness guarantees for transactions [GR93]. In addition, however, they also generally exhibit the following characteristics and requirements:

- The predominance of ‘rifle-shot’ transactions, where each transaction consists of one or a small number of keyed table accesses using existing indexes, frequently hashed indexes;
- In many cases (including prepay billing, fraud prevention, and intelligent toll-free number mapping), predictable response times—on the order

of only tens of milliseconds—such that event processing can take place during the critical set-up phase of a telephone call;

- The ability to scale a system up or down to achieve whatever peak, busy-hour throughput is required (at a reasonable cost), while continuing to meet response-time goals;
- The reliability of a utility such as gas or electricity, whereby the system cannot be taken off-line at will for maintenance and upgrades; and
- The ability to define new event-processing steps and new summaries—or adapt existing processing steps and summaries to new requirements—and install those changes in an on-line system, without interrupting on-going event processing.

Currently, custom database systems are often used in practice to meet the needs of applications with these requirements. Such custom solutions work well, but make it impossible to amortize the cost of a system over a large number of applications, and thus become expensive to develop and maintain.

This short paper describes Sunrise, a database system developed at Bell Labs.¹ Sunrise is general purpose in that it serves as a platform for many real-time telecommunication applications. However, it is also specialized in that its architecture and functionality have been adapted specifically to meet the requirements sketched above. Only an architectural overview and selected highlights are presented here, and the interested reader is referred to the corresponding white paper for more details [BBKS98].

2 Architectural Overview

This section discusses several highlights of the Sunrise architecture, which is illustrated in Figure 1.

2.1 A Main-Memory Database Platform

Current network applications with real-time performance requirements do not rely on conventional disk-resident database systems. Disk-resident database systems buffer only a small part of a database in main memory. The rest of the data is accessed from the disk when and if it is required. A single disk access can account for from tens to hundreds of milliseconds, making the goal of predictable response times (on the order of tens of milliseconds) unachievable. Moreover, there can also be comparable overhead due to communication costs among clients and servers, and due to

¹Sunrise has been announced as a Lucent Technologies product under the name QTM.

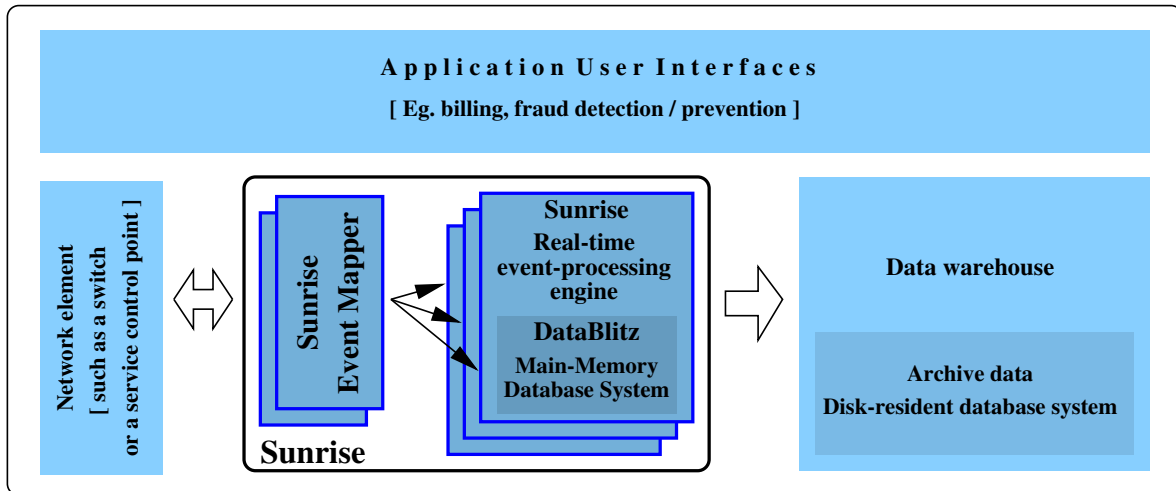


Figure 1: Sunrise’s run-time architecture: Real-time event processing and aggregation for network applications, with a data warehouse for long-term archiving of processed-event records

the lookups required to locate pages in the database buffer.

To meet its real-time performance requirements, Sunrise uses DataBlitz, a main-memory storage manager, as its underlying storage system² DataBlitz, having been designed under the assumption that the entire database resides in main memory, processes database operations at main-memory speeds. The only disk operations during normal processing are those necessary to write the log records of committed transactions to disk. As memory prices fall, main-memory residency is becoming an increasingly-attractive means to achieving the performance demanded by many telecommunication applications.

For performance reasons, Sunrise and DataBlitz execute within the same process address space. The advantage of this approach is that the communication overheads of a client-server architecture are all but eliminated, and database data need not be copied between buffers. The potential disadvantage, however, is that any error condition occurring in Sunrise could corrupt the persistent database. This is a serious concern since Sunrise is programmable at the user level, and cannot be shipped as an exhaustively-tested and verified unit of code. This potential safety risk is addressed through the service authoring environment, which is discussed in Section 2.4.

2.2 Storage of Processed-Event Records

For many applications, processed-event records cannot simply be discarded, and storing all such records

²DataBlitz is a Lucent Technologies product based on the ‘Dal’ research prototype developed at Bell Labs [BRSS97, JLR⁺94].

in a main-memory database would be prohibitively expensive. As such, Sunrise uses DataBlitz for storing only summary data, while processed-event records are archived in a conventional, disk-based data warehouse. This situation is illustrated in Figure 1. Sunrise provides an interface through which one popular commercial DBMS can be used as the data warehouse, although that can easily be replaced with any other functionally-equivalent relational system.

The distinction between main-memory data and archived data is fundamental to the Sunrise architecture. Event processing, which is response-time critical, generally depends only on main-memory resident data, while a conventional, disk-based data warehouse is used for more voluminous, archived data with less stringent access requirements. In particular, because of the costs and addressing limitations of main memory, Sunrise itself requires the space complexity of its applications to be bounded over any sequence of events, regardless of the length of the sequence.

2.3 Scalability and Parallel Processing

Sunrise in fact consists of a number of components. The ‘real-time event-processing engine’ is the database component which is coupled to DataBlitz and performs event processing. To achieve scalability, Sunrise allows a number of instances of this engine to run in parallel, as illustrated in Figure 1. In particular, the database is partitioned according to some key attribute (such as a phone number or an account number), and configuration, summary, and aggregation data for each key is clustered at a single site. Sunrise’s external interfaces are exposed through components known as ‘event mappers’. Event mappers assign each event for pro-

cessing at the site at which the corresponding key data is stored. In the (infrequent) case that event processing spans several sites, then a single site is designated as the coordinator. The coordinator is then responsible for dispatching sub-events to participating sites, and aggregating the results of sub-event processing.

All components are usually tightly coupled on a fast interconnect, and communicate with one-another using TCP/IP. This parallel, shared-nothing architecture is well-suited to the rifle-shot transactions common in telecommunication systems, and can deliver close to linear scale-up.

2.4 The Service Authoring Environment

Sunrise provides a high-level, fourth-generation programming interface referred to as the ‘service authoring environment’. The service authoring environment consists of a set of graphical user interfaces for administration, maintenance and authoring, and a so-called *service-authoring language* (or ‘SAL’) in which new services are programmed. SAL consists of a primitive set of event-oriented programming constructs, and a small but rich set of declarative data-management features.

The data-management features of SAL are restricted versions of the four table operators of SQL (`insert`, `delete`, `update`, and `select`). SAL scripts are validated at a high level, then translated to C++ code and compiled. The resulting object code is then dynamically linked into the event-processing engine, without interrupting regular event processing. This approach to the compilation and installation of authored code achieves three goals:

- Compilation in this way avoids the costly overhead of interpreting event-processing logic at execution time, thereby improving performance. In particular, event-processing logic which is authored in a high-level language, is then executed as compiled C++ code directly within the same address space as the database itself.
- Through the use of dynamic linking, applications can be installed or upgraded on-line, without interrupting on-going event processing.
- The use of a high-level language such as SAL allows many classes of errors to be eliminated. In particular, errors such as infinite loops, memory leaks, erroneous updates through misdirected pointers, or segmentation violations cannot occur. Other data-dependent errors such as division-by-zero are handled safely as exceptions. This mitigates the potential safety risks associated with executing application-specific code within the same process address space as the database itself.

In addition to compiling and installing services in this way, SAL also provides a mechanism to control the services which are executed for each event, and the order in which they are executed. In particular, event processing is subscription based. We assume that, for each event, a ‘subscriber identifier’ can be extracted and used to access subscriber-specific information in a subscription table. The subscriber identifier is the same as the ‘key’ mentioned in Section 2.3 above, and need not correspond to any notion of customer or a person. Subscription information is used to determine the set of services to invoke in processing an event for a particular subscriber. Subscription information is encoded in a form which is amenable to efficient interpretation at run-time (much like Java byte-codes).

This approach was chosen as a compromise between the performance benefits of compiled, stored queries, and the flexibility of ad hoc queries. The service part is compiled, and the subscription part interpreted. Moreover, subscription information consists of data entries in a table, which can be updated without re-compilation or dynamic linking.

2.5 Summary and Aggregated Data

SAL supports two approaches to maintaining summary and aggregation tables. The first approach is to define a service that encodes the logic necessary for summary maintenance in terms of explicit table inserts, updates and deletes. This approach, however, places the programming burden on the author, and is inherently error prone since *all* authors must be aware of when and how summary maintenance routines should be invoked. The second approach to maintaining summaries is through a view mechanism. As in SQL, views are defined declaratively. In Sunrise, however, views are always materialized within the main-memory database. Given a view’s declaration, its materialization is kept up-to-date automatically as a side-effect of updates to base tables. Thus, a lookup in a view is processed at the same speed as a lookup in any other database table.

The semantics of Sunrise views differs somewhat from the semantics of conventional database views. In conventional databases, views are defined over other tables and views in the database. In Sunrise, the base data for aggregation and summary is the processed-event records, which are usually stored in the data warehouse. To accommodate this case, SAL defines the concept of a ‘chronicle’, or an append-only table [JMS95]. Although chronicles can be thought of as tables, they are not stored in the main-memory store, but rather are an abstraction in terms of which outputs are generated, and views are defined. The only update operation on a chronicle is `insert`. Whenever

a tuple is inserted into a chronicle, a record is generated on an output stream. Views can be defined over base tables, chronicles and other views, and are always materialized.

Chronicles and views are defined in such a way that incremental view maintenance can be performed without visiting the processed-event records of the underlying chronicles. This is important since lookups in the data warehouse would have a serious impact on response time, and possibly also on throughput too. Views can frequently be maintained in time which is independent of or logarithmic in the size of the database. A ‘chronicle algebra’ has been defined previously [JMS95], and SAL provides an adapted version of that formal algebra cast into an SQL-like syntax.

2.6 Machine Architecture and Performance

Although Sunrise should run on many Unix systems, so far it has been developed and fully tested only on SUN Solaris platforms. Portability from one platform to another is limited mainly by the portability of DataBlitz, the underlying storage manager.

Performance tests from an early implementation phase have shown throughput on the order of 600 events processed per second on a single-processor SUN UltraSPARC machine (using group commit). This was for a simple, update-intensive billing application with the data warehouse running on the same processor. Performance tests with DataBlitz directly have demonstrated 860 updates per second, and nearly 60,000 reads per second.

3 Conclusion

This short paper has sketched a class of telecommunication applications requiring high throughput and real-time responsiveness. We have also presented an overview of Sunrise, a real-time event-processing and aggregation system developed at Bell Labs to meet the needs of such applications. A first full release of the Sunrise platform, which consists of nearly 400,000 lines of code, was made in January 1998, and the first products based on Sunrise are scheduled for delivery during the third quarter of 1998.

Another emerging area for real-time event processing and aggregation is internet network management, authentication, authorization and billing. Consider, for instance, the case of billing for internet access. The two currently-prevalent pricing models—based on either flat monthly fees or connection times, or a combination of these—have limitations. Flat-fee pricing has the advantage of simplicity, but is unattractive to low-volume customers (those who perhaps access e-mail only once or twice a week). Moreover, since there is no incremental cost to using the network, there is little to

prevent a small group of users from consuming a large proportion of the network’s resources. Also, emerging technologies such as Digital Subscriber Line (or DSL) and cable modems make connection-time–billing somewhat obsolete. In particular, DSL offers the possibility of permanent network connectivity, with no need to start and stop dial-up connections explicitly. In this context, a move towards usage-based billing for home internet access seems possible, and we are currently considering the real-time event processing and aggregation capabilities which will be required to support such internet billing applications.

More detailed information on Sunrise is available at <http://www.bell-labs.com/project/sunrise>, including a white paper [BBKS98]. Also, more detailed information on DataBlitz is available at <http://www.bell-labs.com/project/dali>.

References

- [BBKS98] Jerry Baulier, Stephen Blott, Henry F. Korth, and Avi Silberschatz. Sunrise: A real-time event-processing system. *The Bell Labs Technical Journal*, 3(1), January-March 1998.
- [BRSS97] Philip L. Bohannon, Rajeev R. Rastogi, Avi Silberschatz, and S. Sudarshan. The architecture of the Dalí main memory storage manager. *The Bell Labs Technical Journal*, 2(1):36–47, Winter 1997.
- [GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., 1993.
- [JLR⁺94] H. V. Jagadish, Daniel Lieuwen, Rajeev Rastogi, Avi Silberschatz, and S. Sudarshan. Dalí: A high performance main memory storage manager. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1994.
- [JMS95] H. V. Jagadish, Inderpal Singh Mumick, and Avi Silberschatz. View maintenance issues for the chronicle data model. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 1995.