An approach to successful online transaction processing applications

by ARMOND INSELBERG
Synapse Computer Corporation
Milpitas, California

ABSTRACT
Online transaction processing applications are playing an increasingly important role in corporate activities. While transaction processing applications can be characterized as a large number of users concurrently accessing and updating very large databases, there is a fundamental set of generic user requirements which such applications must satisfy to be successful.

This paper shows that with system hardware and software architecture designed specifically to support online transaction processing, the system is able to fully meet these application requirements. Furthermore, by implementing applications on such a system, the impact of these requirements on the development process is minimized; this in turn reduces the time, cost, and complexity of developing and maintaining transaction processing applications.

Following a brief discussion on the nature of online transaction processing, this paper discusses the user's generic transaction processing requirements, the characteristics of online transaction processing application programs, and how a system's hardware and software architecture can be designed to meet the requirements of transaction processing. The SYNAPSE N+1™ System is used as an example architecture.
INTRODUCTION

There is a major trend to moving applications online. This trend is motivated by corporate competition, customers wanting to know their most current account information, and managers requiring the most up-to-date information in order to effectively make decisions.

In online transaction processing, large databases are concurrently accessed and updated by a large number of online users. Examples of transaction processing applications are provided in Figure 1.

As shown in Figure 2, there are three levels by which the nature of transactions may be defined.

1. A business transaction is viewed as the unit of work needed to complete a business function. Booking a customer's airline flight reservation is an example of a business transaction. While making the reservation may include several processing steps and interactions with the system, the reservations clerk considers the activity to be one logical unit of work.

2. An operator transaction involves an operator's single request and response interaction with the system.

3. A database transaction is a group of database modifications considered to be a single unit. For example, a customer transfer of funds from a checking to a savings account involves several database modifications which comprise a single database transaction. Either all the activities of the transaction must proceed to completion for the database to remain consistent, or none of them are allowed to take place.

The concept of a database transaction emphasizes the fundamental focus of online transaction processing on database activities and database consistency. The definition also provides the foundation for a transaction processing application design model.

Hundreds or thousands of users at their terminals directly generate the incoming online transaction processing workload. The fact that transaction processing arises from user-driven workloads results in the following set of generic user requirements.

1. Accurate and consistent database
2. High degree of concurrency
3. Acceptable throughput and response time
4. High application availability
5. Support of unpredictable workload growth
6. Modifiable application functionality and data requirements
7. Secure database
8. Dialogue-oriented user interface
9. Distributed users and integrated systems

These generic user transaction processing requirements are not inherent to the applications themselves; inherent characteristics are such that, if any one of the characteristics is changed, the application is no longer considered to be the same. Since the requirements are not inherent, they should be as transparent as possible during the application design and implementation process.

To achieve transparency, the functionality of the underlying system hardware and software must be designed to support these requirements. Figure 3 is a model of transaction processing showing user requirements, application characteristics, and corresponding required system functionality. These three
aspects of the model are discussed in the next three sections of this paper.

USER REQUIREMENTS

The nine user requirements encompass a wide range of transaction processing applications. These requirements are each examined from the user's viewpoint.

Consistency

It is mandatory that the database remain accurate and consistent, regardless of the type of transaction processing application. To guarantee database accuracy and consistency, only completed transactions are made permanent, and in the event of a system failure, incomplete transactions are backed out of the database. Database accuracy must also be preserved by allowing only valid data to enter the database.

Concurrency

A transaction processing application consists of a large number of users updating a single large shared database. The same set of transactions are executed repeatedly by many users. A high degree of concurrency must be allowed to enable performance requirements to be met. Additionally, synchronization and control must be maintained over the concurrent database accesses to assure that database integrity is maintained.

Performance

Transaction processing systems must meet an acceptable level of performance. Users perceive performance in terms of throughput and response time. While the specific levels of acceptable performance vary widely depending upon the applications, users expect these levels to be met. The defined response time must be maintained throughout the day, including peak transaction rate periods. Consistent response at a particular level is more important than the measured response time for any selected user.

In general, response times of less than three or four seconds are needed to enable a user to remain effective on the job. It is important to note that while users demand rapid response once they submit a transaction, the think-time between transactions is relatively long. Thus, a user's transactions are separated by long periods of reading the last response, thinking...
about the next data to input, and keying the new input. Batch processing is also needed so that noninteractive transactions can be easily spun-off and processed as background activities.

Availability

As users become increasingly dependent on the online transaction processing application, the significance of downtime increases. High availability is often viewed as availability 99.98% of the time. Thus, high availability for a 24-hour/day application requires the total downtime for the year to be no more than 105 minutes. Furthermore, high availability means that the system is not only running, but that the database is accurate and consistent; it does no good for the application to be executing using a corrupt database.

Growth

It is the nature of transaction processing applications that their workloads grow at unpredictable rates over time; the growth may be due to an increase in the number of online users, which in turn increases the number of transactions per second; it may arise from the applications increasing in functionality, thus requiring more computing resource per transaction; or the workloads may grow from the development of new applications. It is important that response times remain consistent as the transaction processing workloads grow. While the functionality of an application tends to vary inversely with the transaction rate, as shown in Figure 4, functionality tends to increase as the complexity of the business environment increases.

Modifiability

Online transaction processing applications must be modified as the business activities that they support change. The modifications may require a change to the functionality of the applications. The application's data requirements may also change, requiring additional data to be held in the database, or the application to share portions of the database previously used by other applications.

Security

The sensitive nature of transaction processing data requires that the database be secure. Users must be protected from accidental or deliberate attempts at unauthorized access to their data. The security mechanism must provide auditability of such attempts. Furthermore, users must be able to perform both development/testing and production on the same machine at the same time without breaching security.

Dialogue

Offering easy-to-use user interface mechanisms, such as menus and screen forms, facilitates a user's dialogue with the system. The interface should be complemented by an extensive online HELP facility. Such an interface significantly reduces the training requirements and user resistance, especially by users with no data processing background.

Distribution

As online transaction processing applications become broader in scope, it is necessary to address the geographic distribution of users. The users may be geographically distributed throughout a decentralized corporation, and thus need remote access to the central database. As new online applications are introduced into the corporate environment, it is also necessary to be able to integrate these applications into the existing environment.

APPLICATION CHARACTERISTICS

To the user, a transaction processing application executes via a series of menus and screen forms. Such applications generally experience long user think-times, short transaction execution times, repeated use of the transactions, and simultaneous multiple users. In order for applications to fully meet the user's requirements while those requirements remain essentially transparent during the development process, it is important to develop the applications under a design methodology. The methodology takes the form of a design model which defines the structure of transaction processing applications, thereby accommodating the processing needs of both interactive and batch users. The model provides a framework for the design of the applications so that the resulting applications fit the functionality provided by the underlying system software and hardware.

By minimizing the impact of the user requirements during the development process, the time and costs for implementing applications are significantly reduced.

Applications that are designed using the Synapse Design Model are structured, recoverable, and efficient. The pro-
grammer simply writes the program modules to indicate which screen forms to be displayed, which data to be selected from the database, and which program module to invoke next. The programmer does not need to be concerned with database recovery, concurrency, and application restartability issues because they are handled automatically by the Synapse system hardware and software.

By breaking the application into small executable program modules, the Synapse model takes advantage of the long think-time typical of the applications. As indicated in Figure 5, by using small modules which invoke each other, the application minimizes the machine resources being used by "thinking" users who are viewing a displayed screen. Resources can therefore be devoted to providing responses to other waiting users. In this way, the system can efficiently multiplex a large number of users over a finite amount of available resources, thereby minimizing queuing and other traditional bottleneck problems.

**SYSTEM FUNCTIONALITY**

In order for the system hardware and software to fully support the generic user requirements, it is necessary that these requirements be considered at the time the underlying system architecture is designed. Of the three alternative computer architectures available for transaction processing, the monolithic mainframe architecture offers central access to all terminals and the entire database, but with it, entry costs are high, upgrades are expensive and disruptive, and the system is not fault-tolerant. The loosely-coupled multiple computer architecture provides easier upgrades and can be fault-tolerant, but extensive tuning, load balancing and high overhead from interprocess communication reduces its effectiveness. A tightly-coupled multiprocessor architecture with a single shared main memory best fits a transaction processing environment by enabling all terminals direct access to the entire database; system upgrades are non-disruptive, highly flexible, and offer linear performance gains; additionally, the tightly-coupled architecture is fault-tolerant and self-tuning.

Figure 6 shows capacity/cost growth curves for the three alternative computer architectures.

The following subsections describe how a tightly-coupled multiprocessor architecture, such as the SYNAPSE N+1 System, effectively meets the nine indicated requirements of a transaction processing environment.

**Consistency**

The user's requirement for database accuracy and consistency necessitates a highly developed database system logging and recovery mechanism. By considering the unit of recovery to be a database transaction, a database transaction happens either in its entirety or not at all.

As changes are made to the database, they are recorded in a log file. If a system failure occurs, the effects of uncommitted transactions are removed or rolled back from the database, and committed transactions are reapplied or rolled forward. When the recovery is completed, the system will restart the user applications at their appropriate restart point. By this...
process, the Synapse system guarantees that no committed data is lost due to a system failure.\textsuperscript{2,3}

It is important that the application restart point and committed updates to the database be synchronized; otherwise, following a system recovery, an already committed update may be incorrectly reapplied by the restarted application. Because the Synapse system automatically performs a database commit whenever an application checkpoint is established, this synchronization is assured, and the need for the application programmer to explicitly commit transactions is eliminated. Figure 5 shows that the entry point to and exit point from an application unit are automatic application checkpoints.

Database consistency is enhanced by developing the applications within the context of a centralized data dictionary. All data dependencies are catalogued in the dictionary, thus avoiding data inconsistencies, multiple data definitions, and general application redundancies. The Synapse data dictionary, called the APPLICATION DICTIONARY, is fully integrated into the Synapse system and is extensively used by the compilers, utilities, operating software, and applications. The APPLICATION DICTIONARY facility plays a major role in protecting the validity of the database. The range, type, and particular value specifications of data fields are used by the system to automatically validate all incoming data from the user’s screen. The APPLICATION DICTIONARY also validates all program output to the database on a field-by-field basis.

Concurrency

Having a large number of users concurrently accessing the same database requires that no user see a partially completed update transaction. The system can offer a highly concurrent environment, requiring no additional effort from the application programmer, by automatically locking granules of the database.

In the Synapse system, a granule is a row of a relational database table, or the entire table. This is comparable to record- and file-locking in a non-relational system. Based on the locking granularity specified, when an application accesses the database for update, the referenced data is exclusively locked. Alternatively, multiple read-only locks can be placed on the data. This implicit locking greatly simplifies the concurrency design of the application.

Performance

While the level of acceptable performance varies by type of application, the system must be able to offer a sufficient level of performance to be considered cost-effective. A major gain in performance for a transaction processing system is achieved by using a tightly-coupled multiprocessor architecture. Such a system allows multiple processors to communicate through a single shared memory, thereby significantly reducing interprocess communication overhead. Furthermore, all simultaneously executing processes can independently access the same disks. The Synapse caching strategy not only provides performance gains, but solves the classic multiprocessor bottleneck problems of shared main memory and bus contention. As described later, it is important that each Synapse General Purpose Processor uses non-write-through cache.

The Synapse shared main memory also allows the use of a single dynamically-allocated disk buffer. For each reference to a database page, the disk buffer is first searched to determine if the page is main memory. If the page is not in the disk buffer, the page is read from disk. The disk buffer reduces the number of disk I/Os for frequently-referenced data. The shared main memory also enables efficient use of system resources by sharing a single re-entrant copy of the system software and application programs. Sharing the executing software in main memory reduces the need to swap programs residing in memory, and enables the system software to be directly available as part of the execution image of each process.

By specifically designing the system software to meet the needs of transaction processing, performance can be further improved. For example, as shown in Figure 7, the Synapse Relational Database Management System (RDBMS) is actually embedded into the operating system software, and the Kernel Operating System is designed to directly support the Synapse RDBMS requirements. This approach is far superior to having the database management system run as a separate process which causes an additional layer of overhead.

In a transaction processing environment, the database queries are predetermined. This enables a large gain in performance to be achieved by determining the database navigation paths prior to runtime. Following compilation of an application, a Synapse utility called the Transaction Application Builder determines the optimum way to traverse the database and stores this information in the APPLICATION DICTIONARY as a database navigation path access module. At runtime, the access module is used to accelerate the retrieval process.

Another level of performance gain is achieved by evaluating, at a very low level in the Synapse system software, which rows of a database table satisfy the query. While many systems require that either the application or a high-level portion of the database system determine if the records satisfy the query, performing this evaluation at a lower physical-page
level only requires rows satisfying the query criteria to be passed to the application program. This results in fewer exchanges between the application and the Synapse RDBMS, and fewer rows being moved to the application from the database. Further performance is gained by minimizing synchronous writes to the disk for database updates.

Even in an online transaction processing environment, a full set of batch processing functionality must be able to run concurrently with the online applications. Batch activities on the Synapse system may be submitted by a user, or spun-off by online transactions. Batch jobs with similar priority are sent to the appropriate batch queue, assuring the proper work mix and maximum throughput system-wide.

Availability

High application availability means not only that the system is running, but that the database is guaranteed to be accurate and consistent. Therefore, not only must the system be fault tolerant, but it must be able to guarantee database integrity. Fault tolerance means that the system can automatically recover from any hardware or software single point of failure. While applications are not necessarily available during the recovery process, the impact of the system failure is minimized by the system automatically deconfiguring the failed component from the system, reestablishing database consistency, and restarting the applications at their appropriate restart point. Such a recovery process following a system failure ensures data integrity, and thus high application availability.

High availability is also sustained by enabling various operational activities to be performed without requiring the system to be taken offline. Thus, the Synapse system allows database backups and system expansions to be achievable while the system is online. In addition, these operations do not require significant operator effort or expertise.

Growth

Because transaction workload growth is unpredictable, the system must be able to expand modularly as requirements demand. Cost-effective modular expansion is achievable on the Synapse system because processors, channels, primary memory, or various peripheral devices may be added independently. Thus, a tightly-coupled system can expand those specifically needed resources, without having to add costly unrequired hardware facilities.

For modular expansion to proceed smoothly, it is important that the system architecture be symmetrical. This means that all the components must be functionally interchangeable and equitably share the workload. Additionally, a shared main memory enables transactions arriving at the system to be held in a single ready-queue. Because the Synapse General Purpose Processors are symmetrical, they can independently access transactions in the ready-queue. The symmetry of the processors in conjunction with a single transaction queue provides automatic load balancing, thereby eliminating a major ongoing labor-intensive activity by highly skilled technicians.

Another criterion for modular expansion is that the addition of resources provide linear performance. In order to fully expand the system and retain linear performance on a tightly-coupled multiprocessor system, it is necessary that bus and memory contention be minimized. Because each Synapse General Purpose Processor uses non-write-through, private cache, the amount of activity on the SYNAPSE EXPANSION BUS™ caused by a single processor is less than 2% of the total bandwidth of the bus. Non-write-through cache enables changes to be made to a processor's cache without requiring the changes to be immediately reflected in main memory. In conjunction with non-write-through, private cache, the Synapse system uses a proprietary data ownership scheme implemented in hardware to ensure that a processor always obtains the most recent value for each data item.

Another aspect of a system which facilitates its growth is the ease by which system administration can be achieved. By offering straightforward, highly-automated operational facilities, the need for expensive operations staff is eliminated by the Synapse system.

Modifiability

Because online transaction applications are closely tied to the business environment, the applications must be modified as the business activities change. These modifications may require changes to an application's functionality and its data requirements. In the Synapse system, modification of an application's functionality is greatly simplified by the screen forms, database elements, and application programs being defined independently of one another.

The modification of an application's functionality is facilitated through the integration of various development tools. Such tools as the Synapse Program Generator allow application code to be rapidly developed and modified, and user screen forms to be quickly created and altered.

While for many systems the changes to the application's code and screen forms is straightforward, changes to data requirements may be very cumbersome. The difficulty arises when the changes require modification to the underlying database schema. Unless the database system is based on the relational model, modifications to the database schema can require an unload and reload of the entire database under the new schema. The relational model, on the other hand, offers a high degree of data independence from the underlying physical storage structures. Such data independence allows new views of existing data elements and their relationships to be created without requiring modification to the stored database.

Security

Security is important because of the sensitivity of transaction processing databases. Most systems offer front-end security mechanisms to check user logins and OS protection mechanisms which restrict a process to the access of code and data only within its own domain. The Synapse system provides additional security by allowing the restriction of access to system objects, such as files, programs, and batch queues, to various users or groups of users. These restrictions, defined by
access lists and capabilities, can allow various modes of access, such as read with no update capability.

Security in the Synapse transaction processing environment is also provided by the binding of the application's programs, screen forms, and referenced database tables into a single application unit. An application unit provides complete containment of all objects comprising an application. Access to any other tables, forms, or programs is denied by the system. The binding also improves performance by resolving in advance all named references to internal system identifiers, and placing this information in the APPLICATION DICTIONARY.

**Dialogue**

For the transaction applications to viably support business activities, they must offer a dialogue-oriented user interface. Such an interface must be simple to use, yet sophisticated enough to meet the application's requirements. A hierarchy of menus and screen forms enables the user to utilize the functionality of the application by interactively traversing the hierarchy. Such a menu and forms system is offered on the Synapse system, as shown in Figure 7, by integrating the transaction processing manager software into the operating system. All session threads are managed by the Synapse Transaction Processing Manager, and in the case of a system failure, executing applications are automatically restarted at the appropriate point.

**Distribution**

To support geographically distributed users, data communications must be in place. Similarly, communication capabilities are necessary to ensure the proper integration of a new application into the existing environment. The integration can often require the interconnection of machines from different vendors. The online system may appear as a satellite to a large backend batch system, with data downloaded and uploaded between the systems. Appropriate means of connecting users and systems include X.25, 3270 terminal support and emulation, and 2780/3780 protocols.

**CONCLUSION**

The user requirements generic to transaction processing applications must be met in order for the applications to be successful. For these requirements to remain transparent during the development process, and in turn reduce the time, costs and complexity to develop applications, it is necessary for the underlying system hardware and software architecture to be designed to support these requirements. The SYNAPSE N + 1 System exemplifies this approach to providing successful transaction processing applications.

**REFERENCES**
