High Performance and Scalability through Application-Tier, In-Memory Data Management

The TimesTen Team
TimesTen Performance Software
1991 Landings Drive
Mountain View, CA 94043
info@timesten.com

Abstract

TimesTen Performance Software's Front-Tier product is an application-tier data cache that inter-operates with disk-based relational database management systems (RDBMSs) to achieve breakthrough response time and throughput, scalability in transaction load, high availability, and ease of administration and deployment. Front-Tier caches frequently used subsets of the corporate database on multiple servers in the application tier and supports SQL queries and updates to the caches. The caches may or may not be overlapping, are kept synchronized with the corporate database and with each other, and may be dynamically reconfigured to contain different subsets of the corporate database. Front-Tier provides the fundamental bridge between the corporate database and high-performance, scalable application servers. It eliminates the main barrier to application server scalability and high performance, namely the sole reliance on a centralized corporate database server for data management.

1. Introduction

Three-tier and multi-tier computing architectures have been popular for several years, having first been accepted as the standard architectures for enterprise computing, and more recently as the standard architectures for Internet computing. These architectures separate the presentation logic, hosted in the tier closest to the end user, from the application tier where all the business logic is implemented, and finally from the data management tier where the corporate database is centrally managed and administered. While these architectures have been suitable for enterprise computing, they are not as suitable for data-intensive, high-performance Internet applications. With the former, the number of users is limited and the interactions are well defined while with the latter, the number of users keeps increasing and low and predictable response times are a competitive differentiator. The separation of data management from the application tier imposes a performance overhead in accessing the required data, and a scalability barrier in forcing all queries and updates through the database tier.

This paper examines these traditional architectures, their advantages and drawbacks, and describes a new approach to data management that greatly improves the performance of data-intensive Internet applications.

2. Internet Computing Systems

Computing systems that service Internet requests consist of multiple tiers of systems that include web servers, application servers, and backend corporate DBMS server(s). The web servers handle incoming requests, routing them to the appropriate application server. Application servers implement the logic required to service requests. They execute the business logic and periodically query or update the corporate database. Finally, the database server hosts one of the popular disk-based RDBMSs and handles requests to the corporate database. Web servers and application servers may be scattered around the corporate Intranet, while the database server is typically consolidated in a centralized server(s).

These architectures have the benefit of providing somewhat scalable architectures since increasing the

Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000

number of web servers and application servers can accommodate increases in number of users. They also have the benefit of consolidating all data management in the database server; thus simplifying the management and administration of the corporate database and avoiding issues of replicated or partitioned data management. Even in geographically dispersed and heavily accessed Intranets, it is seldom the case that the corporate database is either replicated or partitioned at multiple sites because of the complexity of synchronizing, managing, backing up and archiving multiple copies of the corporate database. But, the centralization of data management poses scalability problems. As the number of users increases, the number of requests to the database server increases. Eventually, the throughput of the database server is reduced thus degrading performance or requiring the upgrade of the database server to a faster, larger, more expensive multi-processor machine. Even if throughput problems are resolved, response time remains a problem because each access to the database server requires network messages.

Application developers and application server vendors have recognized these problems and have used a number of approaches to reduce contention on the database server and improve performance. These approaches are described in section 3.1. Each of these approaches has some drawbacks as described in sections 3.1 and 3.2. What application servers need is a lightweight, high-performance DBMS that is easy to install and maintain, has a small-enough footprint to reside in the application tier, and provides the bridge between the application tier and the backend database server.

TimesTen's Front-Tier™ product [3,4] is an application tier data cache that provides the following functionality and advantages:

1. It caches frequently-used subsets of the corporate database in the application tier. This improves response time by reducing network overhead to the database server, and increases throughput by reducing contention on the database server.
2. It services data management requests in the application tier in an in-memory DBMS that delivers unprecedently low response time and high throughput.
3. It supports declarative SQL queries and updates through the standard ODBC and JDBC APIs [3,4,5]. This reduces development time as it offers the same familiar APIs as the database server.
4. It synchronizes updates to the cache with the corporate database.
5. It supports caching in multiple application servers with potentially overlapping caches. Overlapping caches provide high availability and enable load balancing. Non-overlapping caches are used to partition the workload among application servers.
6. It provides an infrastructure for detection and notification of failed replication components to support fail-over and auto-restart.
7. It does not modify the corporate database schema, thus preserving the advantages of centralized corporate data management and administration.

3. Data Caching with Front-Tier

3.1 Reducing Contention on the Database server

Application developers and application server vendors have used a number of approaches to reduce contention on the database server, and improve response time. These approaches include:

- *Result Set Caching*. This approach consists of caching some of the results obtained from the database server in the application tier, and reusing them if they happen to process requests identical to the ones whose results have already been cached. This caching technique is severely limited because:
  - ❖ It lacks query processing capability. The only queries that can be answered from the cache are queries whose results were previously cached.
  - ❖ The caches cannot be shared or updated. They are typically read-only caches with no concurrency control. Furthermore, multiple application servers running on the same machine do not typically share caches among themselves. The result is duplicate data that consumes resources unnecessarily on the application server.
- *Disk-Based RDBMS Replication*. With this approach, subsets of the corporate database are replicated in the application tier. While this approach does provide transaction management and complex query processing in the application tier, it is too heavy weight, because disk-based RDBMSs are fairly demanding in their resource requirements. Furthermore, they do not provide the performance advantages of in-memory data management.
- *Disk-Based RDBMS Partitioning*. This approach consists of partitioning the corporate database into several databases; each managed by a disk-based RDBMS that resides in the database tier. This approach reduces contention over a single database server by distributing it over several database servers, but presents several disadvantages:
  - ❖ Response time remains an issue, as each database access requires network messages.
  - ❖ The advantage of a consolidated repository of corporate data for analysis and report generation is lost.
  - ❖ The corporate database is harder to administer, as it becomes spread over several servers. Furthermore, scaling by adding more database servers poses the challenge of having to repartition the corporate database.
  - ❖ Some transactions may require distributed transaction management support and will suffer from its accompanying performance overhead.

3.2 Front-Tier: In-Memory Data Management in the Application Tier

Front-Tier manages an organization's frequently-used data in the application tier, using TimesTen's in-memory database technology. Unlike disk-optimized RDBMSs that have been designed to reduce disk I/O through the buffering of disk data in main memory, Front-Tier relies on the memory residence of data to eliminate much of the overhead associated with disk-based RDBMSs. For example, Front-Tier does not have to maintain, manage, or search a buffer pool since all data is always in memory. Similarly, Front-Tier uses index structures optimized to reduce CPU processing and memory consumption. The result is a system that is an order of magnitude faster than fully cached disk-based RDBMSs. In addition to high performance, Front-Tier offers the standard capabilities found in disk-based RDBMSs such as data sharing, transaction management and SQL functionality. Front-Tier's architecture is lightweight, easy to embed, and can be deployed on a range of application tiers and computing platforms. TimesTen Performance Software's in-memory database technology has been described in [1,2].

Managing frequently-accessed data in the application tier reduces contention on the backend database server thus improving overall throughput, and brings data close to the application, thus improving response time by avoiding network overhead. These advantages, coupled with the inherent performance advantages of in-memory data management, provide a powerful boost to response time, throughput, and overall system scalability. Front-Tier's approach has the added benefit of not modifying the schema of the corporate database, which can remain centralized and whole, thus permitting global data analysis and ease of administration.

The approaches described in section 3.1 and the Front-Tier approach are summarized in the following table, together with their advantages:

		Approach			
		Result set caching	Disk-Based RDBMS repl.	Disk-based RDBMS part.	Front-Tier
FEATURE / BENEFIT	SQL support	No	Yes	Yes	Yes
	Transaction mgmt support	No	Yes	Yes	Yes
	Lightweight	Yes	No	NA	Yes
	Ease of admin of corp DB	Yes	Yes	No	Yes
	Reduced resp time	Yes	Somewhat	No	Yes
	Improved Throughput	Yes	Yes	Yes	Yes

3.3 Defining the Content of a Cache

To define what is to be cached, Front-Tier provides a Web-based easy-to-use tool called the *Front-Tier Administrator*. Through the Front-Tier Administrator, the application designer may view the schema of a chosen corporate database. From that schema, the user chooses the sub-schema that should be cached using the concept of Cache Groups. A *Cache Group* is a set of Front-Tier tables that corresponds to a set of related and frequently used tables in the corporate database. SQL syntax is used to define Cache Groups and may be used to further qualify which columns and rows from a set of related tables belong to a cache. The Front-Tier Administrator assists the user in defining Cache Groups and automatically generates the appropriate SQL syntax. Users may also define Cache Groups programmatically using SQL syntax.

Example:

Assume that the following tables exist in the corporate database:

Customer (CustId, Name, Age, Gender,
StreetAddress, State, ZipCode, PhoneNo)
Order (CustId, OrderId, PurchaseDate, Amount)
CustInterest (CustId, Interest)

An application may want to cache the profiles of customers who have placed one or more purchase orders worth more than $500 since January 1, 2000. To that end, it may define the following two cache groups:

```
CREATE CACHE GROUP PacificCustomers
SELECT CustId, Name, Age, Gender, Interest
FROM Customer, Order, CustInterest
WHERE Customer.CustId = CustInterest.CustId
AND Customer.CustId = Order.CustId
AND Customer.State IN ('WA', 'OR', 'CA', 'NV')
AND Order.PurchaseDate >= 'JAN 1 2000'

CREATE CACHE GROUP MountainCustomers
SELECT CustId, Name, Age, Gender, Interest
FROM Customer, Order, CustInterest
WHERE Customer.CustId = CustInterest.CustId
AND Customer.CustId = Order.CustId
AND Customer.State IN
    ('MT', 'ID', 'UT', 'AZ', 'WY', 'CO', 'NM')
AND Order.PurchaseDate >= 'JAN 1 2000'
```

where the Cache Groups PacificCustomers and MountainCustomers are to be cached on different application servers.

Two tables will be cached in Front-Tier. They are:
Customer (CustId, Name, Age, Gender)
CustInterest (CustId, Interest)

They can be used to answer any queries over these tables for the columns listed above. (Note that there is no need to cache the Order table.)

An additional concept used by Front-Tier is that of a Cache Instance. A *Cache Instance* is a complex object or a collection of related records that are uniquely identifiable. Cache Instances form the unit of cache loading and cache aging as will be described in section 3.4. In the example above, all records in the Customer and CustInterest tables that belong to a given customer id belong to the same Cache Instance.

When data is cached into Front-Tier, types must be converted from the corporate database types to Front-Tier's data types. The Front-Tier Administrator assists the user in recommending the Front-Tier data types that most closely match the corporate database types.

3.4 Caching Data and Managing the Cache

Once a Cache Group has been defined, the data that it describes can be loaded all at once from the corporate database into Front-Tier for processing. Alternatively, Cache Instances may be faulted into Front-Tier, or loaded, on demand, from the corporate database. Data that has been loaded into Front-Tier is available for SQL processing through JDBC or ODBC. The user may choose to periodically refresh Cache Groups from the corporate database, unload Cache Groups, and/or load different Cache Groups. This may be accomplished programmatically while the application is running.

Note that Front-Tier enables developers to create indexes on cached data. Front-Tier indexes may match the indexes in the corporate database or may be different. The application designer can use the flexibility of Front-Tier to create multiple indexes on the same table and may define indexes over multiple columns.

Cache Instances are automatically aged out of the cache when the cache capacity is exceeded. Aging is based on last time of access, and uses an LRU scheme by default. In addition, Front-Tier provides applications with a number of cache-aging options. An application may set up different durations for different Cache Groups as well as for different Cache Instances. Furthermore, the application may specify that certain Cache Groups should never be aged out. For example, the application may want to keep catalog information in the cache all the time, while a user's profile is only relevant while the user is connected to the application.

3.4 Synchronization

During normal processing, applications read and update data cached in Front-Tier. Applications residing on the same machine can share caches. Furthermore, different caches of the same corporate database may reside on the same machine or on different machines. These caches may be identical or may contain different subsets of the corporate database. For example, an application tier may consist of several servers each dedicated to serve a subset of subscribers. The subscribers may be partitioned according to zip code, area code, user identifier, etc. With such a scheme, the data cached on each server will contain a different subset of the corporate database.

Read-only transactions do not require communication with the corporate database. However, when the application completes a transaction that has modified the database, Front-Tier first commits the transaction in the corporate database, and then in Front-Tier. This technique allows the corporate database to apply any required logic related to the data before it is committed in Front-Tier. As a result, the corporate RDBMS always reflects the latest image of the data.

Similarly, if the content of caches overlap in different application servers, Front-Tier's replication keeps the content of the caches consistent.

4. Conclusion

TimesTen Performance Software's Front-Tier product is an in-memory application-tier data cache targeted at high-performance, data-intensive Internet applications. In contrast to simple result cache mechanisms, Front-Tier can process new SQL queries over cached data. Front-Tier caches can be shared among different applications. Updates can be applied to the caches, and the caches are kept consistent with the backend corporate database. Front-Tier is also superior to disk-based RDBMS replication schemes that replicate parts of a corporate database in the application tier because it provides better performance and because it is lightweight and therefore requires fewer resources on the application server. Finally, Front-Tier is superior to schemes that partition the corporate database into multiple disk-based databases because it preserves the centralized management and administration of the corporate database.

By bringing data closer to the application, and by processing queries in an in-memory RDBMS, Front-Tier reduces response time significantly. By offloading some of the data processing work from the database server, Front-Tier improves overall throughput without interfering with the centralized management and administration of the corporate database.

4. References

[1] The TimesTen Team. In-Memory Data Management for Consumer Transactions: The TimesTen Approach. *Proc. of the Int. Conf. on Management of Data*, June 1999.

[2] The TimesTen Team. In-Memory Data Management in the Application Tier. *Proc. of the 16th Int. Conf. on Data Engineering*, February 2000.

[3] *Front-Tier JDBC Developer's Guide*. TimesTen Performance Software. http://www.timesten.com.

[4] *Front-Tier ODBC Developer's Guide*. TimesTen Performance Software. http://www.timesten.com.

[5] *Front-Tier SQL Reference Manual*. TimesTen Performance Software. http://www.timesten.com.