Category: Concepts, Trends, and Challenges

High Availability and Data Consistency for Three-Tier Enterprise Applications

Wenbing Zhao
Cleveland State University, USA

Louise E. Moser
University of California, Santa Barbara, USA

P. Michael Melliar-Smith
University of California, Santa Barbara, USA

INTRODUCTION

Enterprise applications, such as those for e-commerce and e-government, are becoming more and more critical to our economy and society. Such applications need to provide continuous service, 24 hours a day, 7 days a week. Any disruption in service, including both planned and unplanned downtime, can result in negative financial and social effects. Consequently, high availability and data consistency are critically important for enterprise applications.

Enterprise applications are typically implemented as three-tier applications. A three-tier application consists of clients in the front tier, servers that perform the business logic processing in the middle tier, and database systems that store the application data in the backend tier, as shown in Figure 1.

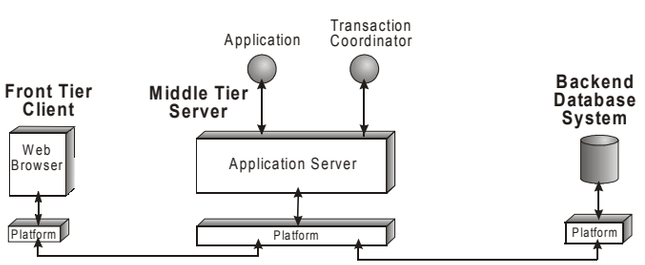
Within the middle tier, a server application typically uses a transaction processing programming model. When a server application receives a client's request, it initiates one or more transactions, which often are distributed transactions. When it finishes processing the request, the server application commits the transaction, stores the resulting state in the backend database, and returns the result to the client.

A fault in the middle tier might cause the abort of a transaction and/or prevent the client from knowing the outcome of the transaction. A fault in the backend tier has similar consequences. In some cases, the problems can be a lot worse. For example, a software design fault, or an inappropriate heuristic decision, might introduce inconsistency in the data stored in the database, which can take a long time to fix.

Two alternative recovery strategies, namely roll-backward and roll-forward, can be employed to tolerate and recover from a fault. In roll-backward recovery, the state of the application that has been modified by a set of unfinished operations is reversed by restoring it to a previous consistent state. This strategy is used in transaction processing systems. In roll-forward recovery, critical components, processes, or objects are replicated on multiple computers so that if one of the replicas fails, the other replicas continue to provide service, which enables the system to advance despite the fault. Many applications that require continuous availability take the roll-forward approach. Replication is commonly employed in the backend tier to increase the reliability of the database system.

There has been intense research (Frolund & Guerraoui, 2002; Zhao, Moser, & Melliar-Smith, 2005a) on the seamless integration of the roll-backward and roll-forward strategies in software infrastructures for three-tier enterprise applications, to achieve high availability and data consistency. *High availability* is a measure of the uptime

Figure 1. A three-tier enterprise application



of a system, and typically means five nines (99.999%) or better, which corresponds to 5.25 minutes of planned and unplanned downtime per year. *Data consistency* means that the application state stored in the database remains consistent after a transaction commits. Both transactions and replication require consistency, as the applications execute operations that change their states. Transactions require data consistency, and replication requires replica consistency.

BACKGROUND

Transaction Processing

A transaction is a set of operations on the application state (Gray & Reuter, 1993) that exhibit the following ACID properties:

- **Atomicity:** Either all the operations succeed in which case the transaction commits, or none of the operations is carried out in which case the transaction aborts.
- **Consistency:** If the application state is consistent at the beginning of a transaction, the application state remains consistent after the transaction commits.
- **Isolation:** One transaction does not read or overwrite intermediate results produced by another transaction—that is, the transactions appear to execute serially.
- **Durability:** The updates to the application state become permanent (or persist) once the transaction is committed, even if a fault occurs.

A transaction processing (TP) system typically consists of a TP monitor, communication channels, database servers, operating systems, and applications. A TP monitor provides tools, mechanisms, and application programming interfaces (APIs) to ease or automate the application programming, execution, and administration of the transactions.

When a transaction involves operations on more than one computer, it is called a *distributed transaction*. In a distributed transaction, an atomic commit protocol is necessary, so that the ACID properties hold for the state at all of the computers involved in the transaction. The most popular atomic commit protocol is the two-phase commit (2PC) protocol.

The 2PC protocol involves a transaction coordinator, which drives the protocol, and one or more transaction participants, which control the application state to be persisted. The 2PC protocol involves two phases of message passing. In the first phase, the coordinator sends a request to *prepare* to all of the participants. If a participant can successfully write its update to persistent storage, so that it can perform the update even in the presence of a fault, the participant responds to the coordinator with a "Yes" vote. At this point, the participant is *prepared*. If the coordinator collects "Yes" votes from all of the participants, it decides to *commit* the transaction. If the coordinator receives even a single "No" vote, or if a participant does not respond and is timed out, the coordinator decides to *abort* the transaction. In the second phase, the coordinator *notifies* the participants of its decision. Each participant then either commits or aborts the transaction locally and sends an acknowledgment to the coordinator. The coordinator can *forget* the transaction once it receives acknowledgments from all of the participants in the second phase.

Because of its simplicity and efficiency under fault-free conditions, the 2PC protocol has been adopted as the atomic commit protocol for many distributed transaction processing specifications, including the XOpen/XA specification (The Open Group, 1992) and the CORBA OTS specification (Object Management Group, 2000). The 2PC protocol is used by essentially every commercial TP monitor and database server for distributed transaction coordination within a single enterprise.

Replication

Component, process, or object replication, based on the virtual synchrony model (Birman & van Renesse, 1994) is often regarded as orthogonal to transaction processing. Critical components of a distributed system are replicated to achieve the required reliability and availability, so that the failure of one of the replicas will not bring down the entire system.

Active replication, passive replication, and semi-active replication are the most common replication strategies, which are defined as follows (Powell, 1991):

- **Active Replication:** All replicas perform exactly the same actions in the same order and output their results.
- **Passive Replication:** Only one replica (the primary) executes in response to the client's requests. Passive replication has two variations: warm passive replication and cold passive replication. In warm passive replication, the primary periodically checkpoints its state at the other executing replicas (the backups). In cold passive replication, the backups are not launched until the primary is detected to have failed.
- **Semi-Active Replication:** Semi-active replication is a hybrid of the active and passive replication strat-