Fault Tolerant Data Management for Cloud Services

C

Wenbing Zhao
Cleveland State University, USA

INTRODUCTION

The pervasiveness of cloud services has significantly increased the dependability requirement of cloud systems. Most cloud services are implemented according to a three-tier architecture where the presentation, application logic execution, and data management are separately handled by each tier (Zhao, Moser, and Melliar-Smith, 2005). The middle-tier servers implement the application logic and they are designed to be either stateless or to only maintain session state. Hence, this separation of concerns has greatly increased the scalability of such systems because the middle-tier servers can be easily scaled out. On the other hand, this design makes the data management tier ever more important because the availability and integrity of the services hinges on the data management tier. The data must be made highly available and protected against various hardware faults and malicious attacks.

While it is relatively straightforward to ensure high availability for Web servers and application servers by simply running multiple copies according to the three-tier architecture, it is not so for a database management system, which has abundant state. The subject of highly available database systems has been studied for more than two decades and there exist many alternative solutions (Agrawal et al., 1997; Cecchet, Candea, & Ailamaki 2008; Drake et al., 2005; Garcia, Rodrigues, & Preguiça, 2011; Kemme, & Alonso, 2000; Patino-Martinez et al., 2005). In this article, we provide an overview of two most popular database high availability strategies, namely database replication and database clustering. The emphasis is given to those that have been adopted and implemented by major database management systems (Banker 2011; Davies & Fisk, 2006; Vallath 2004).

BACKGROUND

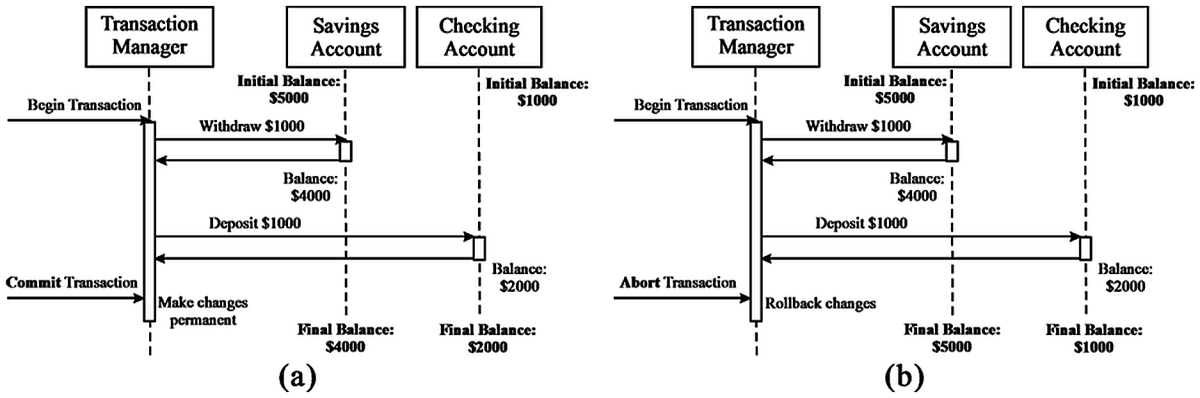
A database management system consists of a set of data and a number of processes that manage the data. These processes are often collectively referred to as database servers. The core programming model used in database management systems is called transaction processing. In this programming model, a group of read and write operations on the some data set are demarcated within a transaction. A transaction has the following ACID properties (Gray & Reuter, 1993):

- **Atomicity:** All operations on the data set agree on the same outcome. Either all the operations succeed (the transaction commits), or none of them are (the transaction aborts).
- **Consistency:** If the database is consistent at the beginning of a transaction, then the database remains consistent after the transaction commits.
- **Isolation:** A transaction does not read or overwrite a data item that has been accessed by another concurrent transaction.
- **Durability:** The update to the data set becomes permanent once the transaction is committed.

An example of an atomic transaction is shown in Figure 1. This transaction involves a debit

DOI: 10.4018/978-1-5225-2255-3.ch094

Figure 1. An example of atomic transactions



operation on a savings account and a credit operation on a checking account. If both operations are successful and the user decides to commit the transaction, the changes to both accounts are made permanent, as shown in Figure 1(a). On the other hand, if the user decides to abort the transaction, the changes to both accounts will be reversed so that the account balances of both accounts are restored to the values at the beginning of the transaction, as illustrated in Figure 1(b).

To support multiple concurrent users, a database management system uses sophisticated concurrency control algorithms to ensure the isolation of different transactions even if they access some shared data concurrently (Bernstein *et al.*, 1987). The strongest isolation can be achieved by imposing a serializable order on all conflicting read and write operations of a set of transactions so that the transactions appear to be executed sequentially. Two operations are said to be *conflicting* if both operations access the same data item and at least one of them is a write operation, and they belong to different transactions. Another popular isolation model is the snapshot isolation. Under the snapshot isolation model, a transaction performs its operations against a snapshot of the database taken at the start of the transaction. The transaction will be committed if the write operations do not conflict with any other transaction that has committed since the snapshot was taken. The snapshot isolation model can provide better concurrent execution than the serializable isolation model.

A major challenge in database replication, the basic method to achieve high availability, is that it is not acceptable to reduce the concurrency levels. This is in sharp contrast to the replication requirement in some other field, which often assumes that the replicas are single-threaded and deterministic (Castro & Liskov, 2002).

To achieve high availability, a database system must try to maximize the time to operate correctly without a fault and minimize the time to recover from a fault. The transaction-processing model used in database management systems has some degree of fault tolerance in that a fault normally cannot corrupt the integrity of the database. If a fault occurs, all ongoing transactions will be aborted on recovery. However, the recovery time would be too long to satisfy the high availability requirement. To effectively minimize the recovery time, redundant hardware and software must be used. Many types of hardware fault can in fact be masked. For example, power failures can be masked by using redundant power supplies, and local communication system failures can be masked by using redundant network interface cards, cables and switches. Storage medium failures can be masked by using RAID (redundant array of inexpensive disks) or similar techniques.