ACID Properties in Distributed Databases

Shiwei Yu

Advanced eBusiness Transactions for B2B-Collaborations

Abstract

A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. There are 2 processes in distributed database, replication and duplication. Replication using specialized software to look for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. Duplication will identify one database as master and then duplicate it. Upon those processes, it seems data in database will be very consistent and will have no problem when some transaction is executed. But in fact, if there is no definition on transaction, the database will be in a mess condition. So how does the database stay consistent and stable? The answer for this is ACID properties, which are a set of properties that guarantee the reliability of database transactions. In this paper, we make the case that transactions will cause a fault in a distributed database without ACID properties. It will raise the need of some methods for the transactions like timestamp, commit and recovery protocols, 2-phase commit, 2-phase locking and a replication protocol to keep the database in order. In addition, we will also see the failure situation and the recovery method when a database meets a fault or error such as Crash faults, Omission faults and Timing faults. However, if the database is not a distributed database but heterogeneous database, ACID properties can also be applied by establishing the global data model or global external view.

1 Introduction

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. Like the central database system, it is necessary to ensure the data in a distributed database should be integrity so that all users who access the database can get the correct data and run their program without any error. To address this issue, we can use ACID [1] properties for distributed database. The

definition of ACID properties for distributed database is that it is a set of properties that guarantee the reliability of database transactions. ACID defines properties that traditional transaction must display; they are Atomicity [1], Consistency [1], Isolation [1], and Durability [1]. However, since they are initially developed with traditional, business-oriented applications like banking, so they not fully support those applications which need to lock the resource for a long time. Overall, there are several methods which will help to make ACID properties work properly in the database. Certainly, some transactions will have a fault, error or even failure due to different situation. Then how to deal with this problem? To avoid those faults, the best way is to use error detection for distributed database. Below let's look at the structure for this paper.

The transaction concept and the implementation method for ACID properties are presented in Section 2. In section 3, we present transaction failure situation, and how the error is recovered. Section 4 illustrates how ACID properties are performed in heterogeneous database. Section 5 concludes with a summary and discusses future steps.

2 Transaction in distributed database

In the database, all operations are completed by transactions. In this section, we will see the transaction concept and how transactions follow ACID properties.

2.1 Transaction concept

In a traditional way, a transaction is an agreement between a buyer and a seller to exchange an asset for payment. This is the definition in a business sense [6]. In a database view point, it can be understand as a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. [7]The transaction in a database should have two purposes. First, is to provide reliable units of work that allow a correct recovery from failure. Second, is to provide isolation between

programs to help them access database concurrently. That is, a transaction in a database must have ACID properties to run the program correctly. In a distributed database, transactions are implemented over multiple applications and hosts. Moreover, distributed transactions also enforce the ACID properties over multiple data stores.

2.2 ACID implementation

As mentioned in Section1, ACID properties are Atomicity, Consistency, Isolation and Durability. Each transaction must follow these 4 properties. In other words, they are implemented by using some method for the transaction.

2.2.1 Atomicity

Atomicity refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them are. Atomicity states that database modifications must follow an “all or nothing” rule. If some part of a transaction fails, then the entire transaction fails, and vice versa.

To implement atomicity, there are 2 parts, one is for index and value, the other is for commit method. The methods for index and value are using a private workspace [11] [12] and writeahead log [11] [12]. These two methods are to ensure that the original index and value can be found before the result value is committed to the database. On the contrary, a commit method is well known as 2-Phase Commit [11] [12], this commit method is to guarantee that the change to database is atomic, that is, commit or abort. Let’s see private workspace and writeahead log first.

Private workspace (Figure 1) states that if we need to modify the data, we don’t modify it directly on its original data index, but to create a new workspace for modifying it. At the same time, map the unchanged data to the data block and occupy new data block with changed data. Before the new data is committed to the database, both original data and new data will exist.

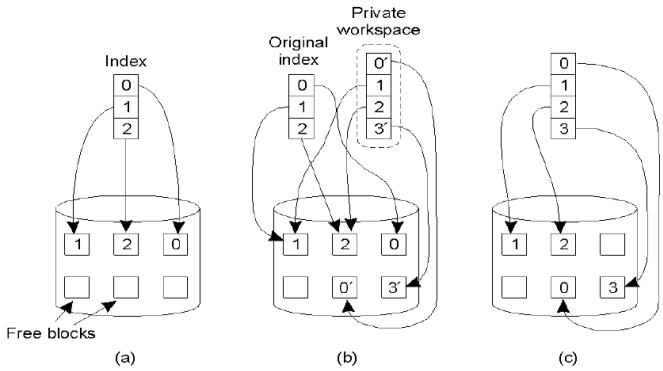

Figure 1

In Figure 1, (a) refers to the file index and disk blocks for a three-block file. (b) is the situation after a transaction has modified block 0 and appended block 3. (c) is the block situation after committing.

Like private workspace, writeahead log also use the log method to ensure the original index and value can be found even when the transaction begins to work, namely, the index and value are all in memory. The log just records the index and value before each statement is executed. Once the transaction fails, the log can help to restore the original index and value. Figure2 illustrates the writeahead log method.

x = 0;	Log	Log	Log
y = 0;			
BEGIN_TRANSACTION;			
x = x + 1;	[x = 0 / 1]	[x = 0 / 1]	[x = 0 / 1]
y = y + 2		[y = 0/2]	[y = 0/2]
x = y * y;			[x = 1/4]
END_TRANSACTION;			
(a)	(b)	(c)	(d)

Figure 2

In Figure 2, (a) is each step of transaction. (b) log the value of x both before and after x changes. (c) and (d) log each step of transaction (a).

After the data is changed by using a private workspace and writeahead log, the 2-Phase commit protocol [11] [12] is implemented to guarantee the atomicity of a

transaction. In 2-Phase Commit, there exist 2 roles, coordinator and participant. So in a distributed database, one node is the coordinator that is regard as the master site, the rest nodes are participants. When a transaction ends, the coordinator asks all participants if they are prepare to abort or commit. If participants agree to commit, it will change the records and its status in the database. Otherwise, the coordinator aborts and tells all nodes to reverse to the original data. The commit phase can be described as below [12],

1. If the coordinator received an agreement message from all participants during the commit-request phase:

a. The coordinator sends a commit message to all the participants.

b. Each participant completes the operation, and releases all the locks and resources held during the transaction.

c. Each participant sends an acknowledgment to the coordinator.

d. The coordinator completes the transaction when the acknowledgments have been received.

2. If any participant sent an abort message during the commit-request phase:

a. The coordinator sends a rollback message to all the participants.

b. Each participant undoes the transaction using the undo log, and releases the resources and locks held during the transaction.

c. Each participant sends an acknowledgement to the coordinator.

d. The coordinator undoes the transaction when all acknowledgements have been received.

By using the private workspace, the writeahead log and the 2-Phase commit, a transaction is said to be Atomic.

2.2.2 Consistency

The Consistency property ensures that the database remains in a consistent state, despite the transaction succeeding or failing and both before the start of the transaction and after the transaction is over.

To implement consistency in a distributed database, there is a method named serializability [12]. This method means when transactions run concurrently, the result is the same as if it runs in serial. Here we assume the operation in database is only read/write, for read refers to read data from the database and write refers to modify data to the database. If two operations are going to access a same data item like read-write (one operation to read and another operation to write), write-read and write-write, then only one operation can access the data in order to keep the data consistently. In this case, if the two operations access to a same data at the same time, it is said that the second operation is conflict with the first one because only one operation can access the data. Since each operation also needs to be atomic by itself, the data should be locked when it is operated by some operation, it is part of Isolation properties and we will discuss it in Section 2.2.3. Now we focus on how the database keeps consistency. Concurrency control of a database is based on timestamp ordering [12]. Each transaction is assigned a unique timestamp value when it start. So the request of operation on data can be totally ordered according to the timestamp. In timestamp ordering, the transaction operation is usually checked by the following rules [12],

No	Transaction 1	Transaction 2	Rule
1	Write	Read	Transaction 1 must not write an object that has been read by any Transaction 2 when Transaction 1 occurs later than Transaction 2. This requires that Transaction 1 is later than the maximum read timestamp of the object.

2	Write	Write	Transaction 1 must not write an object that has been written by any Transaction 2 when Transaction 1 occurs later than Transaction 2. This requires that Transaction1 is later than write timestamp of the committed object.
3	Read	Write	Transaction1 must not read an object that has been written by any Transaction2 where Transaction 2 occurs later than Transaction 2 occurs later than Transaction 1. This requires that Transaction 2 is later than write timestamp of the committed object.

Table 1

According to these rules, the transaction can be executed to make the database in a consistent condition, even some execution fails. This situation can refer to the atomicity property. When a transaction fails, it can send abort message to all participants and reverse the data to the previous correct state.

2.2.3 Isolation

Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction. As discussed above, the Isolation property can helps to implement concurrency of database.

To guarantee Isolation, 2-Phase Locking [11] [12] is typically used in distributed database. Generally speaking, the definition of 2-Phase Locking is that when a resource is accessed by a transaction, it is locked. Once another transaction needs to access it, the existing lock for the resource and the type of accessing would be check by the system. If the resource already locked by some transaction, it is not visible to the transaction which acquire the lock to it until the previous transaction commit or

abort and release the lock for it. We again assume that two types of access method is used,

Write-lock is associated with a database object by a transaction (the transaction locks it; acquires lock for it) before writing this resource.

Read-lock is associated with a database object by a transaction before reading this resource.

In 2-Phase Locking, it usually contains 2 phase, first is the growing of the lock phase, second is shrinking of the lock phase. The first phase is to acquire lock to the resource. When several transactions turn to the ready state, they acquire locks for the corresponding resource to access them. After the resource is locked, the transactions determine a synchronization point which likes atomic commit point to commit or abort. During the progress, while the transaction commits or aborts, the lock is released. In phase two, locks will be released one by one.

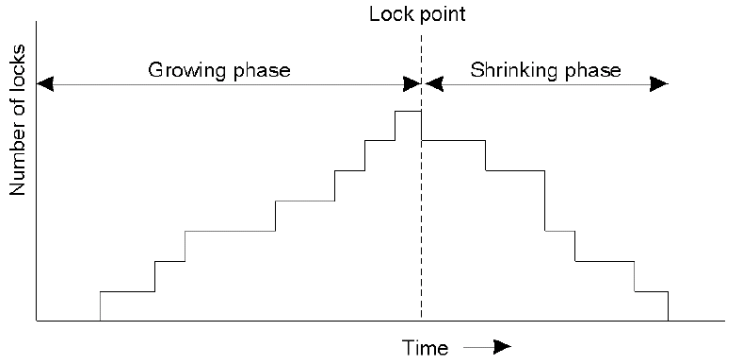

Figure 3

Except 2-Phase Locking, there exist Strict 2-Phase Locking [12]. It is a subclass of 2-Phase Locking. The difference between 2-Phase Locking and Strict 2-Phase Locking is that all locks are released at the same time in phase 2 in Strict 2-Phase Locking, which means, no matter how resource is processed by transactions. Only after all the transactions decide to either commit or abort, the resource will be released

and visible to other transactions. Figure 4 presents the Strict 2-Phase Locking procedure.

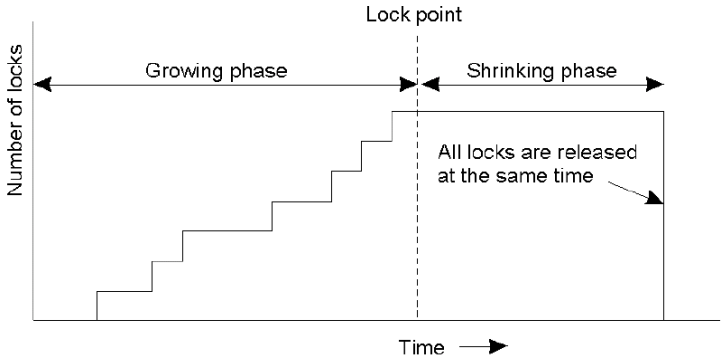

Figure 4

With the help of 2-Phase Locking and Strict 2-Phase Locking, the Isolation property is implemented so that each transaction is executed alone, the effects of a transaction are invisible to other concurrent transactions until that transaction is committed or aborted.

2.2.4 Durability

Durability states that once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent failures. That means when users are notified of success, the transactions will be persist, not be undone and survive from system failure.

The implementation for Durability is usually by using writing all transactions into a transaction log in case that system meets with failure. Usually, the log is a file and stored in some stable storage. Once the transaction crash or the hardware failure occurs, the database management system reviews the database logs for uncommitted transactions and rolls back the changes made by these transactions. Additionally, all transactions that are already committed but whose changes were not yet materialized in the database are re-applied.

Typically, the database log are made up by log sequence number, transaction id number, type and the information about the change that triggered the log record to be written.

3 Transaction failure and recovery

Although we have the modern facilities and service, it is still hard to ensure that all transactions will be a success due to many kinds of failure, error and fault. Once a transaction meet a failure, how to recover it from an error status to a correct status or formal status. In this section, we will talk about transaction failure and recovery. First let's look at the fault classification.

3.1 Fault classifications

The fault of transaction can be divided into the transient and the permanent fault. A transient fault means it will back to normal condition after some time while permanent fault means some hardware failure. As mentioned, permanent fault seem much severe than transient fault, but it is much difficult to check out permanent fault than transient one because transient will recover without any apparent intervention.

Based on this, we can give a classification to the transaction fault [10],

1. Crash faults: which indicate the hardware may meet a failure and completely stop running unless the engineer changes a new component.

2. Omission faults: which means some component fails to perform its service.

3. Timing faults: which point out the component complete the service overtime.

3.2 Fault recovery

In general, the procedure of fault recovery is error detection, damage confinement and error recovery.

Usually, error detection consists of replication check [10], timing check [10] and run-time constraints check [10]. The replication check refers to multiple replicas of a component perform a service simultaneously, their output will be compared and once they don't have the same result, it indicates there is an error in some components. Timing check is especially for time fault like the late response, time out problem. When transaction start, a timer is set to have a point which is the expiry time for the service to complete, if the service is completed before the expiry, timer will be canceled, otherwise, it is a timing error. Then run-time constraints check is the one that such as boundary values of variables not being exceeded are checked at run time, it is supported by some programming language.

Damage confinement is used to determine which component has what kind of error. As soon as it is known, the component will be isolated until it is isolated.

Error recovery means to recover the error from an error state to a valid state. In this part, there are two general approaches, backward error recovery [11] and forward error recovery [11]. In backward error recovery, the system is restored to a previous known valid state. This requires checkpoint [11] the system state and, once an error is detected, roll back the system state to the last checkpoint state. Clearly, this can be a very expensive way of recovery, not only is it necessary to keep copies of previous states, but also it is necessary to stop the operation of the system during checkpoint to ensure that the state that is stored is consistent. But in forward error recovery, it seems more appropriate; this method will drive the system from an erroneous state to a new valid state. However, it also has the difficulty, unless the fault is already known which caused the error, or it will be in a tricky situation.

Let's take 2-Phase Commit as an example. As we know, the diagram of 2-Phase Commit can be draw as below,

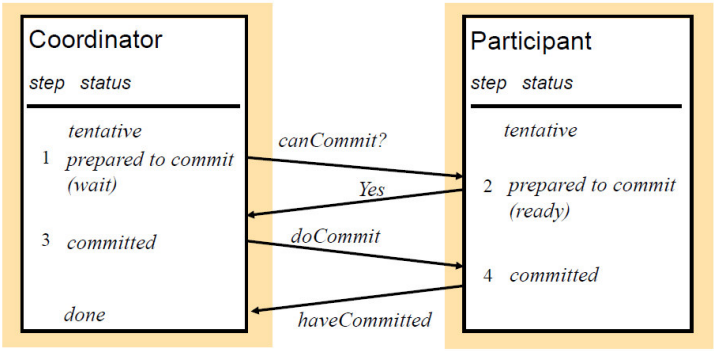

Figure 5

Figure clearly state that in 2-Phase Commit, coordinator and participant need to have 2 commit commands to finish the transaction successfully. But how detail does the coordinator and participant handle the error when the commit command fails to deliver? Below is action example [11] [12],

actions by coordinator:	actions by participant:

while START _2-Phase Commit to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
wait for any incoming vote;
if timeout {
write GLOBAL_ABORT to local log;
multicast GLOBAL_ABORT to all participants;
exit;
}
record vote;
	write INIT to local log;
	wait for VOTE_REQUEST from coordinator;
	if timeout {
	write VOTE_ABORT to local log;
	exit;
	}
	if participant votes COMMIT {
	write VOTE_COMMIT to local log;
	send VOTE_COMMIT to coordinator;
	wait for DECISION from coordinator;
	if timeout {

}
if all participants sent VOTE_COMMIT and coordinator votes
COMMIT{
write GLOBAL_COMMIT to local log;
multicast GLOBAL_COMMIT to all participants;
} else {
write GLOBAL_ABORT to local log;
multicast GLOBAL_ABORT to all participants;
}
	multicast DECISION_REQUEST to other participants;
	wait until DECISION is received;
	write DECISION to local log;
	}
	if DECISION == GLOBAL_COMMIT
	write GLOBAL_COMMIT to local log;
	else if DECISION == GLOBAL_ABORT
	write GLOBAL_ABORT to local log;
	} else {
	write VOTE_ABORT to local log;
	send VOTE ABORT to coordinator;
	}

Table 2

As we can see from Table 2, in the 2-Phase Commit, both coordinator and participate have a syntax “if time out…” this means the timing check is implemented to check whether the response time from coordinator or participate is expire, if it is time out, they may multicast the message abort to others and then restore the data to the valid state. By this check method, it may indicate either node may occur a crash, omission or time fault.

4 ACID in heterogeneous database

All upper discussions raised are about the ACID properties of the transaction in the distributed database, but there are several kinds of database exist. How do transactions perform ACID in Heterogeneous database?

Heterogeneous database is a collection of multiple database systems, it may contain different kind of database systems in which their database model, structure, syntax may different and may based on different operation systems. According to ACID properties, in every database, they should have properties of Atomicity, Consistency, Isolation and Durability. But when they merge together, can the ACID properties still remain?

Having a global data model and some kinds of connection between these different databases is a solution for the Heterogeneous database. Firstly, global data model means there is a global transaction control unit in database system to deal with all transactions for different kinds of databases. If applying this model, Heterogeneous database can be regard as a whole unit which is controlled by a transaction unit. It is just like central database is controlled by its DBMS. In addition, the connection method for different kinds of databases is also important, for example, JDBC [13] [14] can be used as the connection function for different kinds of databases. The API in JDBC defines the class, database connection, SQL syntax, and database structure etc. It allows Java code to use different driver for different databases. So, different kinds of databases can be connected together to use one interface for the transaction. The architecture of Heterogeneous can be described as below,

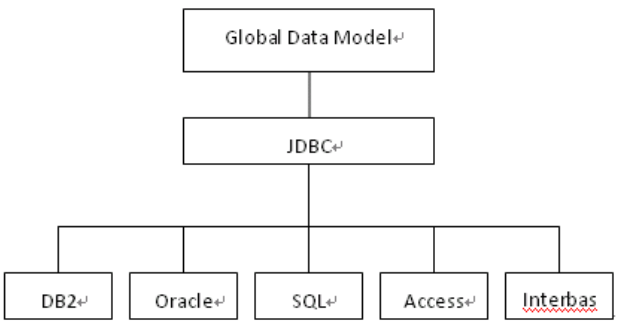

Figure 6

5 Conclusion

In this paper, we mainly discuss ACID properties, its implementation and recovery in distributed database. Further, we also go through Heterogeneous databases and the solution for perform ACID properties in it.

It is really important for database to have the ACID properties to perform Atomicity, Consistency, Isolation and Durability in transactions. It ensures all data in scientific research and business field to be a correct and valid statuse, without them, database will be in a mess.

References:

[1] Brahim Medjahed, Mourad Ouzzani, Ahmed K. Elmagarmid. Generalization of ACID Properties

[2] M. Tamer Özsu. DISTRIBUTED DATABASES

[3] Panayiotis K. Chrysanthis, Krithi Ramamritham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior (1990)

[4] A. Biliris, S. Dar, N. Gehani, H, V. Jagadish K. Ramamritham,(1994). ASSET: A System for Supporting Extended Transactions.

[5] Panayaotas K Chrysanthzs Krztha Ramamntham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior

[6] Gray, Jim (September 1981). "The Transaction Concept: Virtues and Limitations

[7] Gray, Jim; and Reuter, Andreas; Distributed Transaction Processing: Concepts and Techniques, Morgan Kaufman, 1993

[8] M. T. Ozsu and P. Valduriez, principles of Distributed Databases (2nd edition), Prentice-Hall, ISBN 0-13-659707-6

[9] Jalote, P. Fault Tolerance in Distributed Systems, (Prentice Hall, 1994).

[10] Goyer, P., P. Momtahan, and B. Selic. "A Fault-Tolerant Strategy for Hierarchical Control in Distributed Computer Systems," Proc. 20th IEEE Symp. on Fault-Tolerant Computing Systems (FTCS20), (IEEE CS Press, 1990), pp. 290-297.

[11] Tanenbaum, van Steen: Distributed Systems, Principles and Paradigms; Prentice Hall 2002

[12] Coulouris, Dollimore, Kindberg: Distributed Systems, Concepts and Design; Addison-Wesley 2005

[13] Chad Ferguson, Sandra Brey, SAS Institute Inc., Cary, NC. Developing Data-Driven Applications Using JDBC and Java Servlet/JSP Technologies

[14] The ENSIVA Data Server Transcending the JDBC/ODBC Bottleneck. WebSci Technologies, Inc.